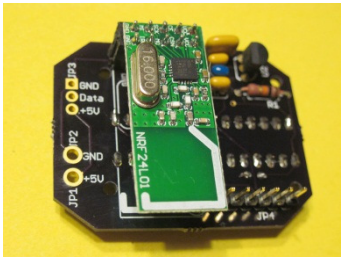
11/25/2013



RFColor2_4 “How-To”

Steps and resources to build a wireless GECE controller

joej85, komby, jchuchla, dmcole, kingofky, CaptKirk

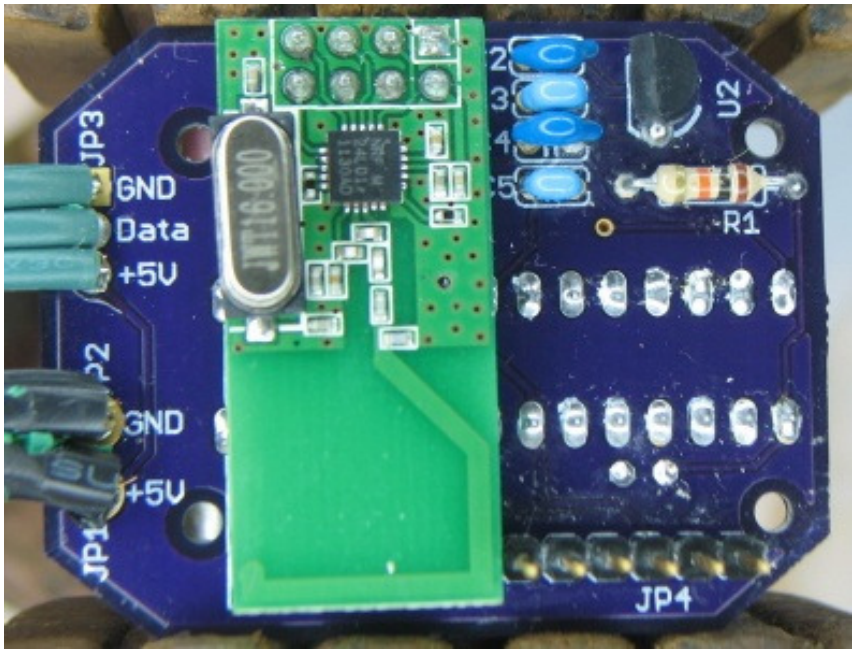
Table of Contents

RFColor2_4

Steps and resources to build a wireless GECE controller

1. Introduction

The RFColor2_4 is an Arduino based wireless controller for GE Color Effects pixel strings designed by Joe Johnson - joej85 and software enhanced by Komby. It is an Arduino based design emulating an Arduino Uno in order to use the vast libraries of Arduino code available. This document will attempt to document how to put together a working setup.



Note: Throughout this document are included some links to EBay items that may or may not still be available. Be aware you may need to search for more current listings for some of those items.

2. What You Will Need:

What	Qty	Where	Notes
Compile instructions	1	See here: *7. Code: Compile instructions*	Documents what is needed to handle the code
Transmitter Code	1	http://doityourselfchristmas.com/forums/attachment.php?attachmentid=17901&d=1359258048	Rename so extension is .ino
Receiver code	1	http://doityourselfchristmas.com/forums/attachment.php?attachmentid=17902&d=1359258141	Rename so extension is .ino
Arduino IDE 1.0		http://arduino.cc/en/Main/Software	Used to upload the TX and RX code to the board. The older 1.0 version is currently recommended.
RF24 library for the NRF24L01+ module		http://maniacbug.github.com/RF24/index.html	
GEColorEffects library		http://www.digitalmisery.com/wp-content/uploads/2011/11/GEColorEffects-Arduino-1.0.zip	1.0 version requires less changes.
digitalWriteFast library		http://code.google.com/p/digitalwritefast/	
FTDI Serial cable		http://www.ebay.com/itm/6pin-FTDI-FT232RL-USB-to-Serial-adapter-module-USB-TO-TTL-RS232-Arduino-Cable-/400356015296	This will allow for loading the code thru the Arduino bootloader and running the transmitter from light control software like Vixen as OpenDMX.
DMX Dongle w/ RS485 to TTL converter		See the DIYC website for sources	Used from light control software as an alternative to the less reliable OpenDMX operation. Set software to Enttec Pro.
BOM		See *3. RFColor2_4 PCB BOM*	1 per board
PCB	2	RFColor2_4_final_6_23_12_no_gp.brd http://doityourselfchristmas.com/forums/attachment.php?attachmentid=17899&d=1359257665	TX and RX use the same PCB. See below for order info

3. RFColor2_4 PCB BOM:

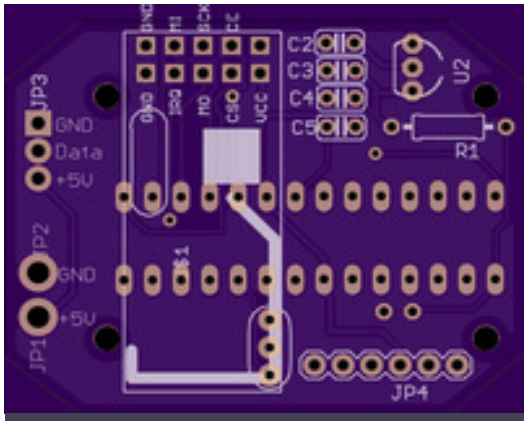
RFColor2_4 V1.0 BOM September 23, 2012
Note prices change- please use the cost column as a guide, not as an absolute!!

Ref	Qty	Manu part #	Package	Desc	Supplier	Supplier part #	Unit Cost	Total Cost
C1,C3, C5	3	21RZ310-RC	2.5mm	0.1uF 50v Ceramic Capacitor	Mouser	21RZ310-RC	$0.08	$0.24
C2	1	FK18X5R1E105K	2.5mm	1.0uF 25V Ceramic Capacitor	Mouser	810-FK18X5R1E105K	$0.112	$0.112
C4	1	FK14X5R1E475K	2.5mm	4.7uF 25V Ceramic Capacitor	Mouser	810-FK14X5R1E475K	$0.25	$0.25
X1	1	AWCR-16.00MD	2.5mm	16MHz 5V Resonator	Mouser	815-AWCR-16.00MD	$0.27	$0.27
R1	1	MF1/4DC1002F	1/4W thru-hole	10k 1/4W Resistor	Mouser	660-MF1/4DC1002F	$0.06	$0.06
U1	1	ATMEGA328P-PU	28 Pin DIP	8-bit Microcontroller (MCU)	Mouser	556-ATMEGA328P-PU (Note: does not have an Arduino bootloader!)	$2.13	$2.13
U2	1	MCP1700-3302E/TO	TO-92-3	3.3v 250mA LDO 2% Voltage Regulator	Mouser	579-MCP1700-3302E/TO	$0.370	$0.37
JP4	1	2340-6111TG	6 pins	40P STRT 1 Row Gold Header	Mouser	517-6111TG	$1.76	$0.29
-	1	NRF24L01+ PN LA Module for TX	-	2.4GHz RF Comms Module with removable antenna and amplifier	Various – see *6.NRF24L01+ Module*		$10 up	tbd
-	1	NRF24L01+ for RX	-	2.4GHz RF Comms Module with integrated antenna	Ditto		$1.50 up	tbd
-	1	PCB			OSHPark - See *4.Ordering Boards*			

