Animating Multiple LED Backpacks

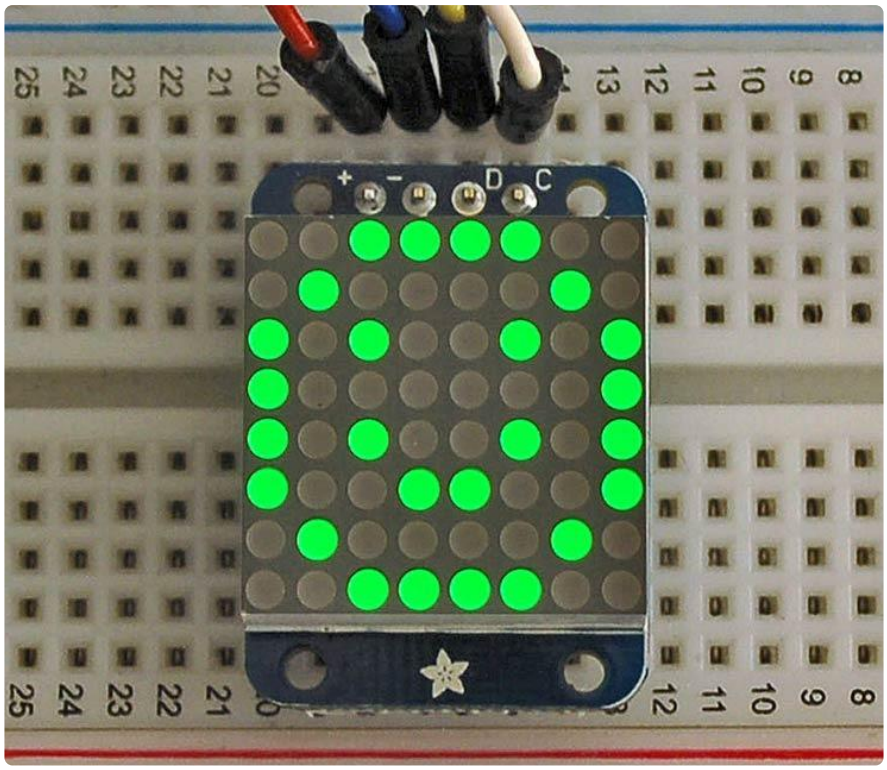
Created by Phillip Burgess

https://learn.adafruit.com/animating-multiple-led-backpacks

Last updated on 2023-08-29 02:12:24 PM EDT

Table of Contents

Overview



Adafruit LED backpacks make it incredibly simple to add small bitmapped displays to a project. Each requires just four wires: power, ground and two communication lines to a microcontroller.

Two common questions we receive:

1. How can I use more than one 8x8 matrix backpack in an Arduino sketch?
2. Can I have two (or more) matrices always showing the same image?

This tutorial demonstrates both.

We've had spooky Halloween displays on the brain lately, but the concepts here are equally applicable to more innocent schemes!

Wiring

These LED backpack displays use the Arduino's Two Wire Interface (or TWI — though you'll also sometimes see this called I^2C, Philips' trademarked name for the company's original implementation).

TWI allows multiple devices to communicate on the same bus. Just two wires are required...you don't need to dedicate Arduino pins to every separate device. This takes place on analog pins 4 (serial data) and 5 (serial clock). Newer "R3" Arduino boards also include distinct SDA and SCL pins serving the same functions, though analog pins 4 and 5 still work too.

