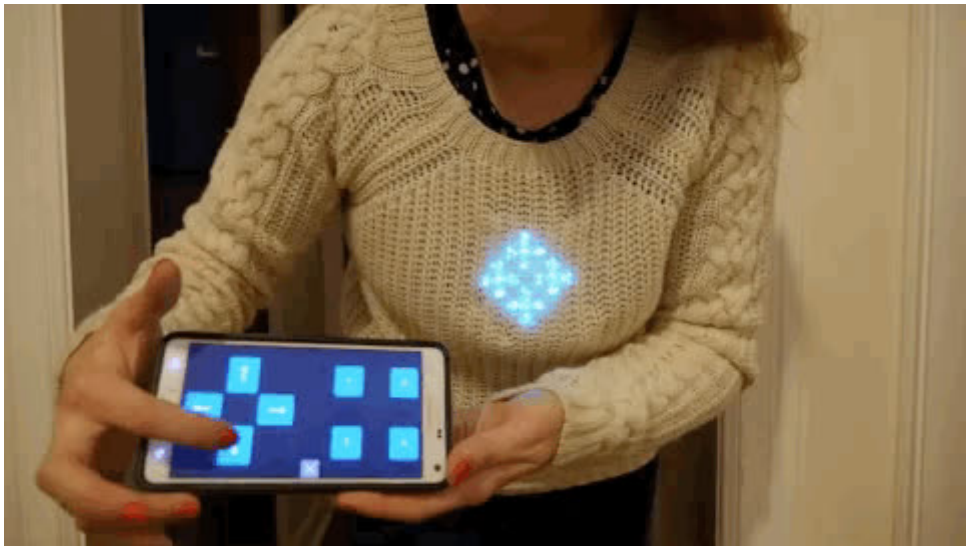
NeoPixel Matrix Snowflake Sweater

Created by Becky Stern



https://learn.adafruit.com/neopixel-matrix-snowflake-sweater

Last updated on 2024-06-03 01:50:28 PM EDT

Table of Contents

Overview

Create a tacky sweater controlled by your phone! It's easy to put together this Bluefruit and NeoPixel matrix circuit to display snowflakes in a sweater, and control the animation and color using the Adafruit Bluefruit LE Connect app for iOS or Android.

This is an easy project to build but probably not best for a "first project" as there are a lot of concepts being mixed together and the matrix can use quite a lot of power. You can build it with any of our Bluefruit products and the microcontroller of your choice, such as FLORA with its BLE module, or the all-in-one Feather 32u4 Bluefruit LE.

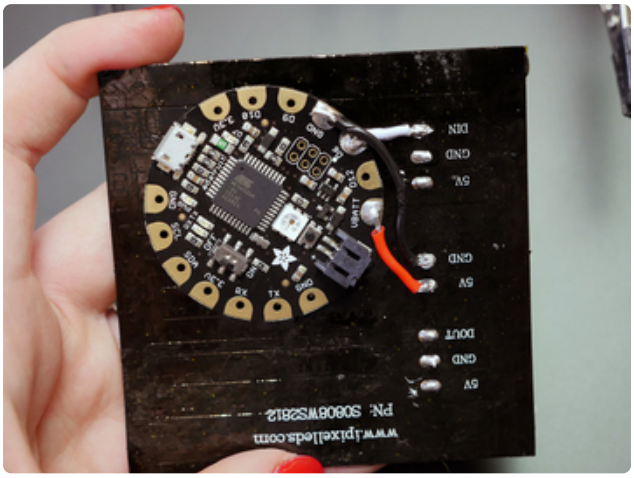
Before you begin, read and understand the following prerequisite guides:

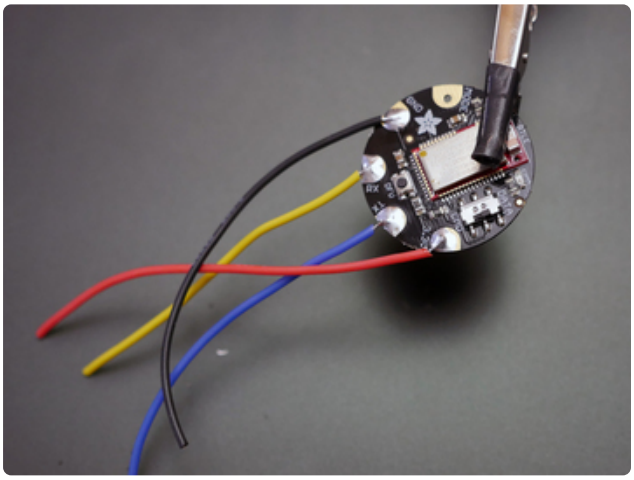
- NeoPixel Uberguide (https://adafru.it/dhw)
- Adafruit Feather 32u4 Bluefruit LE (https://adafru.it/kcc) or
- Getting Started with Adafruit FLORA (https://adafru.it/dwi) and FLORA BLE (https://adafru.it/ioe)
- Battery Powering Wearable Electronics (https://adafru.it/jNa)
- Adafruit Guide to Excellent Soldering (https://adafru.it/drl)

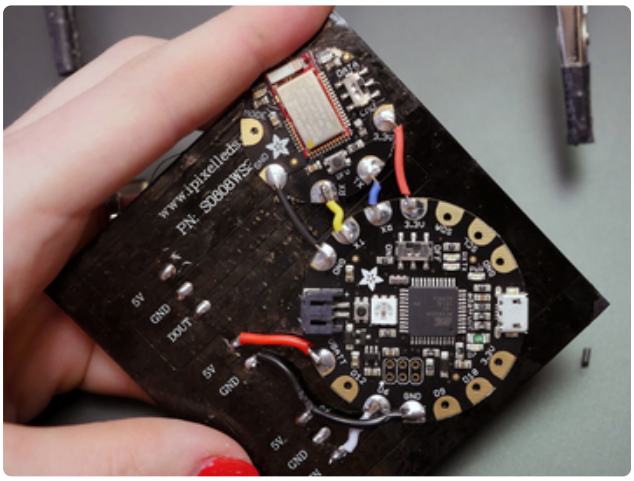
For this project you will need:

