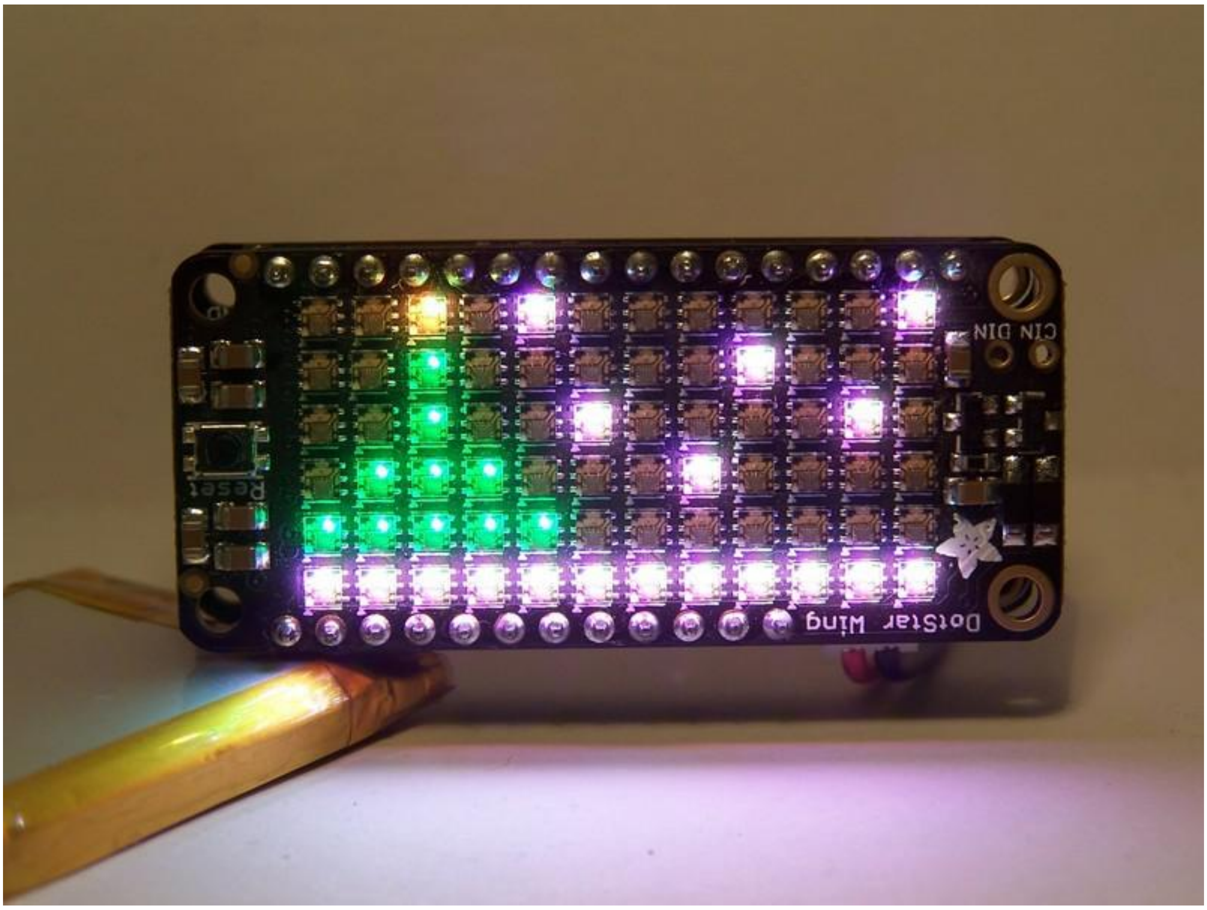
Dotstar Featherwing in CircuitPython

Created by Dave Astels

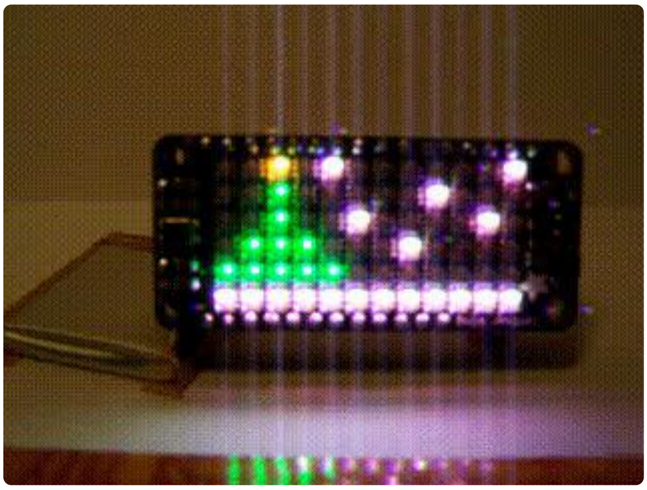


https://learn.adafruit.com/dotstar-featherwing-in-circuitpython

Last updated on 2023-08-29 03:37:33 PM EDT

Table of Contents

Overview



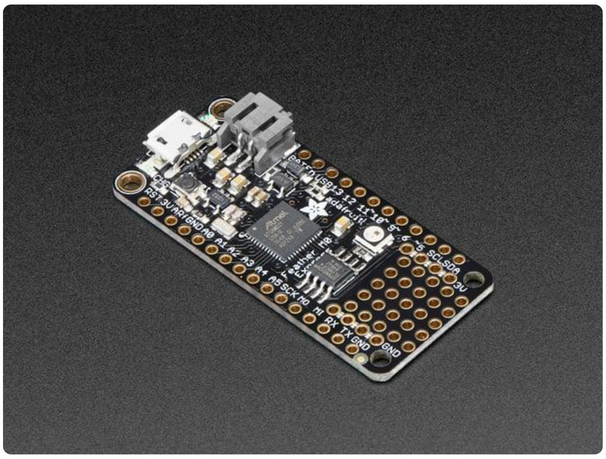
The Feather platform has really gotten my attention recently, especially since its move to using the ATSAMD21 MCU. This is a powerhouse chip based on the ARM Cortex-M0+ core. The other great thing about Feathers is the large (and constantly growing) ecosystem of add-on boards, called FeatherWings ().