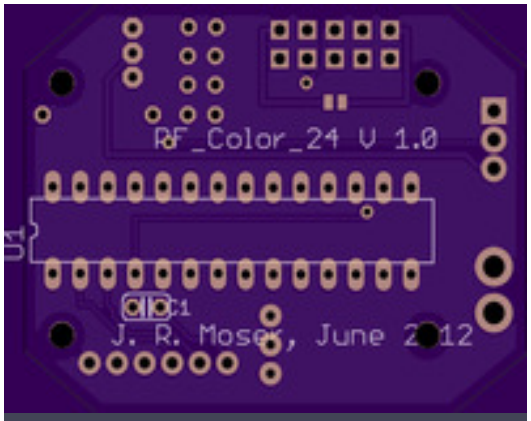
4. Ordering boards:

Please follow these steps if you need to order boards on your own outside of a group buy or getting them from a vendor.

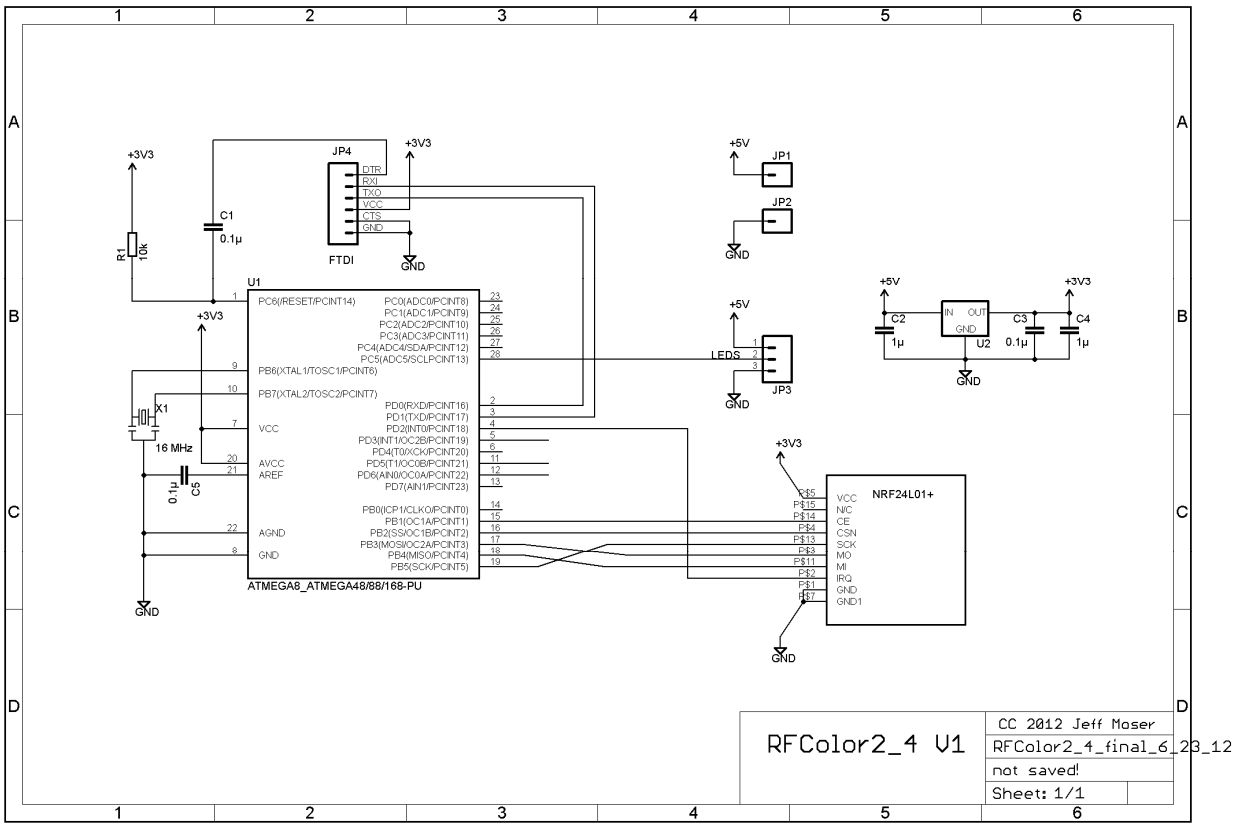
Get the board file (currently "RFColor2_4_final_6_23_12_no_gp.brd") and submit to OHSPark.com. You should be able to get THREE boards made for right around $12 shipped. Just follow the on-screen instructions at OSHPark.com. The boards should look like the following:



Board from OSHPark –TOP VIEW



BOTTOM VIEW



5. ATMEGA328P-PU Arduino Setup

The processor must be setup with the Arduino bootloader for this design to operate. You can purchase parts with the Arduino bootloader already programmed from eBay, Jameco, Virtuabotix https://www.virtuabotix.com/?page_id=3117&productid=0609224531750, and others. This is the recommended way to get them and if you get your Atmega parts in a group buy- there is a good chance the parts will be bootloaded and you can skip this section altogether.

Why might you want to be able to bootload an Atmega? There are a few reasons:

- There have been issues where mailing/shipping parts disrupts the programming. If this happens to you, you might want to be able to reprogram the bootloader.
- Bootloaders, like any code, do get updated periodically. There have been updated bootloaders that take up less space leaving more room for sketches, or fix bugs, or add features, or improve boot time. You may find yourself needing features of a newer bootloader.
- Bootloaded parts tend to cost a bit more than a blank/raw parts- for example Mouser wants $2.50 each for blank 328P parts and on eBay the best price found when this document was written is around $3.50 (shipping included) (http://www.ebay.com/itm/Microcontroller-ATmega328P-PU-with-bootloader-/251113394450) for the "Arduino" pre-programmed ones, so if you are doing a few of these, it may be worth setting up a bootloader programming capability.