- Flexible 8x8 NeoPixel matrix (http://adafru.it/2612)
- Feather 32u4 Bluefruit LE (http://adafru.it/2829) or FLORA (http://adafru.it/659) +BLE (http://adafru.it/2487) (or other microcontroller + Bluefruit module of your choice)
- Lipoly battery at least 1200 mAh (http://adafru.it/258)
- JST extension cable (http://adafru.it/1131)
- Adafruit Bluefruit LE Connect app for iOS (https://adafru.it/19rA) or Android (https://adafru.it/19rB)

Assemble Circuit



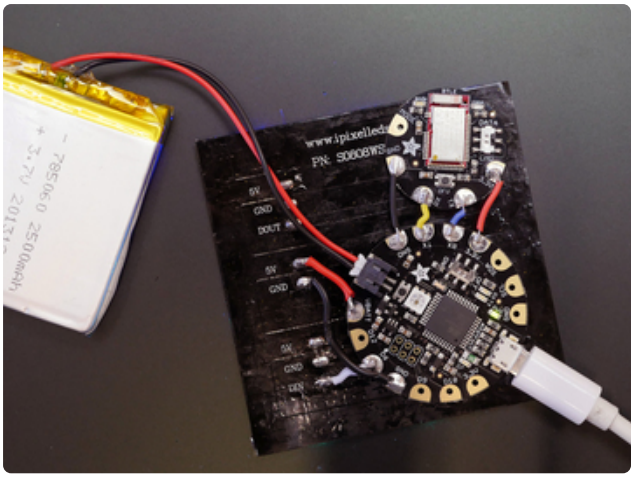


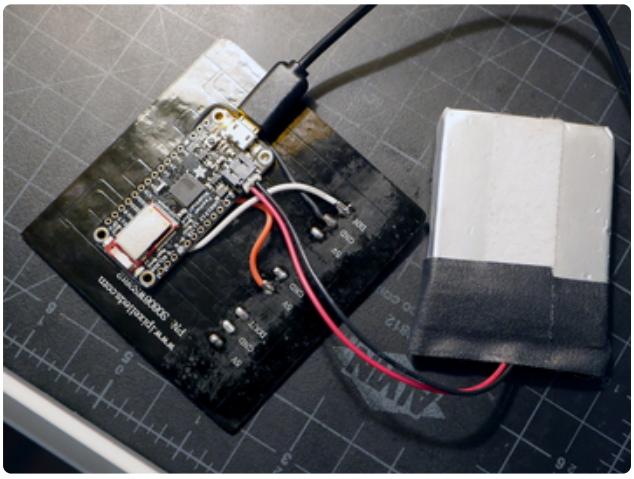


Use your soldering iron to heat up the connections on the back of your NeoPixel matrix, and remove all but one power, one ground, and the data in wire. Reposition the wires to face towards the center of the matrix instead of outwards.

If using FLORA and the FLORA BLE module, solder the collowing connections:

matrix DIN -> FLORA D6
matrix 5V -> FLORA VBATT
matrix GND -> FLORA GND
BLE 3.3V -> FLORA 3.3V
BLE TX -> FLORA RX
BLE RX -> FLORA TX
BLE GND -> FLORA GND
Plug your battery into the JST port and a USB cable to the micro USB port. The USB port on your computer may not be able to supply adequate current to the matrix, so be sure your FLORA's power switch is also set to ON to be sure the battery is supplying power.