Another thing I've enjoyed playing with are NeoPixels (). I've used them in various formats for a variety of projects. One of the joys is that you don't need 3 PWM outputs per LED; you just need a single digital output to drive a fairly large string of NeoPixels. Recently, a new RGB LED has been showing up in Adafruit products: the DotStar (). It requires 2 outputs, but is less timing sensitive, and can be updated faster. And it is a lot smaller. The latter point is huge: it means you can pack them denser, getting more pixels in the same space. Adafruit has done just that with their DotStar Featherwing. It packs a 6x12 DotStars in the same space that fits just 4x8 NeoPixels.

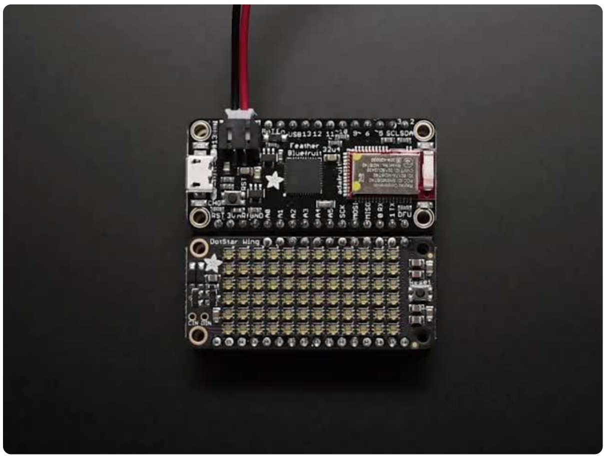
In case you weren't paying attention, a FeatherWing is only 50mm by 23mm. That's small. Lady Ada has packed the space with DotStars so densely they almost touch. That's enough pixels to make simple images, and display text using a simple font.

If you want to work in C/Arduino there's a guide () for that. In this guide I'll be walking you through a library I wrote for using it from CircuitPython.

Download the library



Adafruit Feather M0 Express
At the Feather M0's heart is an ATSAMD21G18 ARM Cortex M0+ processor, clocked at 48 MHz and at 3.3V logic, the same one used in the new
https://www.adafruit.com/product/3403



Adafruit DotStar FeatherWing - 6 x 12 RGB LEDs
A Feather board without ambition is a Feather board without FeatherWings! This is the DotStar FeatherWing, a 6x12 RGB LED Add-on For All Feather Boards! Using...
https://www.adafruit.com/product/3449

The Basics

The grid of DotStars on the wing is electrically one strand of 72 DotStars.

I’ve written a class that wraps the linear string of pixels into a 6×12 grid, which you create like this:

```
wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11)
```

The first argument is the clock pin and the second is the data pin. See the guide () for more information on pin selection. An optional third argument is the brightness: from 0.0 to 1.0.

Once you have an instance created, you can start manipulating the pixels. There are the simple things like `clear()` and `fill(color)`, and `show()`.

`clear()` turns off all pixels.

`fill(color)` sets all pixels to the given color.

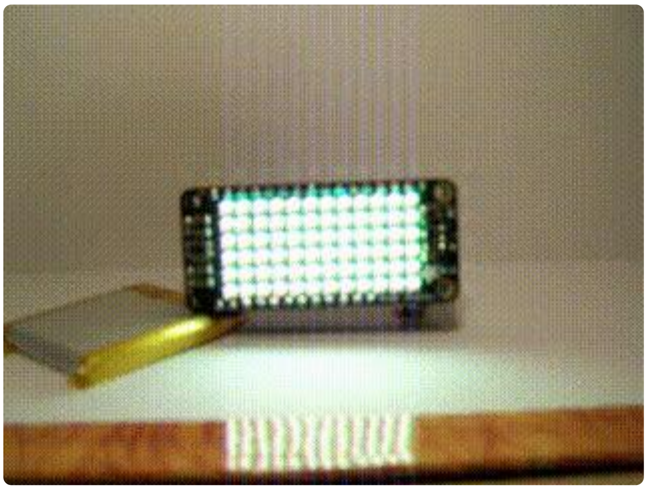
`show()` updates the physical dotstars to reflect the pixel colors. You will generally have to explicitly call `show()` to update the dotstars except for image and text display (which call `show()` internally).

At the beginning of your code, it's a good idea to `clear()` and `show()` to turn off all the dotstars. This way you know you are starting off with a blank display.