*Note: Be aware that if you get the ATMEGA328 rather than the ATMEGA328**P**, (e.g. Atmega328P-PU or Atmega328P-AU) for the dip through hole version of the part used on this board) there are more changes you might need to do to the bootloader code as the "non-P" version of the chip has a different chip ID than the "P" version. That means there are likely code changes in a lot of places that you will need to do to get a bootloader to work. The "non-P" chips can cost a bit less than the "P" version and pull a bit more power but will work as an Arduino if you get the right bootloader installed. However, the "P" version are recommended.*

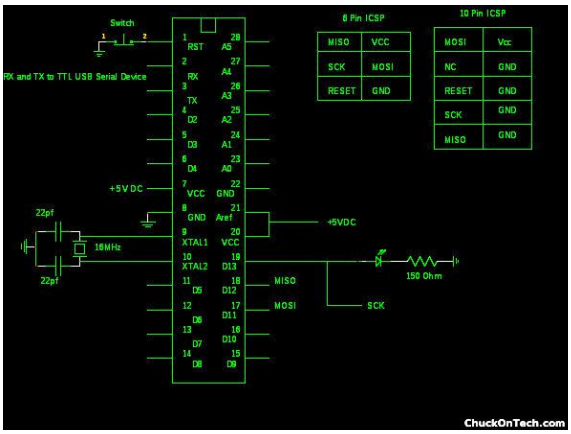
If you opt to get blank parts, there are many procedures to load the Arduino bootloader to the chip that you can find out on the web. This is a fairly advanced process involving bread boarding and code loading. Some programming processes involve using an Arduino Uno board (http://www.3guys1laser.com/blog-burn-bootloader-blank-atmega328atmega328p-arduino-uno), some involve using the AVRISP Mk II programming board or equiv. – see following.

For this document we will discuss the AVR Programmer and Breadboard process as that is what the author had available.

For software, you will need something like AVR Studio and Avr-dude. Also, the AVR-GCC compiler could be helpful (if AVR Studio does not include one). A copy of an Arduino bootloader will also be helpful (detailed later).

http://www.sparkfun.com/tutorials/93 is a great place to get started. Also, there is some good information about the bootloader here: http://arduino.cc/en/Hacking/Bootloader

Start with the Sparkfun tutorial, wire up a breadboard as described there and get the straight-line BLINK code installed and working – this will prove you ARE programming the part properly. Following is another schematic on the 328 pin out and the two styles of AVR Programming port connectors (the 6 pin being more common and used on this board):



For the 10 pin one remember that ALL the GND pins must be wired. Also note that this schematic has the BLINK led on a different pin.

On the Atmel ATmega328p:
Pin 1 is Reset, Pin 17 is MOSI, Pin 18 is MISO, Pin 19 is SCK, Pin 20 and Pin 21 are Vcc, Pin 22 is Ground

If you followed the Sparkfun tutorial (not required), you have Blink working as an inline program but no bootloader yet. The next step is to program in the Arduino bootloader. There are many available and most are interchangeable. The following are recommended:

Optiloader Arduino bootloader: https://github.com/WestfW/OptiLoader

The tutorial for this bootloader (plus the process to use the Uno as programmer) is here:
http://www.3guys1laser.com/blog-burn-bootloader-blank-atmega328atmega328p-arduino-uno

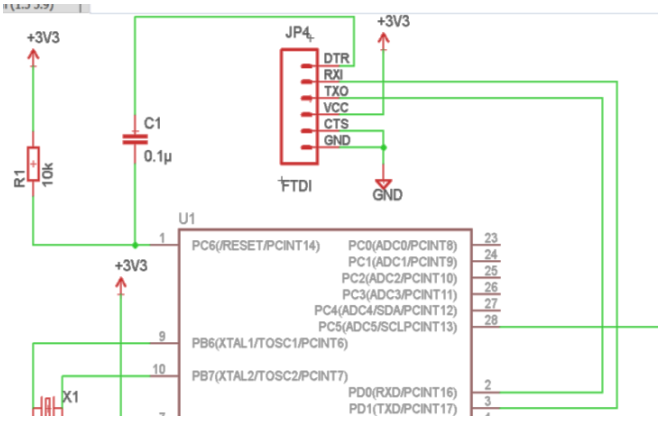
Note: leave the resonator attached like the schematics above despite what the 3guys1laser tutorial says.

Adafruit bootloader: http://www.wulfden.org/downloads/code/arduino/ADABootLoaderR3_9D15.zip contains source, hex files and more information. It is supposed to be improved over the "stock" Arduino bootloader that comes with the ArduinoIDE. Refer to the http://arduino.cc/en/Hacking/Bootloader for info and the process. Also see: http://arduino.cc/en/Main/Standalone down the page a bit for some tips on bootloading.

Once the bootloader is installed, you can disconnect the programmer and try loading the "Blink sketch" contained in the ArduinoIDE through the bootloader and serial cable.

Note: do not use the AVR Programmer to attempt to load the Blink SKETCH as you will overwrite the bootloader. You will want to use the serial connection to upload code through the bootloader once the bootloader is programmed into the part.

Wiring for the serial connections on the breadboard can be duplicated from the JP4 connection in the RFColor2_4 schematic. The recommended FTDI cable plugs straight into this connection.



Basically, Serial cable DTR goes through a .1uf cap to 328 pin 1, RXI to 3, TXO to 2, VCC to 7, CTS and GND to 8. You can also use the RFColor2_4 board with its JP4 connection for the TTL serial port cable but remember that there is no LED on the RFColor PCB to blink (but that can be "fixed" also by tacking an LED and resistor onto pin 13)!

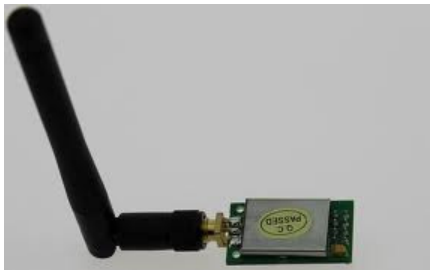
The Blink sketch is loaded using this procedure found here:
http://arduino.cc/en/Guide/Windows#toc5
in the ArduinoIDE. Once the Blink sketch works, you can then replace it with either the TX (controller) code, or RX (receiver) code that will be built in a following section here: Loading the Sketch

6. NRF24L01+ Module

The NRF24L01+ module is a wireless transceiver (transmit and receive) available in a wide range of styles and capabilities. Depending on your layout and controller positions, you can use any combination of modules you see fit. Range and reliability will change depending on a set of factors that cannot be known for your particular situation. If you follow the recommendations, you should have a successful setup.