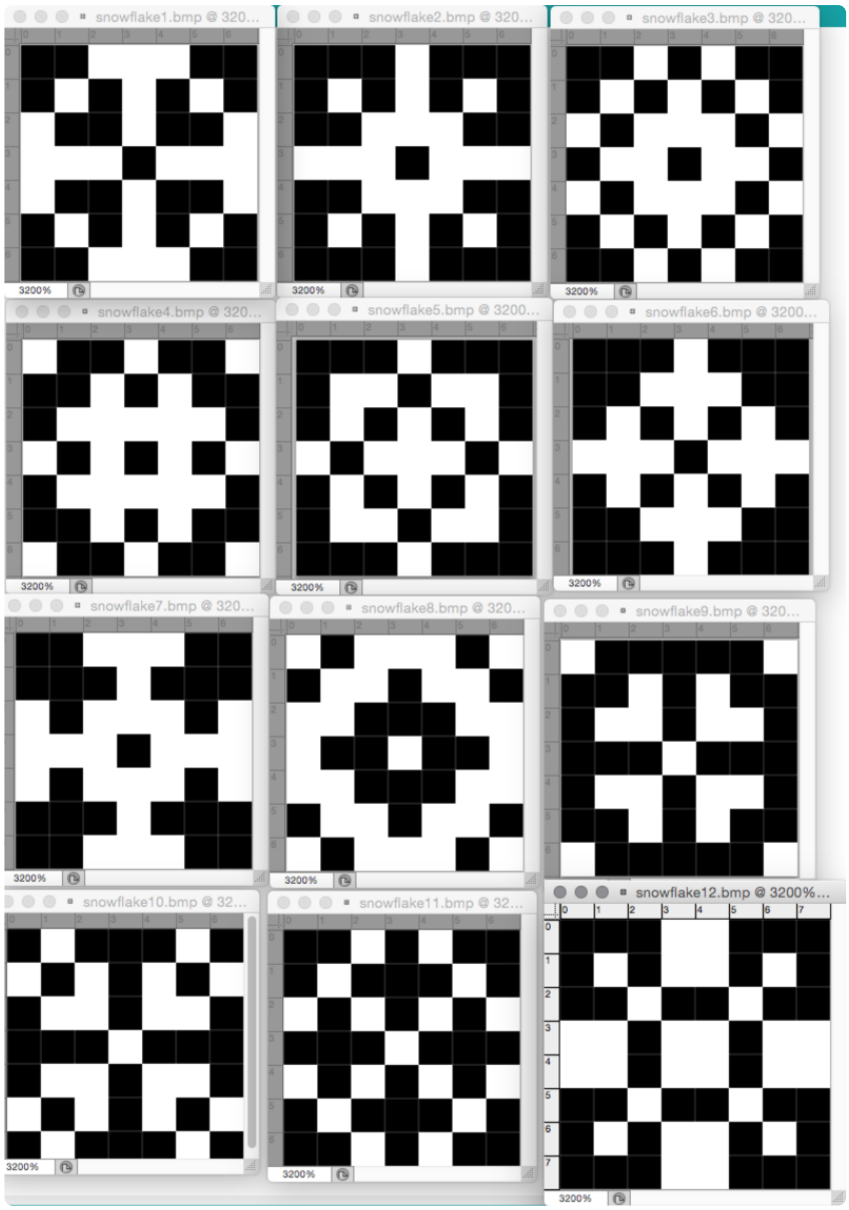


If you're using the Feather 32u4 Bluefruit LE, it's even easier with only three connections:

matrix DIN -> Feather 6
matrix 5V -> Feather BAT
matrix GND -> Feather GND

Code

These are the snowflakes we designed to go with this project-- we sketched them out as bitmaps in Photoshop and then coded little snowflake functions using the GFX library for Arduino.



Download the Code for Your Application

All the code resides in GitHub in this repository (https://adafru.it/EI3).

There are three sets of code:

1. A basic test program, no Bluetooth,
2. Using a Feather 32u4 Bluefruit and
3. Using FLORA and the FLORA BLE module.

See the following pages for each variant. Download the code that matches your board (Feather or Flora), then open up the .ino file inside that folder with Arduino.

Basic Test

To test out your matrix or build this project without bluetooth control, you can use this basic "snowflakes only" sketch:

```
// SPDX-FileCopyrightText: 2019 Anne Barela for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*************************************************************************
 Basic snowflake code (without Bluetooth)

  Accompanies the Adafruit guide
  https://learn.adafruit.com/neopixel-matrix-snowflake-sweater

 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!

 MIT license, check LICENSE for more information
 All text above, and the splash screen below must be included in
 any redistribution
*************************************************************************/
#include <Adafruit_NeoPixel.h>
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
/*=========================================================================
    APPLICATION SETTINGS

    MATRIX DECLARATION        Parameter 1 = width of NeoPixel matrix
                              Parameter 2 = height of matrix
                              Parameter 3 = pin number (most are valid)
                              Parameter 4 = matrix layout flags, add together as
needed:
    NEO_MATRIX_TOP,
    NEO_MATRIX_BOTTOM,
    NEO_MATRIX_LEFT,
    NEO_MATRIX_RIGHT          Position of the FIRST LED in the matrix; pick two,
e.g.
                              NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left
corner.
                              NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are
arranged in horizontal
                              rows or in vertical columns, respectively; pick one
or the other.
    NEO_MATRIX_PROGRESSIVE,
    NEO_MATRIX_ZIGZAG         all rows/columns proceed in the same order,
                              or alternate lines reverse direction; pick one.

                              See example below for these values in action.

    Parameter 5 = pixel type flags, add together as needed:

    NEO_KHZ800                800 KHz bitstream (most NeoPixel products w/WS2812
LEDs)
    NEO_KHZ400                400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811
drivers)
    NEO_GRB                   Pixels are wired for GRB bitstream (most NeoPixel
products)
    NEO_RGB                   Pixels are wired for RGB bitstream (v1 FLORA pixels,
not v2)
    -------------------------------------------------------------------------*/
    #define FACTORYRESET_ENABLE     1
```

```
    #define PIN                     6   // Which pin on the Arduino is connected to
the NeoPixels?

// Example for NeoPixel 8x8 Matrix.  In this application we'd like to use it
// with the back text positioned along the bottom edge.
// When held that way, the first pixel is at the top left, and
// lines are arranged in columns, zigzag order.  The 8x8 matrix uses
// 800 KHz (v2) pixels that expect GRB color data.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_ZIGZAG,
  NEO_GRB            + NEO_KHZ800);
/*=========================================================================*/

void setup(void)
{
  matrix.begin();
  matrix.setBrightness(40);

  matrix.fillScreen(0);
  matrix.show(); // This sends the updated pixel colors to the hardware.
}

void loop(void)
{
        matrix.fillScreen(0);
        SnowFlake1(matrix.Color(200, 200, 200));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake2(matrix.Color(255, 0, 0));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake3(matrix.Color(0, 255, 0));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake4(matrix.Color(200, 200, 200));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake5(matrix.Color(255, 0, 0));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake6(matrix.Color(0, 255, 0));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake7(matrix.Color(200, 200, 200));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        matrix.fillScreen(0);
        SnowFlake8(matrix.Color(255, 0, 0));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
}

void SnowFlake1(uint32_t c){
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawPixel(3, 3, 0); // x, y, color
}

void SnowFlake2(uint32_t c){
```

```