Here's an example of using `clear()`, `fill()`, and `show()`:

```
import board
import dotstar_featherwing
import time
import random

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11, 0.25)

wing.clear()
wing.show()

while True:
    color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    wing.fill(color)
    wing.show()
    time.sleep(0.25)
    wing.clear()
    wing.show()
    time.sleep(0.25)
```

This just flashes the entire display in random colors:



The next function is `set_color()` which takes a row, column, and color and sets the pixel at the row and column to the color. This example chooses a random pixel (by row and column) and sets it to a random color. It then sets another random pixel to black (i.e. `(0, 0, 0)`) thereby turning it off.

```
import board
import dotstar_featherwing
import time
import random
```

```
wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11, brightness=0.10)

wing.clear()
wing.show()

while True:
    row = random.randint(0, 5)
    column = random.randint(0, 11)
    color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    wing.set_color(row, column, color)
    row = random.randint(0, 5)
    column = random.randint(0, 11)
    wing.set_color(row, column, (0, 0, 0))
    wing.show()
```

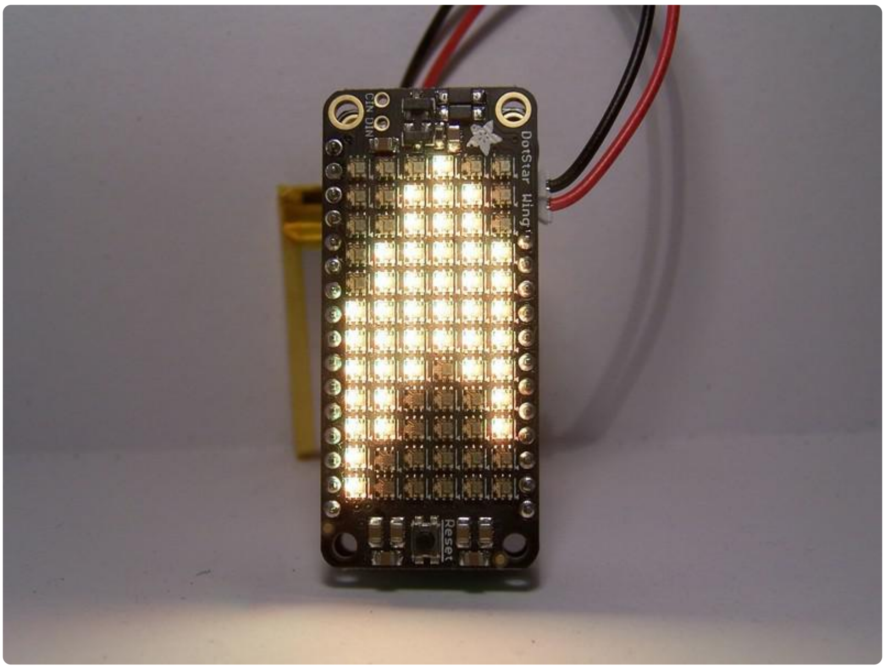
Images

Monochromatic Images

Displaying a single color (and “black”) image is straight-forward. It uses a list of strings to encode which pixels are colored and which are off. Uppercase X is used to indicate a lit pixel, any other character indicates an unlit pixel. I’ve found a period to work well visually.

The `display_image(image, color)` function has two parameters: the bitmap and the color to use for "on" pixels. Each line of the bitmap corresponds to a 12 pixel row of the display. Since the display is 6 pixels tall, the bitmap will have 6 such rows.

```
import board
import dotstar_featherwing

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11)

starfleet = ["XXXXXXX.....",
             "..XXXXXXX...",
             "....XXXXXXX.",
             ".....XXXXXXX",
             "....XXXXXXX.",
             "..XXXXXXX..."]

wing.display_image(starfleet, (32, 32, 32))
```



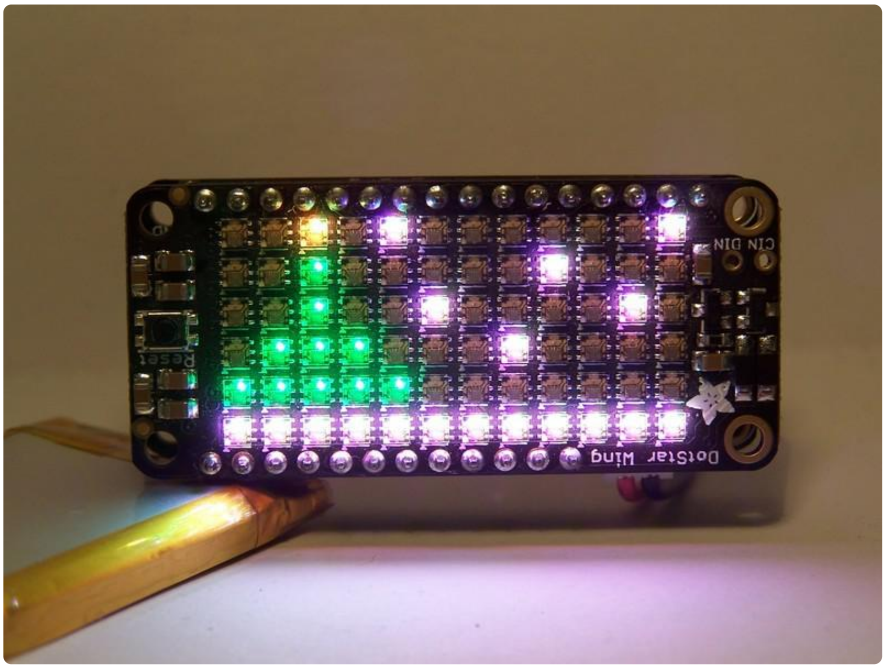
Multi-coloured images

Using a dictionary to map characters in the bitmap strings to colors, a color image can be displayed.

The `display_colored_image(image, colors)` functions takes an image as before, but its second parameter is a color mapping dictionary rather than a single color.

```
import board
import dotstar_featherwing

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11)

xmas = ["..y.w......w",
        "..G.....w...",
        "..G..w....w.",
        ".GGG...w....",
        "GGGGG.......",
        "wwwwwwwwwwww"]

xmas_colors = {'w': ( 32,  32,  32),
               'G': (  0,  32,   0),
               'y': ( 32,  32,   0)}

wing.display_colored_image(xmas, xmas_colors)
```

A snowy winter scene.