6.1. NRF Transceiver Types

The various modules have various sources and varying prices. The following are the variations available:

Name	Picture	Where	Notes
green "?"		http://www.ebay.com/itm/2PCS-2-4GHz-NRF24L01-Wireless-Transceiver-Module-NEW-/110929270444	10 pins. Shortest range NOT RECOMMENDED
Black "zig-zag"		http://www.komby.com/nrf24l01-24-ghz-rf http://www.ebay.com/itm/10pcs-NRF24L01-2-4GHz-Antenna-Wireless-Transceiver-Module-for-Arduino-New/400594940658? www.ebay.com/itm/330808881863	8 pins. Best range for inexpensive RX, great price, can be sensitive to power ripple. Requires a mod to the current board to plug in.
Green PA		http://www.ebay.com/itm/nRF24L01-PA-LNA-wireless-communication-modules-w-antenna-2-4GHz-2Mbps-/110980385971 http://dx.com/p/nrf24l01-pa-lna-wireless-communication-modules-w-antenna-for-arduino-148822	8 Pins. Higher cost than the non-PA version because it is amplified and uses an antenna. These will provide longer range but are 5x to 10x the price of the "?" or "zig-zag" modules. You should use one of the PA modules for the TX. Using these for RX increases the receive range a lot as well. This one may need the two extra pins on the board manually connected to ground to operate.
Black PA		http://www.komby.com/nrf24l01-24-ghz-rf-pa-with-antenna om http://www.ebay.com/itm/NRF24L01-PA-LNA-SMA-Antenna-Wireless-Transceiver-communication-module-2-4G-G5-/200963287312?	8 pins. Note: You may need to provide an antenna depending on where you get this. The two here both includes the antenna.

6.2. Range Tests

TX	RX	Range
Green "?"	Green "?"	**30 feet**
Black "zig-zag"	Black "zigzag"	**200 feet**
Black PA	Green "?"	**400 feet**
Black PA	Black "zigzag"	**600 feet**
Black PA	Green PA (inside car)	**>1000 feet**
Green PA	Black PA (inside car)	**>1000 feet**

So, here is the recommendation: in general use the Black "zigzag" modules for receivers and either of the "black PA" or "green PA" modules for the transmitter. The Green "?" modules are not very good at transmitting and only OK at receiving. For longer range, use a "PA" module in both TX and RX.

6.3. Transmitter Configuration

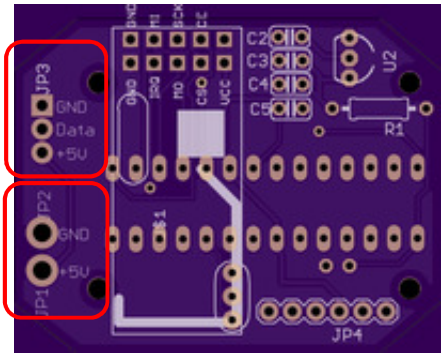
Use the Controller Code (details are in following chapters), with one of either the Black PA or Green PA modules on the board. The 328 will be loaded with the "Controller" code via the serial cable and bootloader.

There are two ways to control this:

1. Serial cable and OpenDMX. It is possible to use the serial cable as the light control interface. This has the advantage that you should already have the serial cable to be able to load the sketches (like Blink) then the TX or RX code so there is no added hardware or fuss. The disadvantage is that OpenDMX is completely at the mercy of the operating system and has a greater chance of dropping data. It requires a fairly "robust" computer to be able to keep up. Also not all light control software supports "OpenDMX" (although Vixen and HLS definitely do). This is a good way to test your board operation, but there is a small chance you could have issues during a show once EVERYTHING is running.
2. DMX dongle, RS485 to TTL converter and Enttec Pro compatible support. The advantage here is that the DMX data stream transmission is managed by the dongle hardware and with a heavily loaded system, is less likely to drop data or mangle the DMX data stream. The disadvantage is that it does require you to have a DMX dongle like the "RPM DMX Dongle" plus you need to do a custom cable with an RS485 to TTL conversion in order for it to work.

6.4. Receiver Configuration

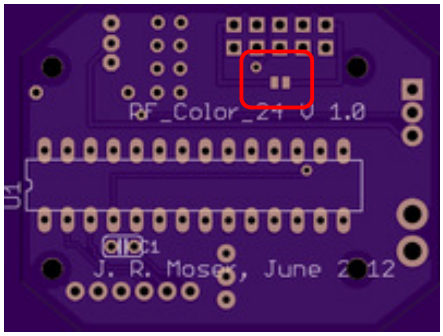
Use the Receiver code and upload to the unit with the serial cable.

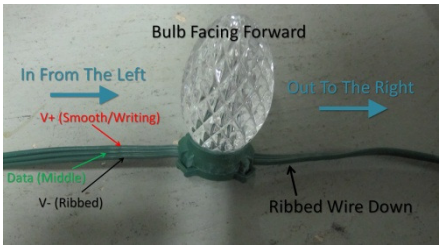


External power – +5V is fed into the pads marked JP1 (+) and JP2 (- / GND) following the polarity marked on the board. *Note: The +5 supply needs enough current to drive the string.* It is possible to use the power supply that came with the strings and mount the receiver in the box once you take out the now useless original electronics.

Strings connect to the JP3 per the pin out silkscreened on the board.

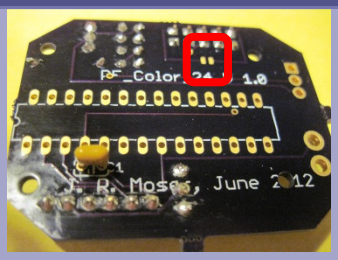
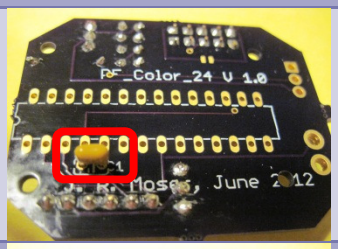
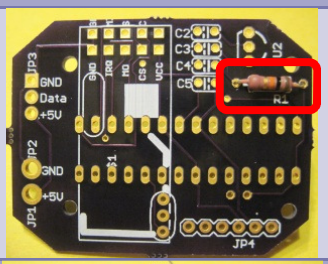
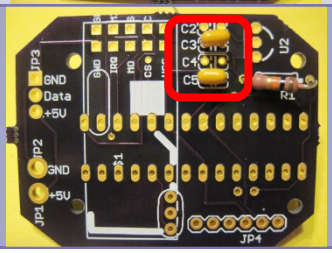
The change to use 8 pin modules is highlighted here. You need to solder across the highlighted pads. *Note: With this jumper, plugging in a 10 pin module WILL result in a short.*