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawLine(1, 1, 5, 5, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, c); // x0, y0, x1, y1, color
  matrix.drawPixel(3, 3, 0); // x, y, color
}

void SnowFlake3(uint32_t c){
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 4, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, c); // x0, y0, x1, y1, color
  matrix.drawPixel(3, 3, 0); // x, y, color
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawLine(2, 6, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
}

void SnowFlake4(uint32_t c){
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 1, 5, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 4, 2, 5, c); // x0, y0, x1, y1, color
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color

}

void SnowFlake5(uint32_t c){
  matrix.fillRect(1, 1, 5, 5, c); // x0, y0, width, height
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 3, 3, 1, 0); // x0, y0, x1, y1, color
  matrix.drawLine(3, 5, 5, 3, 0); // x0, y0, x1, y1, color
  matrix.drawPixel(2, 4, 0); // x, y, color
  matrix.drawPixel(4, 2, 0); // x, y, color

}

void SnowFlake6(uint32_t c){
  matrix.fillRect(1, 1, 5, 5, c); // x0, y0, width, height
  matrix.drawLine(1, 1, 5, 5, 0); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, 0); // x0, y0, x1, y1, color
  matrix.drawPixel(0, 3, c); // x, y, color
  matrix.drawPixel(3, 0, c); // x, y, color
  matrix.drawPixel(3, 6, c); // x, y, color
  matrix.drawPixel(6, 3, c); // x, y, color
}

void SnowFlake7(uint32_t c){
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawPixel(3, 3, 0); // x, y, color
  matrix.drawFastHLine(2, 0, 3, c); // x0, y0, length, color
  matrix.drawFastHLine(2, 6, 3, c); // x0, y0, length, color
  matrix.drawFastVLine(0, 2, 3, c); // x0, y0, length, color
  matrix.drawFastVLine(6, 2, 3, c); // x0, y0, length, color
}

void SnowFlake8(uint32_t c){
  //four corners
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
```

```
  matrix.drawPixel(6, 6, c); // x, y, color

  //upper left corner
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //upper right corner
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 0, 6, 3, c); // x0, y0, x1, y1, color

  //lower left corner
  matrix.drawLine(0, 3, 3, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color

  //lower right corner
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
}
void SnowFlake9(uint32_t c){
  //four corners
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //four boxes near center
  matrix.drawRect(1, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(1, 4, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 4, 2, 2, c); // x0, y0, width, height

  //clear out corner pixel of boxes
  matrix.drawPixel(1, 1, 0); // x, y, color
  matrix.drawPixel(5, 1, 0); // x, y, color
  matrix.drawPixel(1, 5, 0); // x, y, color
  matrix.drawPixel(5, 5, 0); // x, y, color
}

void SnowFlake10(uint32_t c){
  //lines across the corners
  matrix.drawLine(0, 1, 1, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(5, 0, 6, 1, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 5, 1, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(5, 6, 6, 5, c); // x0, y0, x1, y1, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //four boxes near center
  matrix.drawRect(1, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(1, 4, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 4, 2, 2, c); // x0, y0, width, height

  //clear out corner pixel of boxes
  matrix.drawPixel(1, 1, 0); // x, y, color
  matrix.drawPixel(5, 1, 0); // x, y, color
  matrix.drawPixel(1, 5, 0); // x, y, color
  matrix.drawPixel(5, 5, 0); // x, y, color
}

void SnowFlake11(uint32_t c){
  //corner lines
```

```
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color

  //center X
  matrix.drawLine(2, 2, 4, 4, c); // x0, y0, x1, y1, color
  matrix.drawLine(2, 4, 4, 2, c); // x0, y0, x1, y1, color

}

//8x8
void SnowFlake12(uint32_t c){
  matrix.drawLine(1, 1, 6, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 6, 6, 1, c); // x0, y0, x1, y1, color
  matrix.fillRect(3, 0, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(0, 3, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(6, 3, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(3, 6, 2, 2, c); // x0, y0, width, height
  matrix.show();

}

/*
void showAllSnowflakes(uint32_t c){
        SnowFlake1(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake5(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake2(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake8(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake3(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake9(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake4(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake10(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake6(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake11(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake7(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
```

```
        SnowFlake11(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
}
*/
```