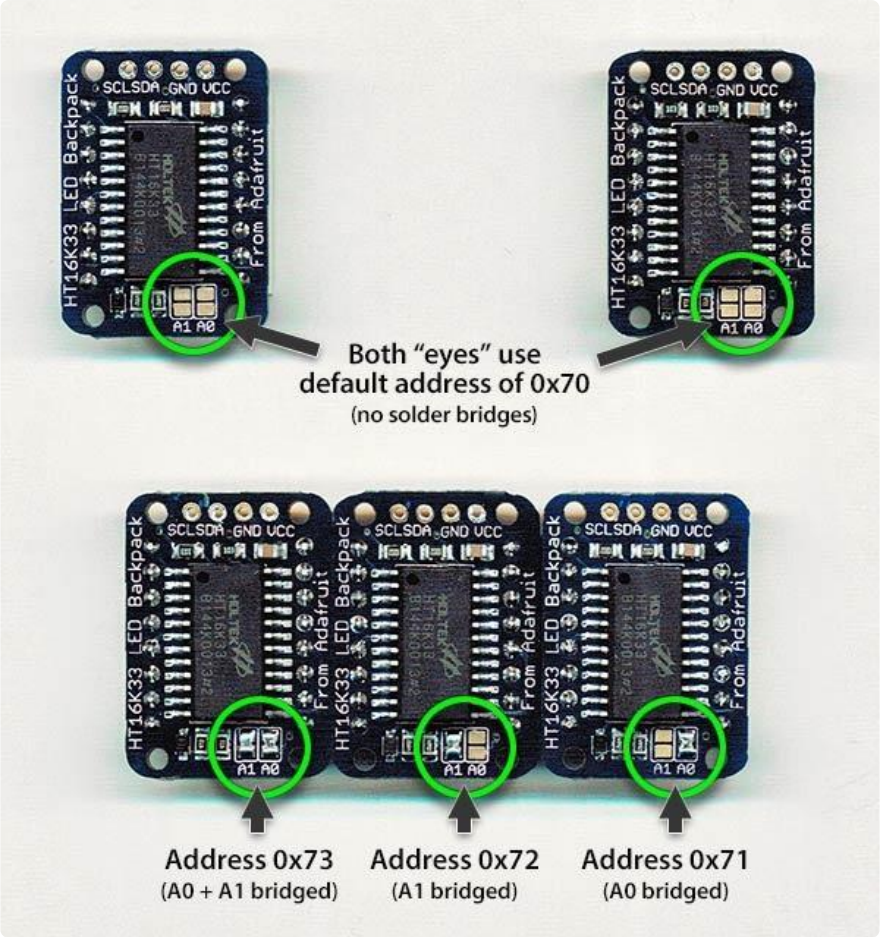
In order to distinguish among multiple devices on the bus, each must be assigned a unique numeric address. Most devices (sensors, etc.) are factory-configured for one specific address, but others (like these LED backpacks) are at least partially configurable. There's a default address (0x70 in hexadecimal notation) which can then be tweaked by joining some solder pads on the back of the board:

Bridging the two copper pads in the “A0” box with a blob of solder will increase this backpack’s address by one (so it’s now 0x71 instead of 0x70). Bridging the “A1” box will add 2, and both can be used in combination for +3. This allows four possible addresses on the “mini” 8x8 matrices. The “small” matrices add an “A2” address selection which adds 4 to the device’s address (again usable in combination, so there’s a total of eight possible addresses with the small matrices).

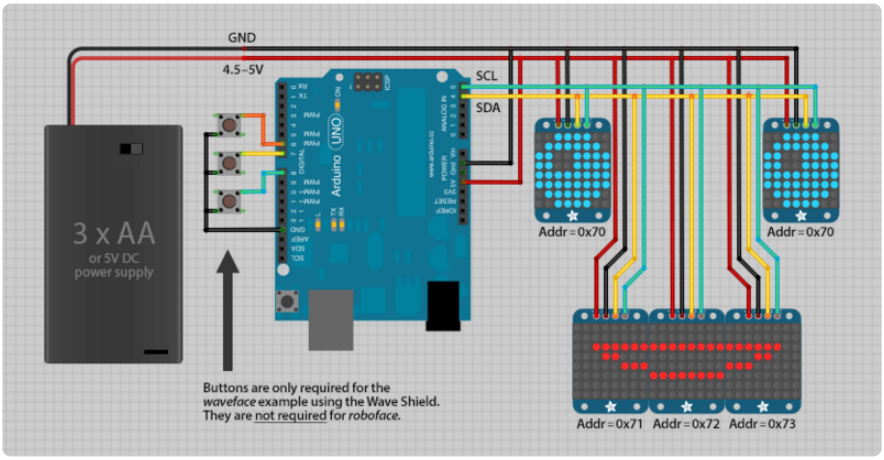
So! For a project, let’s suppose we want to animate a face. We plan to use a mini 8x8 matrix for each eye, and three side-by-side to form a mouth.

