adafruit learning system

Bluetooth-Controlled NeoPixel Goggles

Created by Phillip Burgess

Last updated on 2015-05-13 10:10:09 AM EDT

Guide Contents

Guide Contents 2
Overview 3
How it Works 9

Overview

This is *not* a full project, but rather a follow-up, an add-on to a previous guide. (http://adafru.it/e44)

Our goal is to show how our smartphone app can remote-control even the tiniest of microcontroller projects. To do that, we'll build upon our popular *Kaleodoscope Eyes* NeoPixel goggles project (http://adafru.it/fam), adding the Bluefruit LE UART Friend (http://adafru.it/fao) board to the mix. Change colors to suit your mood…from your smartphone!

To be frank, this is not a very practical extension…there's very little code space left on the Trinket microcontroller for additional modes…but it demonstrates the core idea that adding basic Bluetooth controls no longer requires the most advanced microcontroller or lots of code. Our Bluefruit LE UART Friend (http://adafru.it/2479) and accompanying free *Bluefruit LE Connect* app for iOS (http://adafru.it/ddu) and Android (http://adafru.it/f4G) make it easy! If size and cost are at a premium, Trinket is at least an *option*…and it only gets better with Pro Trinket on up.

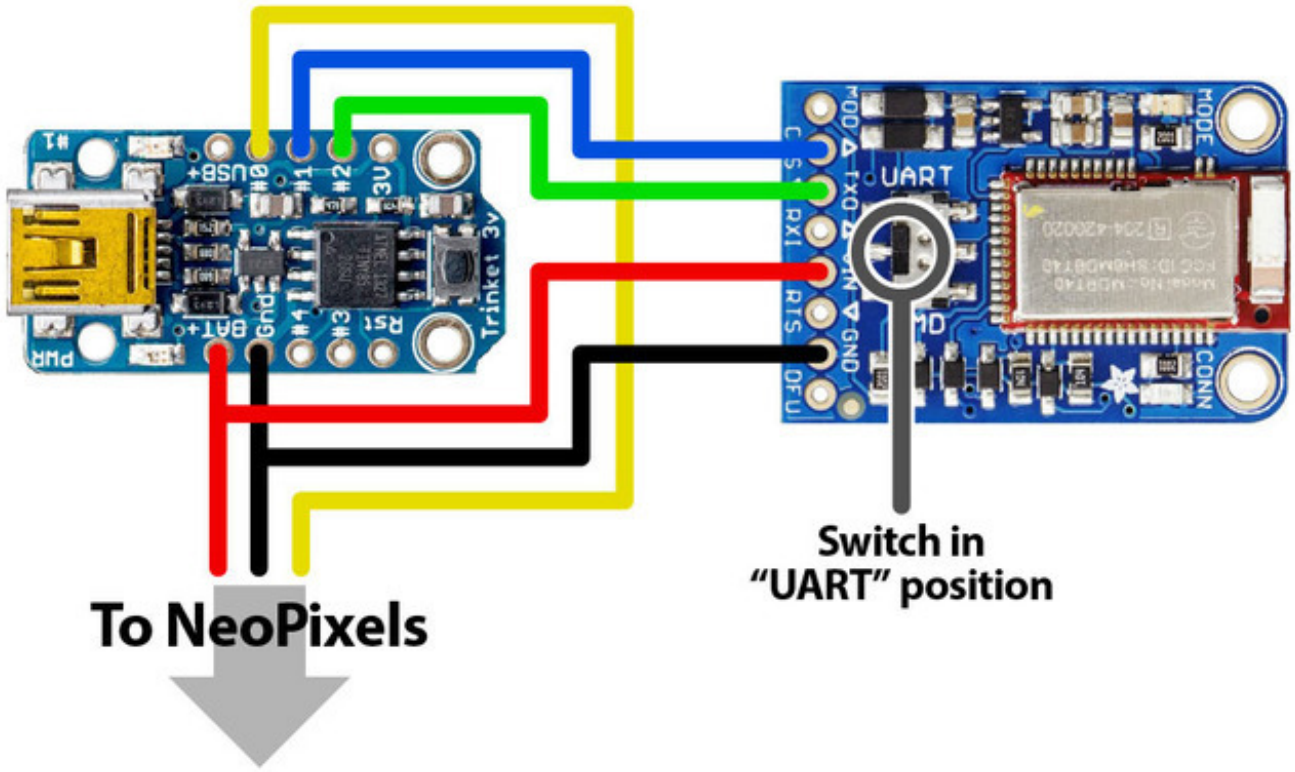
If you've not built the basic NeoPixel goggles, see the *Kaleidoscope Eyes* (http://adafru.it/fap) guide for some background.