Feather BlueFruit LE

For Feather (https://adafru.it/EI4) (hardware SPI) - click Download: Project Zip in the .ino file listing below.

```
// SPDX-FileCopyrightText: 2019 Anne Barela for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*********************************************************************
 This is an example for our nRF51822 based Bluefruit LE modules
 Pick one up today in the adafruit shop! adafruit.com
 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!
 MIT license, check LICENSE for more information
 All text above, and the splash screen below must be included in
 any redistribution
*********************************************************************/

#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#if not defined (_VARIANT_ARDUINO_DUE_X_) && not defined (_VARIANT_ARDUINO_ZERO_)
  #include <SoftwareSerial.h>
#endif

#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"

#include "BluefruitConfig.h"

#include <Adafruit_NeoPixel.h>
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>

/*=========================================================================
    APPLICATION SETTINGS
    FACTORYRESET_ENABLE       Perform a factory reset when running this sketch

                              Enabling this will put your Bluefruit LE module
                              in a 'known good' state and clear any config
                              data set in previous sketches or projects, so
                              running this at least once is a good idea.

                              When deploying your project, however, you will
                              want to disable factory reset by setting this
                              value to 0.  If you are making changes to your
                              Bluefruit LE device via AT commands, and those
                              changes aren't persisting across resets, this
                              is the reason why.  Factory reset will erase
                              the non-volatile memory where config data is
                              stored, setting it back to factory default
                              values.

                              Some sketches that require you to bond to a
                              central device (HID mouse, keyboard, etc.)
```

```
                            won't work at all with this feature enabled
                            since the factory reset will clear all of the
                            bonding data stored on the chip, meaning the
                            central device won't be able to reconnect.

    MATRIX DECLARATION      Parameter 1 = width of NeoPixel matrix
                            Parameter 2 = height of matrix
                            Parameter 3 = pin number (most are valid)
                            Parameter 4 = matrix layout flags, add together as
needed:
    NEO_MATRIX_TOP,
    NEO_MATRIX_BOTTOM,
    NEO_MATRIX_LEFT,
    NEO_MATRIX_RIGHT        Position of the FIRST LED in the matrix; pick two,
e.g.
                            NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left
corner.
                            NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are
arranged in horizontal
                            rows or in vertical columns, respectively; pick one
or the other.
    NEO_MATRIX_PROGRESSIVE,
    NEO_MATRIX_ZIGZAG       all rows/columns proceed in the same order,
                            or alternate lines reverse direction; pick one.
                            See example below for these values in action.

    Parameter 5 = pixel type flags, add together as needed:

    NEO_KHZ800              800 KHz bitstream (most NeoPixel products w/WS2812
LEDs)
    NEO_KHZ400              400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811
drivers)
    NEO_GRB                 Pixels are wired for GRB bitstream (most NeoPixel
products)
    NEO_RGB                 Pixels are wired for RGB bitstream (v1 FLORA pixels,
not v2)
    ---------------------------------------------------------------------*/
    #define FACTORYRESET_ENABLE     1

    #define PIN                     6   // Which pin on the Arduino is connected to
the NeoPixels?

// Example for NeoPixel 8x8 Matrix.  In this application we'd like to use it
// with the back text positioned along the bottom edge.
// When held that way, the first pixel is at the top left, and
// lines are arranged in columns, zigzag order.  The 8x8 matrix uses
// 800 KHz (v2) pixels that expect GRB color data.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_ZIGZAG,
  NEO_GRB            + NEO_KHZ800);
/*=========================================================================*/



// Create the bluefruit object, either software serial...uncomment these lines
/*
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN,
BLUEFRUIT_SWUART_RXD_PIN);
Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
*/

/* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line
*/
// Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);

/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/
IRQ/RST */
```

```
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ,
BLUEFRUIT_SPI_RST);

/* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user
selected CS/IRQ/RST */
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
//                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
//                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);


// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  while (1);
}

// function prototypes over in packetparser.cpp
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout);
float parsefloat(uint8_t *buffer);
void printHex(const uint8_t * data, const uint32_t numBytes);

// the packet buffer
extern uint8_t packetbuffer[];


/**************************************************************************/
/*!
    @brief  Sets up the HW an the BLE module (this function is called
            automatically on startup)
*/
/**************************************************************************/
//additional variables

//Color
    uint8_t red = 100;
    uint8_t green = 100;
    uint8_t blue = 100;

    uint8_t animationState = 1;

void setup(void)
{
  //while (!Serial);  // required for Flora & Micro
  delay(500);

  matrix.begin();
  matrix.setBrightness(40);

  matrix.fillScreen(0);
  showAllSnowflakes(matrix.Color(red, green, blue));
  matrix.show(); // This sends the updated pixel colors to the hardware.

  Serial.begin(115200);
  Serial.println(F("Adafruit Bluefruit NeoMatrix Snowflake"));
  Serial.println(F("---------------------------------------------"));

  /* Initialise the module */
  Serial.print(F("Initialising the Bluefruit LE module: "));

  if ( !ble.begin(VERBOSE_MODE) )
  {
    error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check
wiring?"));
  }
  Serial.println( F("OK!") );

  if ( FACTORYRESET_ENABLE )
  {
    /* Perform a factory reset to make sure everything is in a known state */
```

```
    Serial.println(F("Performing a factory reset: "));
    if ( ! ble.factoryReset() ){
      error(F("Couldn't factory reset"));
    }
  }

  /* Disable command echo from Bluefruit */
  ble.echo(false);

  Serial.println("Requesting Bluefruit info:");
  /* Print Bluefruit information */
  ble.info();

  Serial.println(F("Please use Adafruit Bluefruit LE app to connect in Controller
mode"));
  Serial.println(F("Then activate/use the color picker and controller!"));
  Serial.println();

  ble.verbose(false);  // debug info is a little annoying after this point!

  /* Wait for connection */
  while (! ble.isConnected()) {