But there’s only four addresses, and we have five matrices! So we’ll use a small compromise here: both eyes will always point the same direction and blink together… no winking or crossed eyes. To do this, we just use the same address for both — they’ll receive and display the exact same data in unison. (And that’s all there is to that…sorry if you were expecting a more technical procedure!)

The three mouth matrices are then each assigned a unique address, separate from the “eyes” address. Here’s the view from the back:



Flipping this over then, our wiring diagram will look something like the following. Notice that all five displays connect to the same four points for power, ground, serial clock and data:



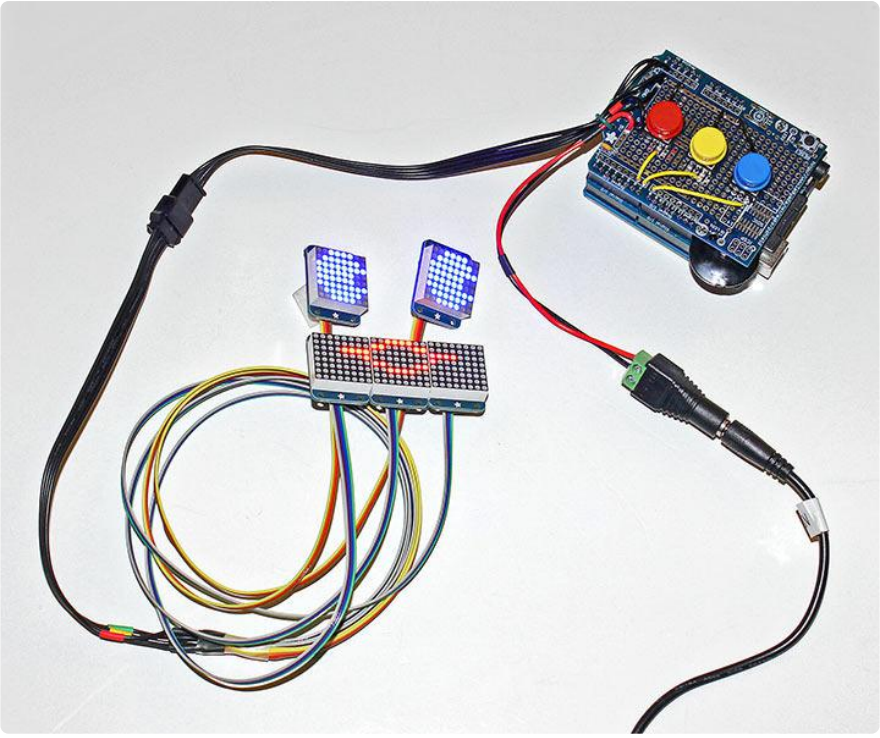
(click to embiggen)

We've illustrated it with a 3x AA battery pack for portable use, but for a stationary installation you can also use a regulated 5 Volt power supply (http://adafru.it/276). Thi s connects to the Arduino's 5V pin, bypassing the voltage regulator, so make absolute ly certain you're using a 5V supply! (Three alkaline AA cells only provide 4.5 Volts, but this is enough to keep the Arduino happy.)

Notice we're not using the DC power jack on the Arduino. Each of these displays can draw up to 200 milliamps (when fully lit), and the 5V regulator on the Arduino is only rated up to 500 mA. It could handle one or two of these displays (e.g. if only using the eyes), but the full face requires more current than this.

The buttons are optional — one of the example sketches ("wavface") works together with a Wave Shield (http://adafru.it/94), and these are used to trigger sounds. A simpler example ("roboface") does not require the Wave Shield or the buttons.

We wanted to test this in a lot of different settings, so we assembled an extra-durable rig with all the parts and wiring permanently soldered to a Proto Shield (http://adafru.it/51). But you don't need to go all-out like this if you don't want to:



Each matrix has about 18 inches of 4-conductor ribbon cable attached. The other ends of the ribbon cables join up at a 4-conductor JST plug cable (http://adafru.it/578), and a mating receptacle (http://adafru.it/579) connects to the shield. Having a point of separation makes it easier to install and remove in different fixtures.