Wiring is nearly the same as the original goggles project, with just a few additions…

Trinket Pin	Bluefruit Pin
1	CTS
2	TXO
BAT+	VIN
GND	GND

As in the original project, NeoPixels connect to Trinket pin 0, BAT+ and GND.

BAT+ and GND now need to connect *two* places — the NeoPixels and the Bluefruit module. With sufficiently narrow wire (e.g. 26 gauge) you may be able to fit two wires through each via on the Trinket board…otherwise, you'll need to do a *three-way splice* to get power to both places.



The switch on the Bluefruit LE UART Friend should be in the "UART" position.

Copy and paste the following code into a new Arduino sketch, then upload to the Trinket board as you normally would.

Scroll down past the code for directions for use, an explanation of how it works and how to apply these techniques to your own projects…

```
// Bluetooth Goggles Sketch -- shows the Adafruit Bluefruit LE UART Friend
// can be used even with Trinket or Gemma!

// https://www.adafruit.com/products/2479

// Works in conjunction with Bluefruit LE Connect app on iOS or Android --
// pick colors or use '1' and '2' buttons to select pinwheel or sparkle modes.
// You can try adding more, but space is VERY tight...helps to use Arduino IDE
// 1.6.4 or later; produces slightly smaller code than the 1.0.X releases.

// BLUEFRUIT LE UART FRIEND MUST BE SWITCHED TO 'UART' MODE

#include <SoftwareSerial.h>
#include <Adafruit_NeoPixel.h>
#ifdef __AVR_ATtiny85__ // Trinket, Gemma, etc.
 #include <avr/power.h>
#endif

#define RX_PIN    2 // Connect this Trinket pin to BLE 'TXO' pin
#define CTS_PIN   1 // Connect this Trinket pin to BLE 'CTS' pin
#define LED_PIN   0 // Connect NeoPixels to this Trinket pin
#define NUM_LEDS 32 // Two 16-LED NeoPixel rings
#define FPS      30 // Animation frames/second (ish)

SoftwareSerial    ser(RX_PIN, -1);
Adafruit_NeoPixel pixels(NUM_LEDS, LED_PIN);

void setup() {
#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000L)
  // MUST do this on 16 MHz Trinket for serial & NeoPixels!
  clock_prescale_set(clock_div_1);
#endif
  // Stop incoming data & init software serial
  pinMode(CTS_PIN, OUTPUT); digitalWrite(CTS_PIN, HIGH);
  ser.begin(9600);

  pixels.begin(); // NeoPixel init
  // Flash space is tight on Trinket/Gemma, so setBrightness() is avoided --
  // it adds ~200 bytes.  Instead the color picker input is 'manually' scaled.
}

uint8_t  buf[3],             // Enough for RGB parse; expand if using sensors
         animMode = 0,       // Current animation mode
         animPos  = 0;       // Current animation position
uint32_t color    = 0x400000, // Current animation color (red by default)
```

```
  prevTime = 0L;       // For animation timing

void loop(void) {
  int      c;
  uint32_t t;

  // Animation happens at about 30 frames/sec.  Rendering frames takes less
  // than that, so the idle time is used to monitor incoming serial data.
  digitalWrite(CTS_PIN, LOW); // Signal to BLE, OK to send data!
  for(;;) {
    t = micros();                          // Current time
    if((t - prevTime) >= (1000000L / FPS)) { // 1/30 sec elapsed?
      prevTime = t;
      break;                               // Yes, go update LEDs
    }                                      // otherwise...
    if((c = ser.read()) == '!') {          // Received UART app input?
      while((c = ser.read()) < 0);         // Yes, wait for command byte
      switch(c) {
       case 'B':       // Button (Control Pad)
        if(readAndCheckCRC(255-'!'-'B', buf, 2) & (buf[1] == '1')) {
          buttonPress(buf[0]); // Handle button-press message
        }
        break;
       case 'C':       // Color Picker
        if(readAndCheckCRC(255-'!'-'C', buf, 3)) {
          // As mentioned earlier, setBrightness() was avoided to save space.
          // Instead, results from the color picker (in buf[]) are divided
          // by 4; essentially equivalent to setBrightness(64).  This is to
          // improve battery run time (NeoPixels are still plenty bright).
          color = pixels.Color(buf[0]/4, buf[1]/4, buf[2]/4);
        }
        break;
       case 'Q':       // Quaternion
        skipBytes(17); // 4 floats + CRC (see note below re: parsing)
        break;
       case 'A':       // Accelerometer
#if 0
        // The phone sensors are NOT used by this sketch, but this shows how
        // they might be read.  First, buf[] must be delared large enough for
        // the expected data packet (minus header & CRC) -- that's 16 bytes
        // for quaternions (above), or 12 bytes for most of the others.
        // Second, the first arg to readAndCheckCRC() must be modified to
        // match the data type (e.g. 'A' here for accelerometer).  Finally,
        // values can be directly type-converted to float by using a suitable
        // offset into buf[] (e.g. 0, 4, 8, 12) ... it's not used in this
        // example because floating-point math uses lots of RAM and code
```