      delay(500);
  }

  Serial.println(F("***********************"));

  // Set Bluefruit to DATA mode
  Serial.println( F("Switching to DATA mode!") );
  ble.setMode(BLUEFRUIT_MODE_DATA);

  Serial.println(F("***********************"));

}

/**************************************************************************/
/*!
    @brief  Constantly poll for new command or response data
*/
/**************************************************************************/
void loop(void)
{

  /* Wait for new data to arrive */
  uint8_t len = readPacket(&ble, BLE_READPACKET_TIMEOUT);
  //if (len == 0) return;

  /* Got a packet! */
  // printHex(packetbuffer, len);

  // Color
  if (packetbuffer[1] == 'C') {
    red = packetbuffer[2];
    green = packetbuffer[3];
    blue = packetbuffer[4];
    Serial.print ("RGB #");
    if (red < 0x10) Serial.print("0");
    Serial.print(red, HEX);
    if (green < 0x10) Serial.print("0");
    Serial.print(green, HEX);
    if (blue < 0x10) Serial.print("0");
    Serial.println(blue, HEX);
  }

  // Buttons
  if (packetbuffer[1] == 'B') {

    uint8_t buttnum = packetbuffer[2] - '0';
    boolean pressed = packetbuffer[3] - '0';
```

```
    Serial.print ("Button "); Serial.print(buttnum);
    animationState = buttnum;
    if (pressed) {
      Serial.println(" pressed");
    } else {
      Serial.println(" released");
    }
  }

      if (animationState == 1){ // animate through all the snowflakes
        matrix.fillScreen(0);
        showAllSnowflakes(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 2){
        matrix.fillScreen(0);
        SnowFlake2(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 3){
        matrix.fillScreen(0);
        SnowFlake3(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 4){
        matrix.fillScreen(0);
        SnowFlake4(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 5){
        matrix.fillScreen(0);
        SnowFlake5(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }
      if (animationState == 6){
        matrix.fillScreen(0);
        SnowFlake6(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }
      if (animationState == 7){
        matrix.fillScreen(0);
        SnowFlake7(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }
      if (animationState == 8){
        matrix.fillScreen(0);
        SnowFlake8(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }


}

void SnowFlake1(uint32_t c){
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawPixel(3, 3, 0); // x, y, color
}

void SnowFlake2(uint32_t c){
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
```

```
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawLine(1, 1, 5, 5, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, c); // x0, y0, x1, y1, color
  matrix.drawPixel(3, 3, 0); // x, y, color
}

void SnowFlake3(uint32_t c){
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 4, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, c); // x0, y0, x1, y1, color
  matrix.drawPixel(3, 3, 0); // x, y, color
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawLine(2, 6, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
}

void SnowFlake4(uint32_t c){
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 1, 5, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 4, 2, 5, c); // x0, y0, x1, y1, color
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color

}

void SnowFlake5(uint32_t c){
  matrix.fillRect(1, 1, 5, 5, c); // x0, y0, width, height
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 3, 3, 1, 0); // x0, y0, x1, y1, color
  matrix.drawLine(3, 5, 5, 3, 0); // x0, y0, x1, y1, color
  matrix.drawPixel(2, 4, 0); // x, y, color
  matrix.drawPixel(4, 2, 0); // x, y, color

}

void SnowFlake6(uint32_t c){
  matrix.fillRect(1, 1, 5, 5, c); // x0, y0, width, height
  matrix.drawLine(1, 1, 5, 5, 0); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, 0); // x0, y0, x1, y1, color
  matrix.drawPixel(0, 3, c); // x, y, color
  matrix.drawPixel(3, 0, c); // x, y, color
  matrix.drawPixel(3, 6, c); // x, y, color
  matrix.drawPixel(6, 3, c); // x, y, color
}

void SnowFlake7(uint32_t c){
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawPixel(3, 3, 0); // x, y, color
  matrix.drawFastHLine(2, 0, 3, c); // x0, y0, length, color
  matrix.drawFastHLine(2, 6, 3, c); // x0, y0, length, color
  matrix.drawFastVLine(0, 2, 3, c); // x0, y0, length, color
  matrix.drawFastVLine(6, 2, 3, c); // x0, y0, length, color
}

void SnowFlake8(uint32_t c){
  //four corners
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color
```

```
  //upper left corner
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //upper right corner
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 0, 6, 3, c); // x0, y0, x1, y1, color

  //lower left corner
  matrix.drawLine(0, 3, 3, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color

  //lower right corner
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
}

void SnowFlake9(uint32_t c){
  //four corners
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //four boxes near center
  matrix.drawRect(1, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(1, 4, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 4, 2, 2, c); // x0, y0, width, height

  //clear out corner pixel of boxes
  matrix.drawPixel(1, 1, 0); // x, y, color
  matrix.drawPixel(5, 1, 0); // x, y, color
  matrix.drawPixel(1, 5, 0); // x, y, color
  matrix.drawPixel(5, 5, 0); // x, y, color
}

void SnowFlake10(uint32_t c){
  //lines across the corners
  matrix.drawLine(0, 1, 1, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(5, 0, 6, 1, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 5, 1, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(5, 6, 6, 5, c); // x0, y0, x1, y1, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //four boxes near center
  matrix.drawRect(1, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(1, 4, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 4, 2, 2, c); // x0, y0, width, height

  //clear out corner pixel of boxes
  matrix.drawPixel(1, 1, 0); // x, y, color
  matrix.drawPixel(5, 1, 0); // x, y, color
  matrix.drawPixel(1, 5, 0); // x, y, color
  matrix.drawPixel(5, 5, 0); // x, y, color
}

void SnowFlake11(uint32_t c){
  //corner lines
```

```
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color

  //center X
  matrix.drawLine(2, 2, 4, 4, c); // x0, y0, x1, y1, color
  matrix.drawLine(2, 4, 4, 2, c); // x0, y0, x1, y1, color

}

//8x8
void SnowFlake12(uint32_t c){
  matrix.drawLine(1, 1, 6, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 6, 6, 1, c); // x0, y0, x1, y1, color
  matrix.fillRect(3, 0, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(0, 3, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(6, 3, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(3, 6, 2, 2, c); // x0, y0, width, height
  matrix.show();

}

void showAllSnowflakes(uint32_t c){
        SnowFlake1(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake5(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake2(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake8(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake3(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake9(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake4(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake10(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake6(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake11(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake7(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake11(matrix.Color(red, green, blue));
```

```
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
}
```