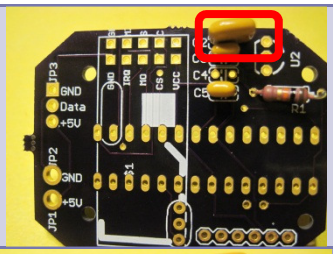
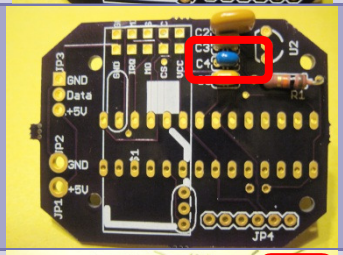
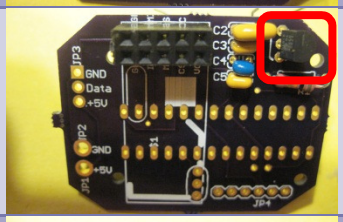
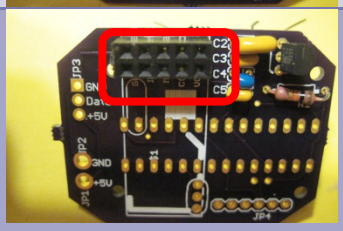
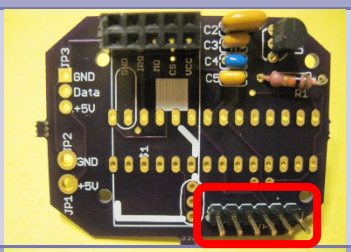
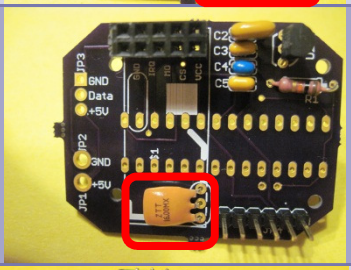
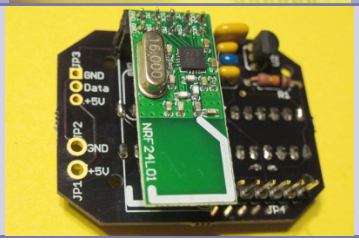




7. Assemble the board

Assembly is very straight forward thru-hole soldering. One tricky part is that parts actually will go on both sides of the board so pay attention to the instructions. The silkscreen on the board will also indicate where and which side of the PCB a part will go. The following is the recommended order to install parts for easiest soldering. You can do it anyway you please but I do recommend starting on the BACK side first (trust me, I did it the other way around taking these pictures and it made soldering the 28 pin socket harder!) Check boxes are provided so you can check off steps as you complete them.

Step#	Instructions	Picture
0 ☐	***On the BACK of the board:*** **<u>If you are using the recommended 8 pin variation of the NRF module</u>** (see NRF Transceiver Types) you should solder bridge across the option indicated before installing the U1 socket as it gets more difficult (but not impossible) to do after. Apply a liberal amount of solder such that the two pads are shorted. This supplies GND to pin 2 of the NRF socket and if you plug in a 10 pin module, it will cause a short as pin 2 on the 10 pin is Vcc.	
1 ☐	***On the BACK of the board:*** <u>Install C1 0.1uf cap</u> Usually coded with "104" to indicate 100000pf = 100nf = 0.1uf Install either direction – there is no polarity	
2 ☐	***On the BACK of the board:*** <u>Install U1 18 pin socket.</u> *(Do not install if you wish to reuse the original GECE case!)* If you are socketing the CPU (recommended but optional), the notch for pin one on the socket should follow the one on the silkscreen. *Note: Sockets will work backwards but you need to follow the notch for pin 1 orientation on the silk screen when installing the chip!*	
3 ☐	***Now flip to the FRONT side of the board:*** <u>Install R1 10K ohm resistor</u> Color code = Brown Black Orange + Gold or Silver typically Precision resistors will be Brown Black Black Red + the band that indicates the resistor precision. Install either direction – resistors have no orientation.	
4 ☐	<u>Install C3 and C5 .1uf caps</u> Install either direction – these types of caps have no orientation.	

5 ☐	Install C2 1.0 uf cap Usually coded with a 105 Install either direction – these types of caps have no orientation.	
6 ☐	Install C4 4.7 uf cap Usually coded with a 475 Install either direction – these types of caps have no orientation.	
7 ☐	Install the 3.3v Regulator. The regulator should be installed oreiented as shown on the silkscreen. Flat side goes toward the center of the board.	
8 ☐	Install the 8 or 10 pin female header connector for the wireless module. *(Do not install if you wish to reuse the original GECE case!)* Optional but recommended. You might opt to solder the wireless module pins directly into the board to reduce height but do that last.	
9 ☐	Install the 6 pin header to be able to attach the serial cable for programming.	
10 ☐	Install the 16MHz Resonator Install either direction – these types of resonators have no particular orientation. Note: In order to miss the wireless module, it should be soldered so it can lay flat against the board as shown.	
11 ☐	Install the CPU, wireless module, power connections and lights and test.	

8. Loading the Sketch

At this point, your board is built, the power is tested, the Atmega is onboard and you are now ready to program the sketch.

If you have been following along with this guide, it is highly likely that you have the sketch code loaded and appropriately modified (as needed).

The process to get the sketch on the board is very simple. The overview of the process follows and detailed in following sections:

- Use an "appropriate" USB to serial TTL adapter and drivers
- Install and run the Arduino IDE
- Configure the serial port of your device in the IDE. Note: varies from device to device
- Load the sketch
- Program

8.1. Selecting a USB to serial TTL adapter

This step is the most bewildering and problem fraught part of the whole Arduino scheme. There are so many options available on the market with so many different capabilities, different pin-outs, and so many unscrupulous manufacturers or inaccurate descriptions that you can end up with something that won't work or requires special handling to get around the difficulties.

8.1.1. Adapter Requirements

What you need is a 6 pin USB to serial adapter that works with the O/S you plan to use to run the Arduino IDE, with the right outputs for an Arduino interface. You must have outputs that include:

- Power (3.3v is best for all Arduino situations due to the NRF 3.3V requirements, but designs that include a 3.3v regulator for the NRF will work from 5V. If you are unsure, just unplug the NRF while updating the sketch.)
- GND
- TX
- RX
- CTS/RTS or DTR (data terminal ready) or RST (Reset?) Note: CTS is not a requirement but is common on some designs along with DTR. This signal is needed to reset the Atmega to initiate the sketch programming via the bootloader.)

The pin-in of the RFColor2_4 is as follows:

Pin #	Description	Notes
1	RTS / DTR / RST	
2	RX in	
3	TX out	
4	PWR	(+5 or +3.3v)
5	CTS	(not used – connected to GND)
6	GND	

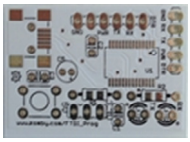
If you examine the descriptions of an adapter, it is best try to try to get as close to this pin out as possible or you will need to make an interconnect cable (or rewire the included one) to allow the board to plug in. DTR is the key to being able to auto-program and it may also be labeled "RTS" or "RST".