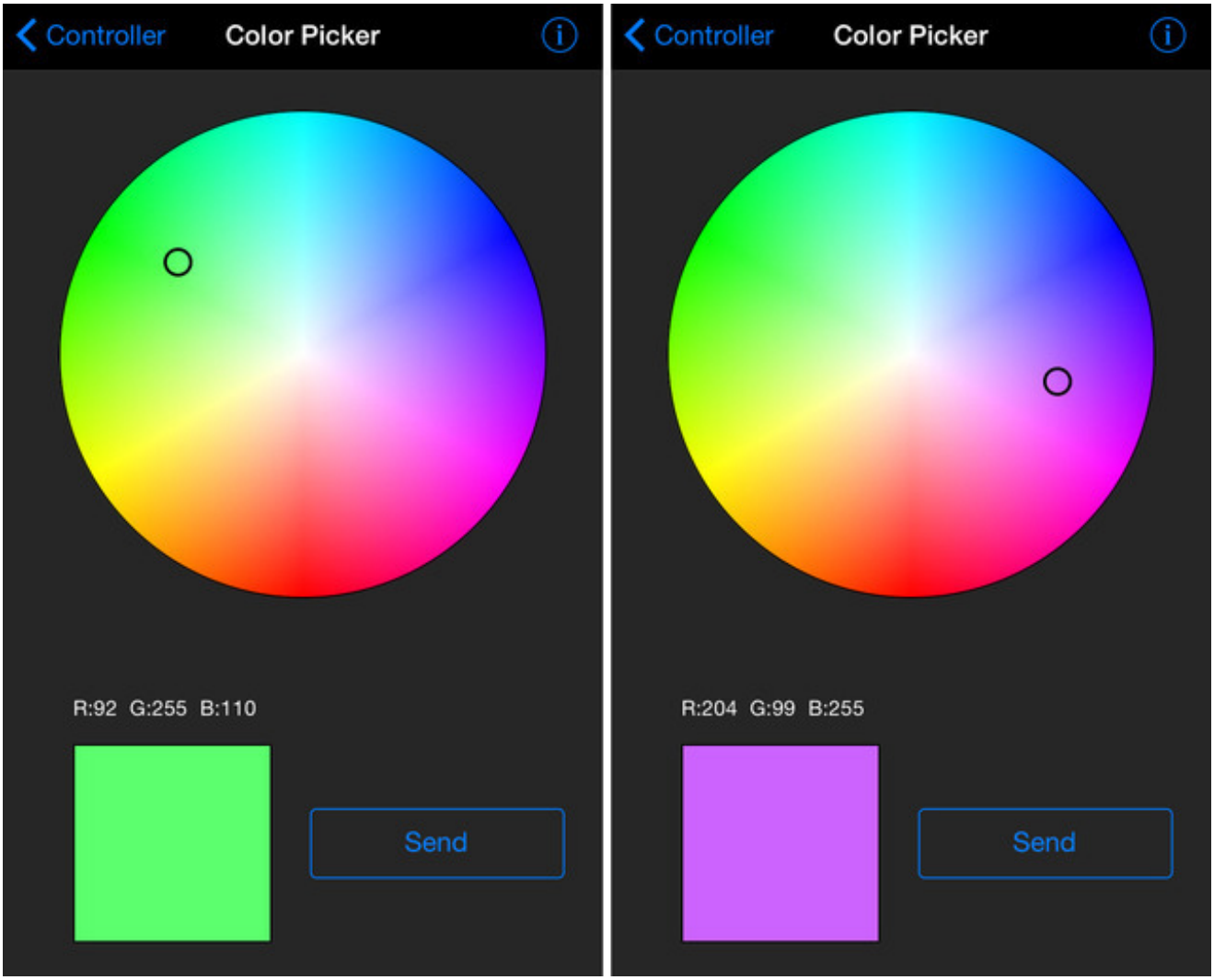
```
        // space, not suitable for the space-constrained Trinket/Gemma, but
        // maybe you're using a Pro Trinket, Teensy, etc.
        if(readAndCheckCRC(255-'!'-'A', buf, 12)) {
          float x = *(float *)(&buf[0]),
                y = *(float *)(&buf[4]),
                z = *(float *)(&buf[8]);
        }
        // In all likelihood, updates from the buttons and color picker
        // alone are infrequent enough that you could do without any mention
        // of the CTS pin in this code.  It's the extra sensors that really
        // start the firehose of data.
        break;
#endif
       case 'G':       // Gyroscope
       case 'M':       // Magnetometer
       case 'L':       // Location
        skipBytes(13); // 3 floats + CRC
      }
    }
  }
  digitalWrite(CTS_PIN, HIGH); // BLE STOP!

  // Show pixels calculated on *prior* pass; this ensures more uniform timing
  pixels.show();

  // Then calculate pixels for *next* frame...
  switch(animMode) {
   case 0: // Pinwheel mode
    for(uint8_t i=0; i<NUM_LEDS/2; i++) {
      uint32_t c = 0;
      if(((animPos + i) & 7) < 2) c = color; // 4 pixels on...