FLORA

For using FLORA (https://adafru.it/EI5), this code has been formatted to work with the FLORA BLE module over hardware serial. Click Download: Project Zip in the .ino file listing below.

```
// SPDX-FileCopyrightText: 2019 Anne Barela for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*********************************************************************
 This is an example for our nRF51822 based Bluefruit LE modules
 Pick one up today in the adafruit shop!
 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!
 MIT license, check LICENSE for more information
 All text above, and the splash screen below must be included in
 any redistribution
*********************************************************************/

#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#if not defined (_VARIANT_ARDUINO_DUE_X_) && not defined (_VARIANT_ARDUINO_ZERO_)
  #include <SoftwareSerial.h>
#endif

#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"

#include "BluefruitConfig.h"

#include <Adafruit_NeoPixel.h>
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>

/*=========================================================================
    APPLICATION SETTINGS
    FACTORYRESET_ENABLE       Perform a factory reset when running this sketch

                              Enabling this will put your Bluefruit LE module
                              in a 'known good' state and clear any config
                              data set in previous sketches or projects, so
                              running this at least once is a good idea.

                              When deploying your project, however, you will
                              want to disable factory reset by setting this
                              value to 0.  If you are making changes to your
                              Bluefruit LE device via AT commands, and those
                              changes aren't persisting across resets, this
                              is the reason why.  Factory reset will erase
                              the non-volatile memory where config data is
                              stored, setting it back to factory default
                              values.

                              Some sketches that require you to bond to a
                              central device (HID mouse, keyboard, etc.)
```

```
                            won't work at all with this feature enabled
                            since the factory reset will clear all of the
                            bonding data stored on the chip, meaning the
                            central device won't be able to reconnect.

    MATRIX DECLARATION      Parameter 1 = width of NeoPixel matrix
                            Parameter 2 = height of matrix
                            Parameter 3 = pin number (most are valid)
                            Parameter 4 = matrix layout flags, add together as
needed:
    NEO_MATRIX_TOP,
    NEO_MATRIX_BOTTOM,
    NEO_MATRIX_LEFT,
    NEO_MATRIX_RIGHT        Position of the FIRST LED in the matrix; pick two,
e.g.
                            NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left
corner.
                            NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are
arranged in horizontal
                            rows or in vertical columns, respectively; pick one
or the other.
    NEO_MATRIX_PROGRESSIVE,
    NEO_MATRIX_ZIGZAG       all rows/columns proceed in the same order,
                            or alternate lines reverse direction; pick one.
                            See example below for these values in action.

    Parameter 5 = pixel type flags, add together as needed:

    NEO_KHZ800              800 KHz bitstream (most NeoPixel products w/WS2812
LEDs)
    NEO_KHZ400              400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811
drivers)
    NEO_GRB                 Pixels are wired for GRB bitstream (most NeoPixel
products)
    NEO_RGB                 Pixels are wired for RGB bitstream (v1 FLORA pixels,
not v2)
    ---------------------------------------------------------------------*/
    #define FACTORYRESET_ENABLE     1

    #define PIN                     6   // Which pin on the Arduino is connected to
the NeoPixels?

// Example for NeoPixel 8x8 Matrix.  In this application we'd like to use it
// with the back text positioned along the bottom edge.
// When held that way, the first pixel is at the top left, and
// lines are arranged in columns, zigzag order.  The 8x8 matrix uses
// 800 KHz (v2) pixels that expect GRB color data.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_ZIGZAG,
  NEO_GRB            + NEO_KHZ800);
/*=========================================================================*/



// Create the bluefruit object, either software serial...uncomment these lines
/*
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN,
BLUEFRUIT_SWUART_RXD_PIN);
Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
*/

/* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line
*/
 Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);

/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/
IRQ/RST */
```

```
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ,
BLUEFRUIT_SPI_RST);

/* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user
selected CS/IRQ/RST */
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
//                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
//                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);


// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  while (1);
}

// function prototypes over in packetparser.cpp
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout);
float parsefloat(uint8_t *buffer);
void printHex(const uint8_t * data, const uint32_t numBytes);

// the packet buffer
extern uint8_t packetbuffer[];


/**************************************************************************/
/*!
    @brief  Sets up the HW an the BLE module (this function is called
            automatically on startup)
*/
/**************************************************************************/
//additional variables

//Color
    uint8_t red = 100;
    uint8_t green = 100;
    uint8_t blue = 100;

    uint8_t animationState = 1;

void setup(void)
{
  //while (!Serial);  // required for Flora & Micro
  delay(500);

  matrix.begin();
  matrix.setBrightness(40);

  matrix.fillScreen(0);
  showAllSnowflakes(matrix.Color(red, green, blue));
  matrix.show(); // This sends the updated pixel colors to the hardware.

  Serial.begin(115200);
  Serial.println(F("Adafruit Bluefruit NeoMatrix Snowflake"));
  Serial.println(F("---------------------------------------------"));

  /* Initialise the module */
  Serial.print(F("Initialising the Bluefruit LE module: "));

  if ( !ble.begin(VERBOSE_MODE) )
  {
    error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check
wiring?"));
  }
  Serial.println( F("OK!") );

  if ( FACTORYRESET_ENABLE )
  {
    /* Perform a factory reset to make sure everything is in a known state */
```

```
    Serial.println(F("Performing a factory reset: "));
    if ( ! ble.factoryReset() ){
      error(F("Couldn't factory reset"));
    }
  }

  /* Disable command echo from Bluefruit */
  ble.echo(false);

  Serial.println("Requesting Bluefruit info:");
  /* Print Bluefruit information */
  ble.info();

  Serial.println(F("Please use Adafruit Bluefruit LE app to connect in Controller
mode"));
  Serial.println(F("Then activate/use the color picker and controller!"));
  Serial.println();

  ble.verbose(false);  // debug info is a little annoying after this point!

  /* Wait for connection */
  while (! ble.isConnected()) {
      delay(500);
  }

  Serial.println(F("***********************"));

  // Set Bluefruit to DATA mode
  Serial.println( F("Switching to DATA mode!") );
  ble.setMode(BLUEFRUIT_MODE_DATA);

  Serial.println(F("***********************"));

}

/**************************************************************************/
/*!
    @brief  Constantly poll for new command or response data
*/
/**************************************************************************/
void loop(void)
{

  /* Wait for new data to arrive */
  uint8_t len = readPacket(&ble, BLE_READPACKET_TIMEOUT);