8.1.2. Adapter Issues

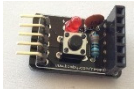
Some of the issues the author has experienced with boards:

1. Board descriptions that sound right, but actual details show they do not actually have the control line to do the reset. You might think a 5 pin device sounds right but if the outputs are +5, +3.3, GND, RX, TX you can see you are missing the DTR/RTS/RST signal to initiate the bootloader.
2. Some board descriptions say 6 pin but looking at the pictures shows only 5 pins. BEWARE.
3. Right signals, but will not work with the O/S. This is typically a driver issue and you should get a driver update from the converter manufacturer or supplier.
4. You have everything hooked up, latest drivers loaded for you OS but the Arduino IDE cannot upload the sketch. This usually indicates that you have a board with a Chinese counterfeit USB to serial chip and the driver from the legitimate chip manufacturer recognizes that the part is "not right". This is more common than you might think! The author has had to send a board to one of the bigger USB Serial chip manufacturers because of this issue.
5. You have the right drivers, the right chips, the right outputs but you cannot initiate the bootloader to do the sketch upload. Sometimes the reset signal out of the adapter needs to be appropriately "conditioned" for some board designs. DTR (RTS / RST) needs to go through a .1uf cap to accomplish reset. Fortunately the RFColor2_4 has this cap on board so this step should not be required. For other boards, in order to be able to plug directly into an Arduino and be able to control the auto-program feature, you likely will need to make a modification either in your cable interconnect or on the board to add an in-line .1uf cap on the DTR (or RST) signal. It is possible that some boards that have an actual Arduino RST output instead of DTR or RTS and already have the in-line cap – you will need to check on your board. A Komby Reset Breakout is another way to solve this issue but is not required for the RFColor2_4 board.
6. Serial port numbers are different on different boards. See the following section on configuring the IDE for the serial port number.
7. Did not put in the serial adapter BEFORE running the IDE. In this case, you usually need to quit the IDE, install the adapter so the COM port is available and then restart the IDE to be able to see the needed COM port. Sometimes you get lucky and just existing the serial selector and going back in gets the new adapter listed in the Arduino IDE serial pick list.

8.1.3. Adapter Recommendations:



The Komby FTDI Serial Programmer board was designed specifically to handle uploading sketches to the Arduino with the 6 pin sketch port on these types of boards. As a bonus, it also has a 5 pin output for other related hardware like the Komby wireless designs. It can be obtained here:
http://www.komby.com/ftdi_prog



At the same time, it might be a good idea to get the Komby Reset Breakout board. This allows non-Komby adapters to have the DTR. RTS or RST signal conditioned to properly handle the reset on boards that need it. It can be obtained here:
http://www.komby.com/reset

The author has had good luck (i.e. works with Windows XP, 7 and 8) with a couple of other adapters from EBay / Alibaba / other sources:

"6 pin FTDI FT232RL USB to serial"

These are a bit more expensive than some of the other adapters, but then you get some data cable to play with rather than needing to supply a different way to get

the signals to the board. As a bonus, these seem to use genuine FTDI parts thus work with the new drivers from FTDI including support for Win 8 and 8.1, etc. You may need to change around the wire order a bit to match the Komby 6 wire pin out. The author uses this one for all sketch updating.

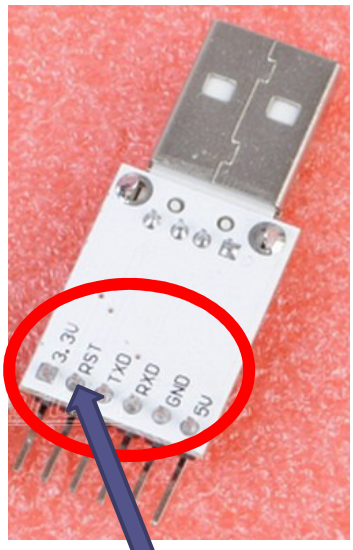
"**USB 2.0 to TTL UART 6PIN Module Serial Converter CP2102**"

This one does not use the FTDI chip, but rather the Silicon Labs part and seems to work well in all variations of Windows. The author has the first one pictured wired as a dedicated interface to a serial transmitter. It can be used to program sketches but is a little less convenient than the FTDI cable version. The one in the second picture has not been tested yet – it claims to have a RST signal and 3.3v out instead of RTS/CTS or DTR as the first one has.

Others might work, but again the key will be to have at least ONE flow control output pin that is used to trigger reset to enable the bootloader upload capability.

To reiterate: The key is to find a 6 pin output that has one output as described above in the adapter requirements. If you opt to go it yourself, you need to have good information about the boards. Use the pictures if the descriptions do not talk about the pin outs. A customer interconnect cable MAY be required.

Example:



Note: output that enables auto-program, has all other needed signals. One on left has a bonus 3.3v so the NRF does not get wrong voltage if installed on the board during sketch uploads. Is "RST" actually supposed to be RTS (request to send) or is it really shorthand for "reset"- hmmmm??

Note: watch for incorrect descriptions!! There are MANY offerings that say "6 pin" but the picture shows a 5 pin. Which is right- the description or the picture. You should stay away from those just to be sure you get the right adapter.

8.2. Install the Arduino IDE

Installing the Arduino IDE is easy by getting the code from the Arduino site:

http://arduino.cc/en/Main/Software

You want the one that supports at least the "Arduino Uno" as that is what the board emulates.

The instructions on USING the IDE can be found here:

http://arduino.cc/en/Guide/HomePage

The instructions on downloading and installing the code for the RFColor2_4 is detailed in the following "Code Compiling" section. However, the Komby code ALSO works for this board and has all the modifications already done. Details can be found here:

http://learn.komby.com/wiki/21/getting-started-guide

8.3. Configure the serial port

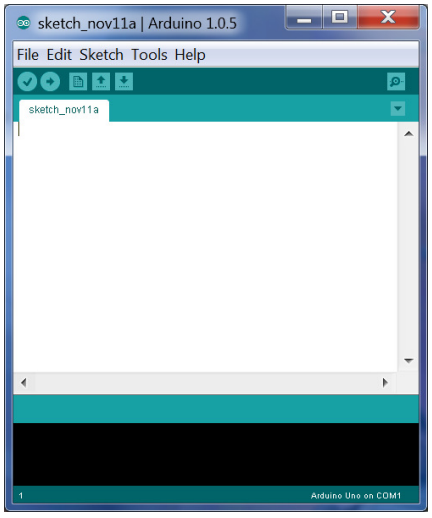
In order for the Arduino IDE to use the serial TTL adapter, you need to tell it what the port number of the device. This assumes your adapter has installed and you have the appropriate driver installed for it. If you plug it in and you do not get error messages, there is a good chance it is working.