The three mouth matrices were tacked together with a bit of hot-melt glue. For the wires, an unused mounting hole on the Proto Shield was used as a strain relief attachment point, so anything pulling on the wires will put pressure here and not on the solder connections.

Software

If this is your first introduction to the Adafruit LED backpacks, you should work through the LED Backpack tutorial () first. Get a single matrix up and running, confirm that code and wiring are all correct before advancing to this “level 2” project.

You’ll need to download and install the Adafruit_GFX, Adafruit_BusIO and Adafruit_LEDBackpack libraries. We have a tutorial explaining how libraries are installed ().

The backpack library already includes the example code for animating the face (“roboface” and “wavface”), these don’t need to be separately downloaded.

If using the Wave Shield example (“wavface”), you’ll also need to download and install the WaveHC library (). It’s imperative then that you work through the original Wave Shield tutorial () before moving on to this more advanced project. There are many separate parts, and a misstep with any one of them can stop the whole system from working. Testing the Wave Shield first lets you know that the shield is properly assembled, the SD card properly formatted and so forth. (If you’re not using the Wave Shield then you can bypass this step.)

Programming for Multiple Matrices

Whether you’re working with one or with several LED backpacks, you’ll need to #include three header files at the top of your sketch: one for the standard Arduino TWI library, and one each for the Adafruit device and graphics libraries:

```
#include &lt;Wire.h&gt;
#include "Adafruit_LEDBackpack.h"
#include "Adafruit_GFX.h"
```

The “matrix8x8” example sketch (included with the Adafruit_LEDBackpack library) is written for a single LED matrix. That code declares a single Adafruit_8x8matrix object like so:

```
Adafruit_8x8matrix matrix = Adafruit_8x8matrix();
```

That single object is then later referred to by its name (“matrix”) for graphics and other operations:

```
  matrix.begin(0x70);
  matrix.drawRect(0, 0, 8, 8, LED_ON);
```

With multiple matrices, we could declare each matrix object with a distinct name:

```
Adafruit_8x8matrix matrixOne   = Adafruit_8x8matrix();
Adafruit_8x8matrix matrixTwo   = Adafruit_8x8matrix();
Adafruit_8x8matrix matrixThree = Adafruit_8x8matrix();
...
```

But we'll usually find it easier to declare an object array. This is what the "roboface" example sketch does. There are four unique matrix addresses (remember, the two eyes share the same address), so we declare a four-element array:

```
Adafruit_8x8matrix matrix[4];
```

Each object can then be instantiated and initialized in the setup() function (using a loop if we like):

```
  for(uint8_t i=0; i&lt;4; i++) {
    matrix[i] = Adafruit_8x8matrix();
    matrix[i].begin(0x70 + i);
  }
```

("roboface" uses a table for the device addresses, but the principle is the same.)

To issue commands to a specific matrix, we follow the array name ("matrix") with an index — the element number in the array (a four-element array has indices 0 through 3). The syntax and parameters are otherwise the same as the single-matrix example. Here we issue different commands to each of four matrices in an array:

```
  matrix[0].drawPixel(0, 0, LED_ON);
  matrix[1].drawLine(0, 0, 7, 7, LED_ON);
  matrix[2].drawRect(0, 0, 8, 8, LED_ON);
  matrix[3].fillRect(2, 2, 4, 4, LED_ON);
```

Don't forget to refresh each display! Can use a loop if we like:

```
  for(uint8_t i=0; i&lt;4; i++) {
    matrix[i].writeDisplay();
  }
```

Animating the Face

The "roboface" example simply flips among a set of six mouth images randomly. "wavface" is synchronized to one of three voice clips played by the Wave Shield (these are in the "wavs" folder — copy them to an SD card). There is no "magic" here, no decoding of the speech or whatnot...the timing and mouth positions were simply determined manually through some educated trial and error, same way traditional

animators do it.

Six mouth images seems like it should be tremendously limiting, but together with the dialogue this coarse animation is enough to fool the eye. Here's a great tutorial explaining the principle ().