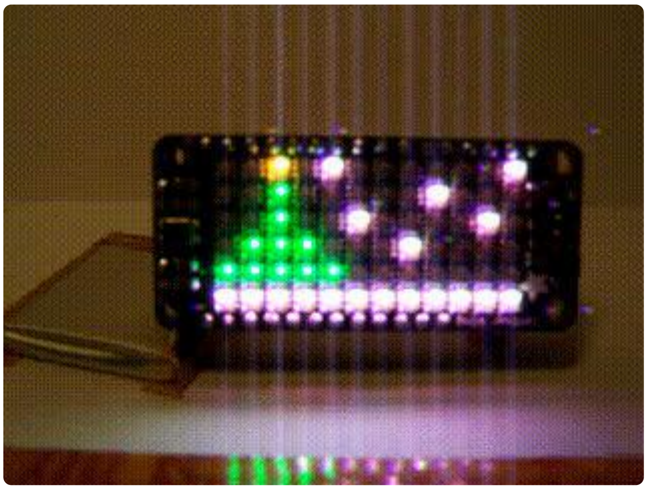
Animation

To go from displaying a static colored image to an animated one is just a matter of displaying a series of images in sequence.

The `display_animation(animation, colors, count, delay)` function does just that. The `animation` parameter is a list of images (each a list of strings as before). These are the frames in the animation. `colors` is the color mapping, again as before. `count` is the number of times to run the animation, which defaults to 1. Finally, `delay` is the time in seconds to wait between frames (including the final frame of a sequence and the first frame of the next repetition), and defaults to 0.1 seconds.

```
import board
import dotstar_featherwing

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11)

xmas_colors = {'w': ( 32,  32,  32),
               'W': (255, 255, 255),
               'G': (  0,  32,   0),
               'y': ( 32,  32,   0),
               'Y': (255, 255,   0)}

xmas_animation = [["..y.w......w",
                   "..G.....w...",
                   "..G..w....w.",
                   ".GGG...w....",
                   "GGGGG.......",
                   "wwwwwwwwwwww"],
                  ["..y.........",
                   "..G.W......w",
                   "..G.....w...",
                   ".GGG.w....W.",
                   "GGGGG..w....",
                   "wwwwwwwwwwww"],
                  ["..Y....W....",
                   "..G.........",
                   "..G.w......w",
                   ".GGG....w...",
                   "GGGGGw....W.",
                   "wwwwwwwwwwww"],
                  ["..y..w....w.",
                   "..G....W....",
                   "..G.........",
                   ".GGGW......w",
                   "GGGGG...w...",
                   "wwwwwwwwwwww"],
                  ["..Y.....w...",
                   "..G..w....W.",
                   "..G....w....",
                   ".GGG........",
                   "GGGGG......W",
                   "wwwwwwwwwwww"]]

wing.display_animation(xmas_animation, xmas_colors, 10, 0.05)
```

By using brighter versions of a couple colors, we can make the snow glitter and the star on top of the tree flash.



Scrolling

Images that fill the display are fine, but the array is pretty small and we might want to display something bigger. A scrolling display is often the way to do it. To make a scrolling display there are the `shift_into_left(stripe)` and `shift_into_right(stripe)` methods.

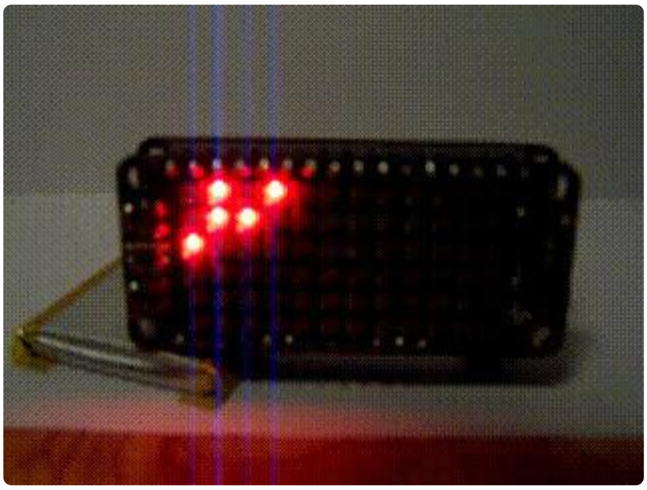
The `stripe` parameter is a list of colors, one per pixel, that will get shifted onto the display with the first one at the top.

The examples below show counting from 0 to 63 (i.e. 6 bits worth) and shifting each number onto the display from the left and right, respectively.

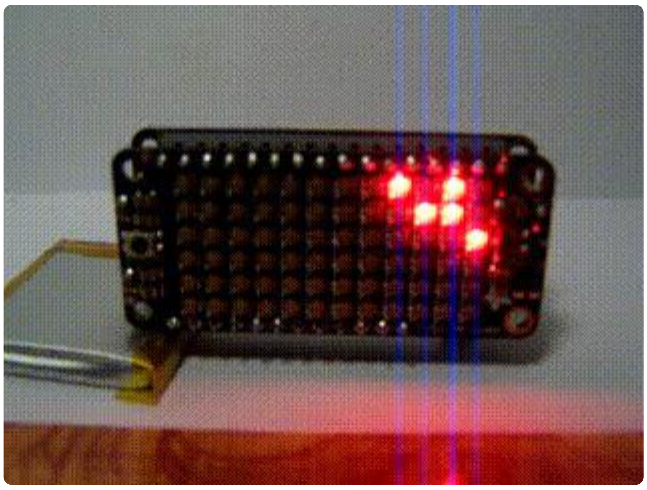
Both examples use the `numbers_to_pixels(x, color)` function that converts the number `x` to a list of color values: the `color` parameter when a pixel should be on, and `(0, 0, 0)` when it should be off. See the section on stripes () for more detail.