      pixels.setPixelColor(   i, c);         // First eye
      pixels.setPixelColor(NUM_LEDS-1-i, c); // Second eye (flipped)
    }
    animPos++;
    break;
   case 1: // Sparkle mode
    pixels.setPixelColor(animPos, 0);     // Erase old dot
    animPos = random(NUM_LEDS);           // Pick a new one
    pixels.setPixelColor(animPos, color); // and light it
    break;
  }
}

boolean readAndCheckCRC(uint8_t sum, uint8_t *buf, uint8_t n) {
  for(int c;;) {
    while((c = ser.read()) < 0); // Wait for next byte
```

```
    if(!n--) return (c == sum);  // If CRC byte, we're done
    *buf++ = c;                  // Else store in buffer
    sum   -= c;                  // and accumulate sum
  }
}

void skipBytes(uint8_t n) {
  while(n--) {
    while(ser.read() < 0);
  }
}

void buttonPress(char c) {
  pixels.clear(); // Clear pixel data when switching modes (else residue)
  switch(c) {
   case '1':
    animMode = 0; // Switch to pinwheel mode
    break;
   case '2':
    animMode = 1; // Switch to sparkle mode
    break;
   case '3':
    break;
   case '4':
    break;
   case '5': // Up
    break;
   case '6': // Down
    break;
   case '7': // Left
    break;
   case '8': // Right
    break;
  }
}
```