8.3.1. Determine the COM Port

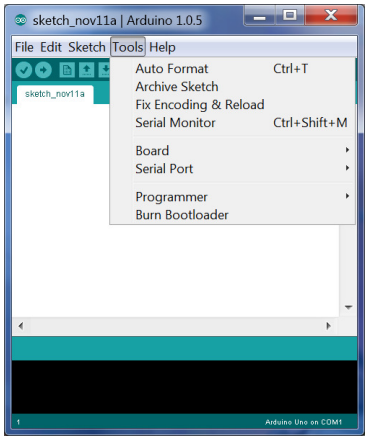
You can use the control panel Device Manager to determine which adapter has what serial port. With the adapter unplugged, open the "Device Manager" found in the Control Panel/System/Device Manager (or Hardware tab/Device Manager in XP) and then open up "Ports (COM & LPT). Note the existing com ports listed there with the adapter unplugged, then plug it in and see which port number is added. Note that number (and label that board with that number also). Usually the port number stays consistent to a specific board on a specific system. It may be a different port number on another computer – it just depends on what was installed before the port you are installing now.

8.3.2. Configure the IDE port

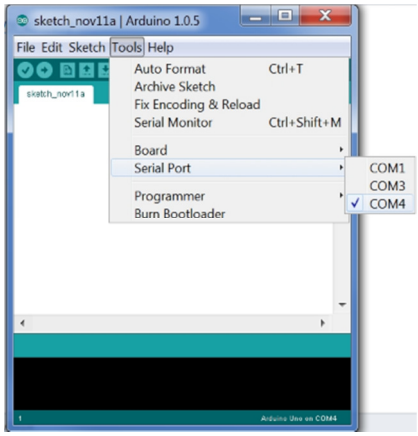
Now start up the IDE. Pictured here is the latest ver. 1.0.5 (at time of writing this doc) on Windows



Next select Tools on the toolbar:



Then select the com port you determined is the one you want:



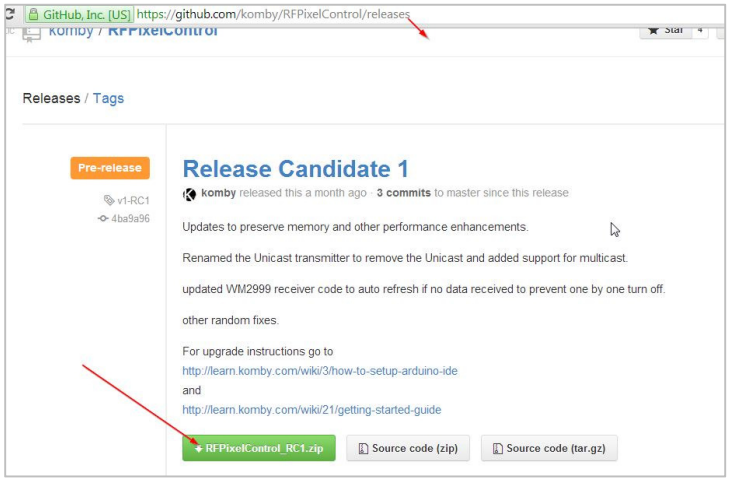
Now when you load the appropriate sketch, and select "upload", the Arduino IDE will access the TTL Serial Adapter to upload to though the bootloader on the Atmega.

8.4. Loading the Sketch

Please refer to the section on "Compiling the Code" unless you want to do this the easier way as follows:

8.4.1. Komby code for the RFColor2_4 Receiver

Alternatively, you can load the Komby Github code and use the GECEReceiver sketch making sure to set the target type to RFColor2_4.



When using the Komby RFPixelControl code you will not need to download your own copies of the GEColorEffects or digitalWriteFast libraries. They are included in the Github repository download (the big green button zip on the RFPixelControl releases page).

To get the most current copy of the release go to: https://github.com/komby/RFPixelControl/releases

The following are the configuration options you need to know about to tune your receiver:

GECEReceiver.ino

```
/*********************  BEGIN CONFIGURATION SECTION ************************************/
 // Define a Unique receiver ID.  This id should be unique for each receiver in your setup.
// If you are not using Over The air Configuration you do not need to change this setting.
// Valid Values: 1-255
#define RECEIVER_UNIQUE_ID 33

//What board are you using to connect your nRF24L01+?
//Valid Values: RFCOLOR2_4, MINIMALIST_SHIELD, RF1_1_2, RF1_1_3, RF1_0_2, RF1_12V_0_1,
KOMBYONE_DUE, WM_2999_NRF
#define NRF_TYPE                          RFCOLOR2_4
//What Pin on your board is the data connected to
```

```
// Pin 2 is used on the RF1 4 pin header.
#define PIXEL_DATA_PIN                          2

//What Speed is your transmitter using?
//Valid Values   RF24_250KBPS, RF24_1MBPS
#define DATA_RATE RF24_250KBPS


// Set OVER_THE_AIR_CONFIG_ENABLEG to 1 if you are making a configuration node to re-program
// your RF1s in the field.  This will cause the RF1s to search for a
// configuration broadcast for a short period after power-on before attempting to
// read EEPROM for the last known working configuration.
#define OVER_THE_AIR_CONFIG_ENABLE 0

// If you're not using Over-The-Air configuration these variables are required:
#define HARDCODED_START_CHANNEL 1
#define HARDCODED_NUM_CHANNELS 150

//What RF Channel do you want to listen on?
//Valid Values: 1-124
#define LISTEN_CHANNEL 100        // the channel for the RF Radio
```

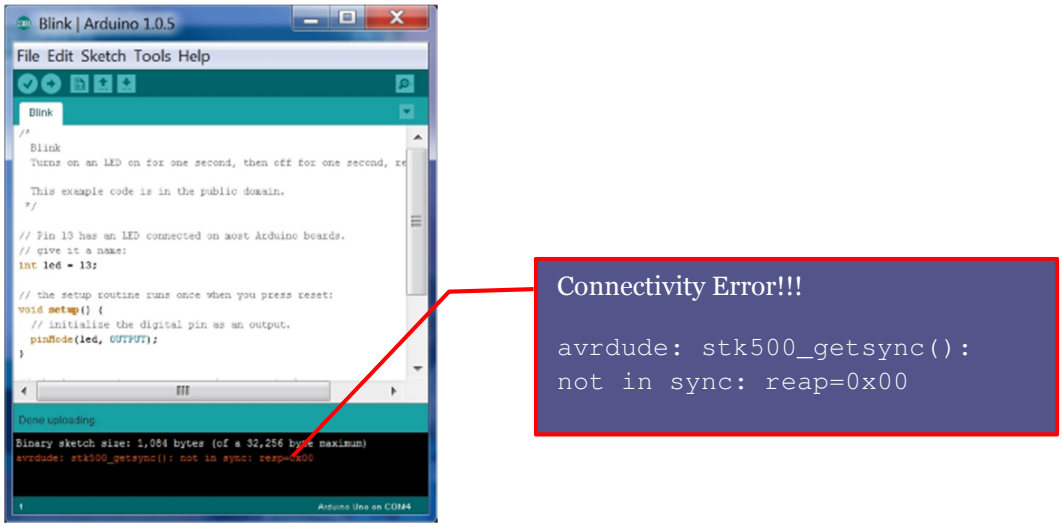
8.4.2. Komby Code for RFColor2_4 Serial Transmitter