Bitmaps

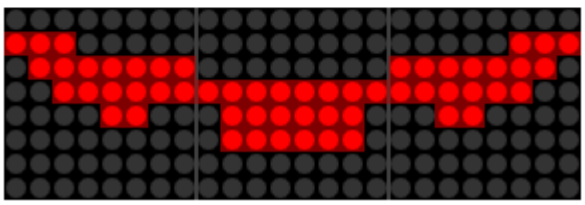
The face is drawn using a series of stored bitmaps rather than having a lot of code drawing lines, circles, etc. The only exception are the moving pupils, which are simply 2x2 pixel black squares.

The Arduino binary number notation simplifies the task of creating these shapes; you can practically see the image right in the code. For example, here's one frame of a Jack-o'-lantern mouth:

```
static uint8_t PROGMEM
  mouthImg[][24] = {
  { B00000000, B00000000, B00000000,
    B11100000, B00000000, B00000111,
    B01111111, B00000000, B11111110,
    B00111111, B11111111, B11111100,
    B00001100, B01111110, B00110000,
    B00000000, B01111110, B00000000,
    B00000000, B00000000, B00000000,
    B00000000, B00000000, B00000000 },
    // ...Subsequent bitmaps go here...
  };
```

Each "1" represents an "on" pixel, and "0" is off. Do you see it?

```
static uint8_t PROGMEM
  mouthImg[][24] = {
  { B00000000, B00000000, B00000000,
    B11100000, B00000000, B00000111,
    B01111111, B00000000, B11111110,
    B00111111, B11111111, B11111100,
    B00001100, B01111110, B00110000,
    B00000000, B01111110, B00000000,
    B00000000, B00000000, B00000000,
    B00000000, B00000000, B00000000 },
    // ...Subsequent bitmaps go here...
  };
```



A few things to keep in mind with bitmaps:

- The binary number notation (numbers starting with “B”) requires 8 digits…no longer, no shorter.
- By extension, a bitmap therefore must be some multiple of 8 pixels wide (e.g. 24 in the example above). You can pad an image with extra unused zeros at the right to fill out the multiple-of-8 rule.
- Bitmap data must be declared with the PROGMEM () directive. This stores data in the Arduino’s flash program memory rather than using scarce RAM.

The image on each matrix is sideways!

“Small” (1.2") and “mini” (0.8") matrices have different default orientations. With small displays, one line in the code (commented out by default) must be enabled (remove the leading slashes “//”), around line 180:

```
    matrix[i].setRotation(3);
```