  //if (len == 0) return;

  /* Got a packet! */
  // printHex(packetbuffer, len);

  // Color
  if (packetbuffer[1] == 'C') {
    red = packetbuffer[2];
    green = packetbuffer[3];
    blue = packetbuffer[4];
    Serial.print ("RGB #");
    if (red < 0x10) Serial.print("0");
    Serial.print(red, HEX);
    if (green < 0x10) Serial.print("0");
    Serial.print(green, HEX);
    if (blue < 0x10) Serial.print("0");
    Serial.println(blue, HEX);
  }

  // Buttons
  if (packetbuffer[1] == 'B') {

    uint8_t buttnum = packetbuffer[2] - '0';
    boolean pressed = packetbuffer[3] - '0';
```

```
    Serial.print ("Button "); Serial.print(buttnum);
    animationState = buttnum;
    if (pressed) {
      Serial.println(" pressed");
    } else {
      Serial.println(" released");
    }
  }

      if (animationState == 1){ // animate through all the snowflakes
        matrix.fillScreen(0);
        SnowFlake1(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 2){
        matrix.fillScreen(0);
        SnowFlake2(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 3){
        matrix.fillScreen(0);
        SnowFlake3(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 4){
        matrix.fillScreen(0);
        SnowFlake4(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }

      if (animationState == 5){
        matrix.fillScreen(0);
        SnowFlake5(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }
      if (animationState == 6){
        matrix.fillScreen(0);
        SnowFlake6(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }
      if (animationState == 7){
        matrix.fillScreen(0);
        SnowFlake7(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }
      if (animationState == 8){
        matrix.fillScreen(0);
        SnowFlake8(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
      }


}

void SnowFlake1(uint32_t c){
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawPixel(3, 3, 0); // x, y, color
}

void SnowFlake2(uint32_t c){
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
```

```
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawLine(1, 1, 5, 5, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, c); // x0, y0, x1, y1, color
  matrix.drawPixel(3, 3, 0); // x, y, color
}

void SnowFlake3(uint32_t c){
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 4, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, c); // x0, y0, x1, y1, color
  matrix.drawPixel(3, 3, 0); // x, y, color
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawLine(2, 6, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
}

void SnowFlake4(uint32_t c){
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 1, 5, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 4, 2, 5, c); // x0, y0, x1, y1, color
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color

}

void SnowFlake5(uint32_t c){
  matrix.fillRect(1, 1, 5, 5, c); // x0, y0, width, height
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 3, 3, 1, 0); // x0, y0, x1, y1, color
  matrix.drawLine(3, 5, 5, 3, 0); // x0, y0, x1, y1, color
  matrix.drawPixel(2, 4, 0); // x, y, color
  matrix.drawPixel(4, 2, 0); // x, y, color

}

void SnowFlake6(uint32_t c){
  matrix.fillRect(1, 1, 5, 5, c); // x0, y0, width, height
  matrix.drawLine(1, 1, 5, 5, 0); // x0, y0, x1, y1, color
  matrix.drawLine(1, 5, 5, 1, 0); // x0, y0, x1, y1, color
  matrix.drawPixel(0, 3, c); // x, y, color
  matrix.drawPixel(3, 0, c); // x, y, color
  matrix.drawPixel(3, 6, c); // x, y, color
  matrix.drawPixel(6, 3, c); // x, y, color
}

void SnowFlake7(uint32_t c){
  matrix.drawRect(2, 2, 3, 3, c); // x0, y0, width, height
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawFastVLine(3, 0, 7, c); // x0, y0, length, color
  matrix.drawFastHLine(0, 3, 7, c); // x0, y0, length, color
  matrix.drawPixel(3, 3, 0); // x, y, color
  matrix.drawFastHLine(2, 0, 3, c); // x0, y0, length, color
  matrix.drawFastHLine(2, 6, 3, c); // x0, y0, length, color
  matrix.drawFastVLine(0, 2, 3, c); // x0, y0, length, color
  matrix.drawFastVLine(6, 2, 3, c); // x0, y0, length, color
}

void SnowFlake8(uint32_t c){
  //four corners
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color
```

```
  //upper left corner
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 3, 3, 0, c); // x0, y0, x1, y1, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //upper right corner
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(3, 0, 6, 3, c); // x0, y0, x1, y1, color

  //lower left corner
  matrix.drawLine(0, 3, 3, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color

  //lower right corner
  matrix.drawLine(3, 6, 6, 3, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color
}

void SnowFlake9(uint32_t c){
  //four corners
  matrix.drawPixel(0, 0, c); // x, y, color
  matrix.drawPixel(0, 6, c); // x, y, color
  matrix.drawPixel(6, 0, c); // x, y, color
  matrix.drawPixel(6, 6, c); // x, y, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //four boxes near center
  matrix.drawRect(1, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(1, 4, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 4, 2, 2, c); // x0, y0, width, height

  //clear out corner pixel of boxes
  matrix.drawPixel(1, 1, 0); // x, y, color
  matrix.drawPixel(5, 1, 0); // x, y, color
  matrix.drawPixel(1, 5, 0); // x, y, color
  matrix.drawPixel(5, 5, 0); // x, y, color
}

void SnowFlake10(uint32_t c){
  //lines across the corners
  matrix.drawLine(0, 1, 1, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(5, 0, 6, 1, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 5, 1, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(5, 6, 6, 5, c); // x0, y0, x1, y1, color

  //center dot
  matrix.drawPixel(3, 3, c); // x, y, color

  //four boxes near center
  matrix.drawRect(1, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 1, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(1, 4, 2, 2, c); // x0, y0, width, height
  matrix.drawRect(4, 4, 2, 2, c); // x0, y0, width, height

  //clear out corner pixel of boxes
  matrix.drawPixel(1, 1, 0); // x, y, color
  matrix.drawPixel(5, 1, 0); // x, y, color
  matrix.drawPixel(1, 5, 0); // x, y, color
  matrix.drawPixel(5, 5, 0); // x, y, color
}

void SnowFlake11(uint32_t c){
  //corner lines
```

```
  matrix.drawLine(0, 2, 2, 0, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 0, 6, 2, c); // x0, y0, x1, y1, color
  matrix.drawLine(0, 4, 2, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(4, 6, 6, 4, c); // x0, y0, x1, y1, color

  //center X
  matrix.drawLine(2, 2, 4, 4, c); // x0, y0, x1, y1, color
  matrix.drawLine(2, 4, 4, 2, c); // x0, y0, x1, y1, color

}

//8x8
void SnowFlake12(uint32_t c){
  matrix.drawLine(1, 1, 6, 6, c); // x0, y0, x1, y1, color
  matrix.drawLine(1, 6, 6, 1, c); // x0, y0, x1, y1, color
  matrix.fillRect(3, 0, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(0, 3, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(6, 3, 2, 2, c); // x0, y0, width, height
  matrix.fillRect(3, 6, 2, 2, c); // x0, y0, width, height
  matrix.show();

}

void showAllSnowflakes(uint32_t c){
        SnowFlake1(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake5(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake2(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake8(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake3(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake9(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake4(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake10(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake6(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake11(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake7(matrix.Color(red, green, blue));
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
        matrix.fillScreen(0);
        SnowFlake11(matrix.Color(red, green, blue));
```

```
        matrix.show(); // This sends the updated pixel colors to the hardware.
        delay(500);
}
```

Sew into Sweater & Wear!

To temporarily attach the matrix to your sweater, use a needle and thread to tack down the corners of the matrix. Plug your battery in through a JST extension cable so you can tuck the battery into your pocket. Wear over another shirt so the circuit is not shorted by your skin!