The *Bluefruit LE Connect* app for iOS is explained in this guide (http://adafru.it/f9i). A version is now available for Android (http://adafru.it/f4G) as well. Either one requires a phone or tablet with support for Bluetooth 4.0 (e.g. iPhone 4S or later, second-generation Nexus 7 tablet, etc.).

When you run the *Bluefruit LE Connect* app and scan for devices, the Bluefruit UART Friend will show up as “Adafruit Bluefruit LE.” A more elaborate setup with TX, RX and MODE pins connected could change this, but we’re aiming for small, minimalist code and the least wires here, we’ll use the default.

Tap “Connect,” then “Controller,” then either the “Color Picker” or “Control Pad” interfaces. The color picker is pretty straightforward — select a color from the wheel and tap “send,” and the goggles should immediately switch to this new color. On the control pad screen, only the “1” and “2” buttons work in this example — one selects the “pinwheel” animation from the original goggles project, the other the “sparkle” effect. There’s a little bit of space left on the Trinket…additional controls could be added *if you can keep the code very simple.*



How it Works

The code is based on the original goggles project, with some additions for Bluetooth support.

The *SoftwareSerial* library (included standard with the Arduino IDE) is used to receive data from the Bluetooth module at 9600 bits/second. With the switch set to “UART,” it just looks to us like a plain serial connection. To minimize the number of wires, we’re only using the receive end; no transmit pin is defined or connected.

Data from the *Bluefruit LE Connect* app is encoded in short packets, always starting with a "!" character and ending with a checksum. Given the limited space on the Trinket, the code for interpreting these packets has been stripped bare…only colors and button presses are handled right now, but there are some notes in there for extending this to other sensor data from the phone…this will almost certainly require a more capable microcontroller (e.g. Pro Trinket or larger).

With limited RAM on Trinket, we can't afford a large buffer to store data received over Bluetooth. The *CTS* pin is used so the UART sends data only when we can handle it. Set LOW, the module sends data; HIGH and data is stopped…queued up on the module but not sent yet. Since we're only using the app's button and color interfaces, the data stream is fairly sparse and we could probably skip the CTS pin (just connect to GND) and it would still work…but more sophisticated sketches using lots of sensor data will require it.

The animation is synchronized to run at about 30 frames per second by using the micros() function and comparing against the prior frame (rather than using delay()). Drawing each frame takes much less time than that, so we exploit this "dead time" for polling the UART connection…this provides smoother handling of the data and the animation than if we used delay().

The code is pretty thoroughly commented to explain some of the shenanigans needed to keep things simple on the Trinket (e.g. avoiding setBrightness()).

If you decide to extend this with more modes or features and still run on a Trinket, it's recommended to use version 1.6.4 or later of the Arduino IDE — it produces code about 10% smaller than the 1.0.X or earlier releases. This guide shows how to add Trinket support (http://adafru.it/fan) to the IDE in just a few steps. And Gemma support is already built-in!

WTF those goggles in the picture?!

That's a *very* custom set I made for myself…German safety goggles from Restoration Hardware, laser-cut "infinity mirrors" and 144 LED/m NeoPixel strip. *The exact same Trinket microcontroller and Bluefruit LE UART Friend are inside though!* From a code standpoint, this all works the same on the standard goggles kit. (Check out our Pocket Galaxy tutorial on how to create an infinity mirror w/NeoPixels (http://adafru.it/faz))