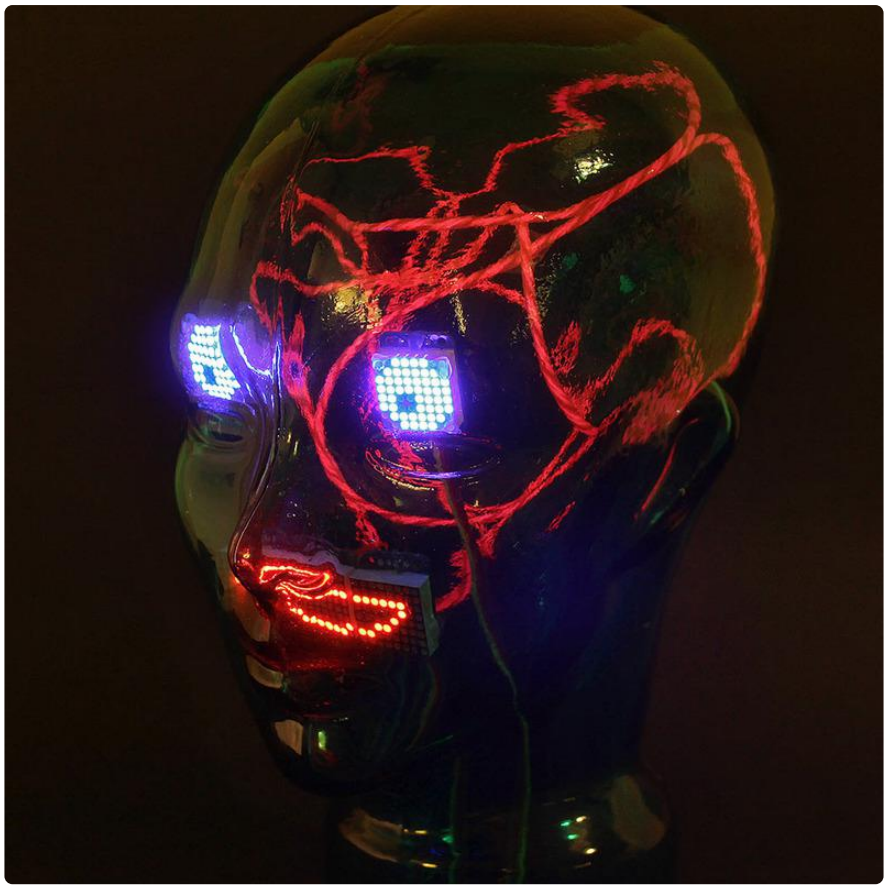
Ideas

Craft stores have these wonderful realistic styrofoam skulls around Halloween time. Here we’re using LED matrices for just the eyes, no mouth (the “roboface” sketch will work all the same). We simply carved out the eye sockets to the back and pushed the LED matrices through. The eyes peer around and blink:

Level up: add a joystick to move the eyes...or if you're really committed, use a webcam or a Kinect sensor on a PC (connecting to the Arduino through USB) to make the eyes automatically follow victims around the room.

This Jack-o'-lantern (another craft store find made of foam) uses all five matrices (eyes and mouth). Note the 45 degree installation of the eyes. Try to think how you can break free of simple grids and alignment:

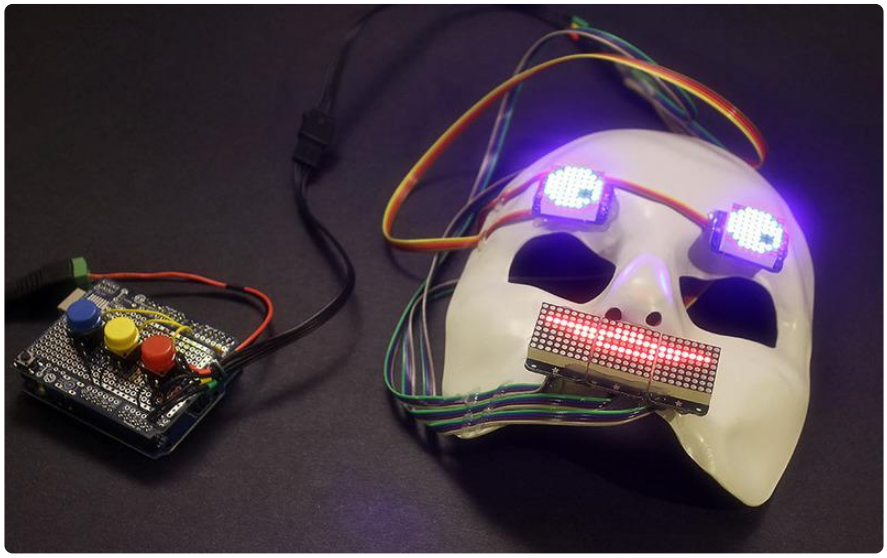
This glass head (which originally inspired the project) came from Pier 1 Imports. The addition of a bundled up length of Flowing Effect EL Wire (http://adafru.it/953) creates an impression of coursing blood or a pulsing brain:



The matrices were held in place with hot-melt glue, which can be cleanly removed later using a Q-Tip dipped in rubbing alcohol.

Lesson learned: at 200 milliamps each, these displays get warm enough to soften the glue. Nothing fell off, but it's something to keep in mind. If the situation permits, use the matrix backpacks' mounting holes (obviously this won't work with the glass head).

Another thought was to glue the displays to a mask, which could be worn under a "morph suit" — the LEDs are extremely bright and will show through the thin fabric:



It’s an interesting concept but still needs some work. Placing the LED backpacks directly over one’s mouth presents a problem: moisture from exhaled breath is conductive enough to confuse the address pads on the backpacks, and they quickly end up all showing the same image! A small blob of hot-melt glue covering the “open” address pads took care of this, but there may still be other issues lurking...the whole thing should probably be potted or enclosed, sealed off from breath and perspiration.

Level up: have the mouth move in sync with the wearer’s own...perhaps a pressure or flex sensor under the chin, or using a microphone based on volume.