For example, `numbers_to_pixels(5, (32, 32, 8)` would result in `((0, 0, 0), (0, 0, 0), (0, 0, 0), (32, 32, 8), (0, 0, 0), (32, 32, 8))`

```
import board
import dotstar_featherwing
import time

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11)

# count from 0-&gt;63, shifting the binary pattern in from the left
while True:
    wing.clear()
    for x in range(64):
        wing.shift_into_left(wing.number_to_pixels(x, (64, 0, 0)))
        wing.show()
        time.sleep(0.2)
```



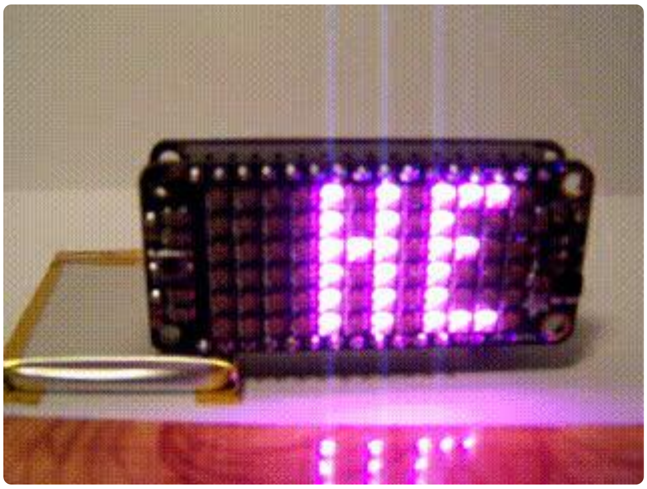
```
import board
import dotstar_featherwing
import time

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11)

# count from 0-&gt;63, shifting the binary pattern in from the right
while True:
    wing.clear()
    for x in range(64):
        wing.shift_into_right(wing.number_to_pixels(x, (64, 0, 0)))
        wing.show()
        time.sleep(0.2)
```



Text

Now that we can shift in arbitrary bit patterns, we can get more elaborate. This example will use the `shift_in_string(font, s, color, delay)` function to scroll text onto the pixel array. `font` is the font to use for character images, `s` is the string to be displayed, `color` is the color to use, and `delay` is the time to wait in seconds between columns.

```
import board
import dotstar_featherwing
import time
import font3

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11, 0.1)

while True:
    wing.clear()
    wing.shift_in_string(font3.font, "hello adafruit discord!", (32, 32, 32), 0.05)
    time.sleep(2)
```



Even a font of character 3 pixels wide takes up some space in memory. That last demo pretty much fills the Feather-M0s RAM. If you have a specific message or messages to scroll, consider removing any character from the font that you don’t need.

Another neat capability of the library is that characters (glyphs really) don’t have to all be the same width. This lets you have some custom images stored in a font to be displayed as and when you wish.

Fonts

A font is simply a dictionary that maps characters to their bitmap. This bitmap is a list of integers that define each vertical stripe of the character image. The first stripe is the left-most (and will get shifted onto the display first, from the right) and the lowest bit of each stripe is at the top.

Here's `font3`:

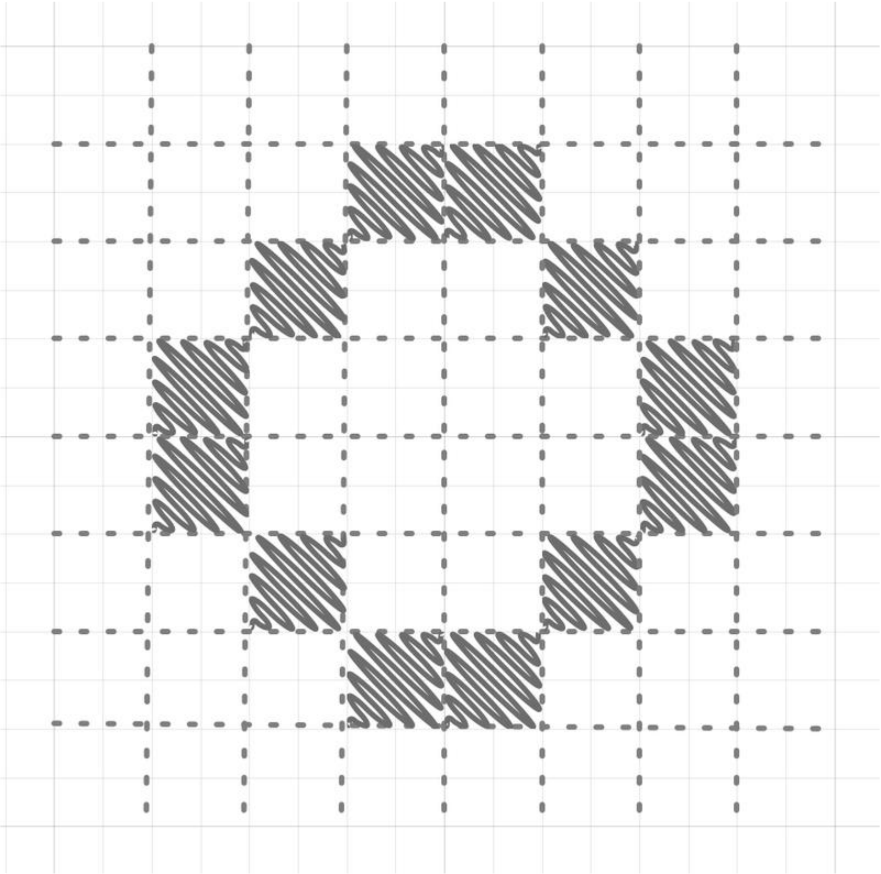
```
font = {' ': [ 0,  0,  0],
        'A': [62,  5, 62],
        'B': [63, 37, 26],
        'C': [30, 33, 18],
        'D': [63, 33, 30],
        'E': [63, 37, 33],
        'F': [63,  5,  1],
        'G': [30, 41, 26],
        'H': [63,  4, 63],
        'I': [33, 63, 33],
        'J': [33, 31,  1],
        'K': [63,  4, 59],
        'L': [63, 32, 32],
        'M': [63,  2, 63],
        'N': [63, 12, 63],
        'O': [30, 33, 30],
        'P': [63,  5,  2],
        'Q': [30, 33, 62],
        'R': [63,  5, 58],
        'S': [18, 37, 26],
        'T': [ 1, 63,  1],
        'U': [31, 32, 63],
        'V': [31, 32, 31],
        'W': [63, 16, 63],
        'X': [59,  4, 59],
        'Y': [ 3, 60,  3],
        'Z': [49, 45, 35],
        '0': [30, 33, 30],
        '1': [34, 63, 32],
        '2': [50, 41, 38],
        '3': [33, 37, 26],
        '4': [ 7,  4, 63],
        '5': [23, 37, 25],
        '6': [30, 41, 25],
        '7': [49,  9,  7],
        '8': [26, 37, 26],
        '9': [38, 41, 30],
        '!': [ 0, 47,  0],
        '?': [ 2, 41,  6],
        '.': [ 0, 32,  0],
        '-': [ 8,  8,  8],
        '_': [32, 32, 32],
        '+': [ 8, 28,  8],
        '/': [48,  12, 3],
        '*': [20,  8, 20],
        '=': [20, 20, 20],
        'UNKNOWN': [63, 33, 63] }
```

The `UNKNOWN` entry is used internally when a character not specified in the dictionary is encountered.

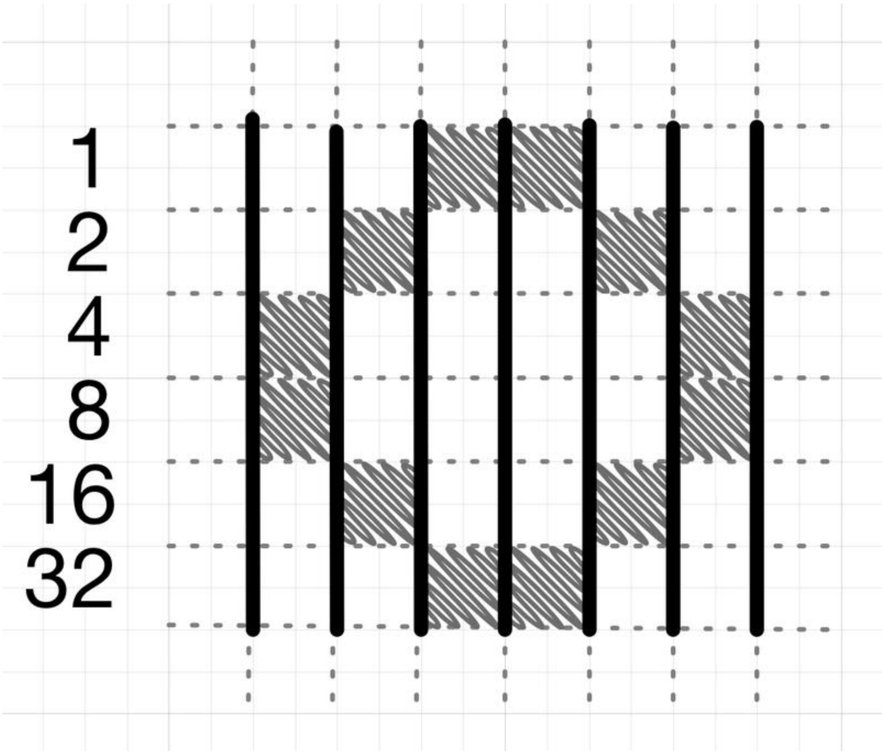
More on Stripes

How do we make our own shapes to shift onto the display? How do we make a custom font? How do we come up with those numbers for the slices?

Start by sketching out the shape you want: which pixels should be on and which should be off. Using graph paper helps with this. Remember that it can only be 6 pixels tall. For example, here's a 6 by 6 circle:



Once you have the shape defined, the next step is to look at each column of pixels, and convert them to numbers.