To use the RFColor2_4 board as a transmitter connected to the PC via the USB serial adapter used to program the sketches, you need to do a little additional tweaking. The Komby site details what is needed to do. Screenshots of how to do the hardware serial hack are available for both mac and PC:

http://learn.komby.com/wiki/20/hardware-serial-modification

8.5. Program

Click Upload once your code tweaks are done and it should upload. Successful upload will result in no error messages so Watch for errors. Connectivity errors (like serial driver issues, or wrong serial port selected) get an error during upload similar to:



9. Code: Compile instructions for non-Komby code

The code for RF_Color_24 has originally developed in the Arduino 1.0 environment by Joej85. It is unknown if this code will work in other versions, so it's best to use 1.0 to avoid unknown problems.

Note: If you are using the KOMBY code as described before, you do not do any of these manual steps. This information is provided to show the original work that was done by joej85 that is now comprehended by the Komby code releases.

To compile the RF_Color_24 code you need to have the RF24 library for the NRF24L01+ module:
http://maniacbug.github.com/RF24/index.html

You also need the GEColorEffects library:
http://www.digitalmisery.com/wp-content/uploads/2011/11/GEColorEffects-Arduino-1.0.zip

An alternative is here, but this file is older and needs additions detailed below to compile:
http://www.digitalmisery.com/wp-content/uploads/2011/11/GEColorEffects.zip

You also need the digitalWriteFast library:
http://code.google.com/p/digitalwritefast/

9.1. Modifications:

In order to compile and load the controller/receiver code for the RF_Color_24, the RF24.cpp and RF24.h files from the RF24 library must be modified. You may be able to get already modified code from places like www.komby.com, or the wiki/file repositories on the DIY sites.

Note: If you are using the KOMBY code as described before, you do not do any of these manual steps. This information is provided to show the original work that was done by joej85 that is now part of the Komby code releases.

9.1.1. RF24.cpp

In order to minimize the time it takes to load a packet to the TX FIFO buffer, the SPI bus clock rate was increased to 8MHz. The default is 4MHz. To do this:

Find: SPI.setClockDivider(SPI_CLOCK_DIV4);

Change To: SPI.setClockDivider(SPI_CLOCK_DIV2);

9.2. RF24.h

There are several low level functions that are hidden in the RF24 library. Some of the low-level functions in the code need to be able to use the NRF24L01+ module in exactly the way the author wanted. In order to un-hide these low level functions do this:

Find: protected:

Change To: public:

9.3. Hardwareserial.cpp

In order for the serial DMX interface to work, the hardwareserial.cpp Arduino file must be modified. The serial interrupt must be commented out as shown below. The hardwareserial.cpp file is located under the Arduino 1.0 base directory in the hardware\arduino\cores\arduino subdirectory. Note: It is important to note that making this change to hardwareserial.cpp, you will likely be breaking any other application that depends on serial communication. This change was problematic when trying to load bootloaders on bare Atmega328 chips. Don't forget to change it back or keep a second copy appropriately labeled if you work on other projects that should not use these changes.

Here are the changes:

```
// Begin comment here
//
//#if !defined(USART0_RX_vect) && defined(USART1_RX_vect)
//// do nothing - on the 32u4 the first USART is USART1
//#else
//#if !defined(USART_RX_vect) && !defined(SIG_USART0_RECV) && \
//    !defined(SIG_UART0_RECV) && !defined(USART0_RX_vect) && \
// !defined(SIG_UART_RECV)
//  #error "Don't know what the Data Received vector is called for the first UART"
//#else
//  void serialEvent() __attribute__((weak));
//  void serialEvent() {}
//  #define serialEvent_implemented
//#if defined(USART_RX_vect)
//  SIGNAL(USART_RX_vect)
//#elif defined(SIG_USART0_RECV)
//  SIGNAL(SIG_USART0_RECV)
//#elif defined(SIG_UART0_RECV)
//  SIGNAL(SIG_UART0_RECV)
//#elif defined(USART0_RX_vect)
//  SIGNAL(USART0_RX_vect)
//#elif defined(SIG_UART_RECV)
//  SIGNAL(SIG_UART_RECV)
//#endif
//  {
//  #if defined(UDR0)
//    unsigned char c  =  UDR0;
//  #elif defined(UDR)
//    unsigned char c  =  UDR;
//  #else
//    #error UDR not defined
//  #endif
//    store_char(c, &rx_buffer);
//  }
//#endif
//#endif
// end comment here
```

9.4. GEColorEffects.cpp

These functions in the GEColorEffects.cpp file were modified to use the digitalwritefast method. All timing values were verified using an oscilloscope and work well.

Note: If you are using the KOMBY code as described before, you do not do any of these manual steps. This information is provided to show the original work that was done by joej85 that is now part of the Komby code releases.

Search the GEColorEffects.cpp file for these functions and modify them as shown below:

```
void GEColorEffects::begin() {
   noInterrupts();
   digitalWriteFast(19, HIGH);
   delayMicroseconds(10); // Output should be ~ 10uS long
   digitalWriteFast(19, LOW);
   interrupts();
 }
//
void GEColorEffects::one() {
   noInterrupts();
   digitalWriteFast(19, LOW);
   delayMicroseconds(19);  // Output should be ~ 20uS long
   digitalWriteFast(19, HIGH);
   delayMicroseconds(10);   // Output should be ~ 10uS long
   digitalWriteFast(19, LOW);
   interrupts();
 }
//
void GEColorEffects::zero() {
   noInterrupts();
   digitalWriteFast(19, LOW);
   delayMicroseconds(9);  // Output should be ~ 10uS long
   digitalWriteFast(19, HIGH);
   delayMicroseconds(21);  // Output should be ~ 20uS long
   digitalWriteFast(19, LOW);
   interrupts();
 }
 //
 void GEColorEffects::end() {
   noInterrupts();
   digitalWriteFast(19, LOW);
   delayMicroseconds(30);  // Quiet time should be ~ 30us long
   interrupts();
 }
```

9.5. Other compile notes

- If you are using Arduino IDE 1.0.1 or later, you may also need to change the included header file from "wprogram.h" to "arduino.h" in the gecoloreffects and digitalwritefast libraries.
- To be able to compile, you may need to add:

  #include "digitalWriteFast.h"

  into the GEColorEffects.cpp file.