Using the numbers beside each row (which are the values corresponding to each bit), add up each column using 0 if the pixel is off and the corresponding row/bit number if it is on. Like so:

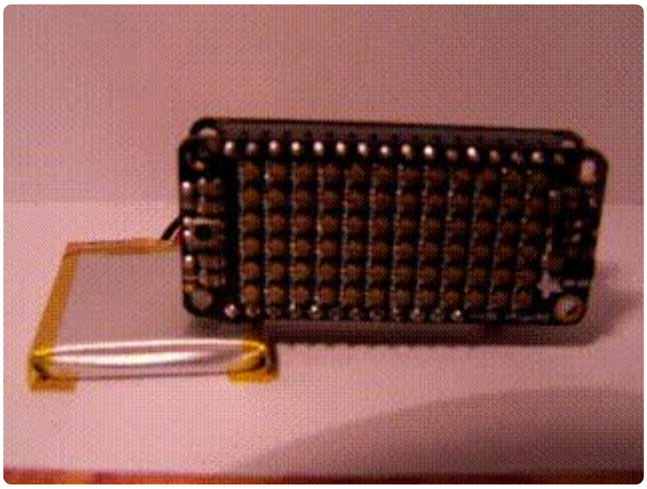
```
0 + 0 + 4 + 8 +  0 +  0 = 12
0 + 2 + 0 + 0 + 16 +  0 = 18
1 + 0 + 0 + 0 +  0 + 32 = 33
1 + 0 + 0 + 0 +  0 + 32 = 33
0 + 2 + 0 + 0 + 16 +  0 = 18
0 + 0 + 4 + 8 +  0 +  0 = 12
```

This takes advantage of the fact that a pixel, in this case, can be on or off. That is to say, each pixel corresponds to a single binary bit. By adding up the place values where the bits are 1/on we can arrive at a single number to represent the entire pattern.

If you are unfamiliar with binary, there's a great tutorial () here on Adafruit.

Now we use these along with the `shift_into_...` functions:

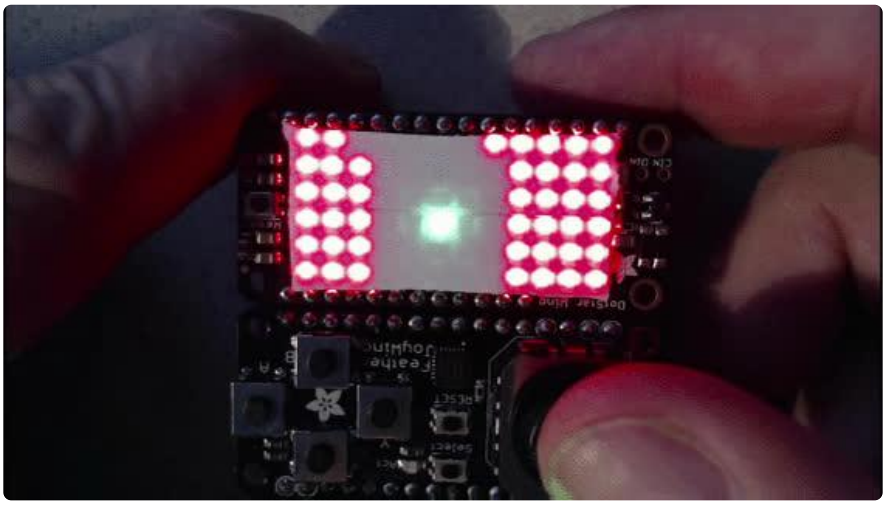
```
import board
import dotstar_featherwing
import time

wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11)

wing.clear()
for x in [12, 18, 33, 33, 18, 12]:
    wing.shift_into_left(wing.number_to_pixels(x, (0, 64, 0)))
    wing.show()
    time.sleep(0.2)
```



A Gauntlet Game

I was thinking: "What kind of game can I do on a 6x12 screen?" What I came up with was a gauntlet game. This is sort of like a running game and much like early racing games (early as in Atari 2600) in that the track continually advances and swerves, and the player has to stay on it. I added the extra feature of having randomly spawning targets that add to the players score when they are hit. Finally if the player hits the edge of the track the score and number of steps is output to the console and the game restarts.

I’ll be showing edited snippets of the code as I discuss various aspects. The full code is at the end of this page and in the examples directory of the library.



Setup

We need to import the libraries and create the global variables that will be used later.

```
import time
import random

import board
import busio
import dotstar_featherwing
import Adafruit_seesaw

i2c = busio.I2C(board.SCL, board.SDA)
ss = Adafruit_seesaw.Seesaw(i2c)
wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11, 0.1)

black = 0x000000
wall = 0x200800                 # must not have any blue, must have red
pellet = 0x000040               # must have blue
player = 0x00FF00
```

The reason the wall and pellet colors have the noted constraints is to simplify (and thus optimize for space) the tests later.

Making the track

The first thing that was needed was a way to shift a row onto the display from the top. I added a method to do that to the library. I added a slight change from the left/right shifting methods: an offset into the slice at which to start taking 12 pixels. By randomly changing the offset into a slice that is wider than the 12 column display I can make the track swerve back and forth.

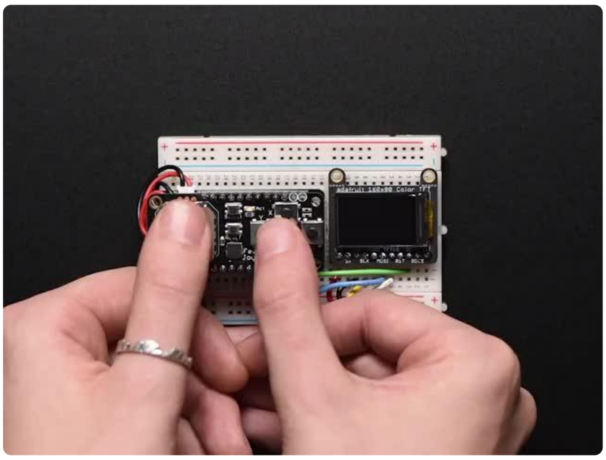
```
row = (wall, wall, wall, wall,
       wall, wall, wall, wall,
       black, black, black, black, black,
       wall, wall, wall, wall,
       wall, wall, wall, wall)

offset = 4
# ...
offset = min(max(0, offset + random.randint(-1, 1)), 9)
wing.shift_into_top(row, offset)
```

The main loop simply adjusts the offset like that and shifts into the top (edited to show just the track creation):

```
while True:
    offset = min(max(0, offset + random.randint(-1, 1)), 9)
    wing.shift_into_top(row, offset)
    wing.show()
    time.sleep(0.1)
```

The player

Next we need a player sprite. We need to be able to move the player sprite back & forth to stay on the track. That's where the Joy FeatherWing comes in. Functionally it's a Seesaw and we use the Seesaw library to interact with it. In this case interaction is simple reading the horizontal axis of the thumbstick.



Adafruit Joy FeatherWing for all Feathers
Make a game or robotic controller with this Joy-ful FeatherWing. This FeatherWing has a 2-axis joystick and 5 momentary buttons (4 large and 1 small)...
https://www.adafruit.com/product/3632

```
wing.set_color(3, player_x, black)

joy_x = ss.analog_read(3)
    if joy_x &lt; 256 and player_x &gt; 0:
        player_x -= 1
    elif joy_x &gt; 768 and player_x &lt; 11:
        player_x += 1

wing.set_color(3, player_x, player)
```

Score pellets

To add a goal I spawn a score pellet with a 5% chance each time through the loop. If the player maneuvers their sprite over a score pellet they get a point.

The score pellet only needs to be initially placed; the shifting/scrolling will take care of moving it.

```
if random.randint(1, 20) == 1:
    wing.set_color(0, random.randint(8, 12) - offset, pellet)
```

Collisions

There are two things that the player sprite can collide with: the wall and a score pellet.

Colliding with the wall causes the game to end, print the players score and number of steps they survived to the console, and start a fresh game.

Colliding with a pellet increments the score.

This is where the pellet having blue in it, and the wall having red and no blue comes into play to drastically simplify the check.

```
r, _, b = wing.get_color(3, player_x)
if b:
    score += 1
elif r:
    return score
```

I also have it increment the score every 25 iterations and increase the speed every 100.

```
steps += 1
if steps % 25 == 0:
    score += 1
if steps % 100 == 0:
    step_delay *= 0.9
sleep(step_delay)
```

Notice that in the case of a wall collision, the function returns the number of steps and score. The top level code in main.py calls the game function that I've been describing, prints the returned values, flashes the screen red, and loops.

```
while True:
    result = run()
    # got here because of a crash, so report score and restart
    wing.shift_in_string(numbers, '{:03d}'.format(result), 0x101010)
    sleep(5)
```

The entire code

```
# The MIT License (MIT)
#
# Copyright (c) 2018 Dave Astels
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.

"""
A gaunlet running game using the dotstar wing and the joy wing.
"""

from time import sleep
from random import randint

import board
import busio
import dotstar_featherwing
import Adafruit_seesaw

i2c = busio.I2C(board.SCL, board.SDA)
ss = Adafruit_seesaw.Seesaw(i2c)
wing = dotstar_featherwing.DotstarFeatherwing(board.D13, board.D11, 0.1)

black = 0x000000
wall = 0x200800           # must not have any blue, must have red
pellet = 0x000040               # must have blue
player = 0x00FF00

numbers = {
    ' ': [0,  0,  0],
    '0': [30, 33, 30],
    '1': [34, 63, 32],
    '2': [50, 41, 38],
    '3': [33, 37, 26],
    '4': [ 7,  4, 63],
    '5': [23, 37, 25],
    '6': [30, 41, 25],
    '7': [49,  9,  7],
    '8': [26, 37, 26],
    '9': [38, 41, 30],
}

row = (wall, wall, wall, wall,
       wall, wall, wall, wall,
       black, black, black, black, black,
       wall, wall, wall, wall,
       wall, wall, wall, wall)


def run():
    """Play the game."""

    player_x = 6
    score = 0
    steps = 0
    step_delay = 0.15
    offset = 4

    for _ in range(wing.rows):
        wing.shift_into_top(row, offset)
    wing.show()


    while True:
        # remove player sprite
        wing.set_color(3, player_x, black)

        # shift/advance the track
        offset = min(max(0, offset + randint(-1, 1)), 9)
        wing.shift_into_top(row, offset)
        # Maybe add a pellet
        if randint(1, 20) == 1:
            wing.set_color(0, randint(8, 12) - offset, pellet)

        # Adjust player position
        joy_x = ss.analog_read(3)
        if joy_x &lt; 256 and player_x &gt; 0:
            player_x -= 1
        elif joy_x &gt; 768 and player_x &lt; 11:
            player_x += 1

        # Check for collisions
        r, _, b = wing.get_color(3, player_x)
        if b:
            score += 1
        elif r:
            return score

        # Show player sprite
        wing.set_color(3, player_x, player)

        # Update some things and sleep a bit
        wing.show()
        steps += 1
        if steps % 25 == 0:
            score += 1
        if steps % 100 == 0:
            step_delay *= 0.9
        sleep(step_delay)

while True:
    result = run()
    # got here because of a crash, so report score and restart
    wing.shift_in_string(numbers, '{:03d}'.format(result), 0x101010)
    sleep(5)
```

Downloads

Getting the code

You can download the library as well as the above examples using the button below. Just copy `dotstar_featherwing.py` and `font3.py` from the `dotstar_featherwing` directory in the zip file to the `lib` folder of your `CIRCUITPY` drive and you're ready to go. The examples shown in this guide are in the `examples` directory.

Download the library