Styx Networked LED Controller

STYX
LED CONTROLLER

Introduction:

The Styx Networked LED controller allows for 8-bit resolution control of up to four LED color channels (Red, Green, Blue, White). The controller provides two input modalities, allowing it to be integrated into projects using either an ethernet interface or an i^2c interface. The controller will accept 12V or 24V allowing it to control a variety of LED devices across application domains. This controller is perfect from any project seaking to quickly add connectivity to LED lighting solutions. Example software ships with the controller that demonstrates basic interfacing capabilities through the ethernet and i2c interface.

General Controller Setup:

As previously mentioned, the LED controller can take 12V or 24V DC as an input. Make sure to match the supply of the controller to the supply that your LED load is expecting (i.e. if your LED strip takes 24V make sure to size the controller supply accordingly). From here, the setup is rather simple, and the procedure is outlined in the steps below. Note that figure one in this section contains a picture of the controller with these connection points highlighted.

1. Connect the ground lead of your power supply to the GND input of the controller power input screw terminal, and connect the 12V/24V lead of your power supply to the Vin input of the controller power input screw terminal.
2. Connect the respective LED strip/panel color channels to the controller color channel output. Note that the silkscreen next to the controller channel output screw terminals indicate which specific terminal connection corresponds to red, green, blue and white.
3. Connect the +12V/+24V line of your LED strip/panel to the connector on the two-pin color channel output screw terminal with the “+” silkscreen next to it.
4. Turn on the LED controller and make sure that the 5V and 3.3V rail status LEDs on the board turn on.

Instructions from this point forward are application specific. If you are using the ethernet connection modality, please proceed to the “Ethernet Communication Mode” section, and if you are using the i^{2}c connection modality, please proceed to the “I^{2}C Connection Mode” section.

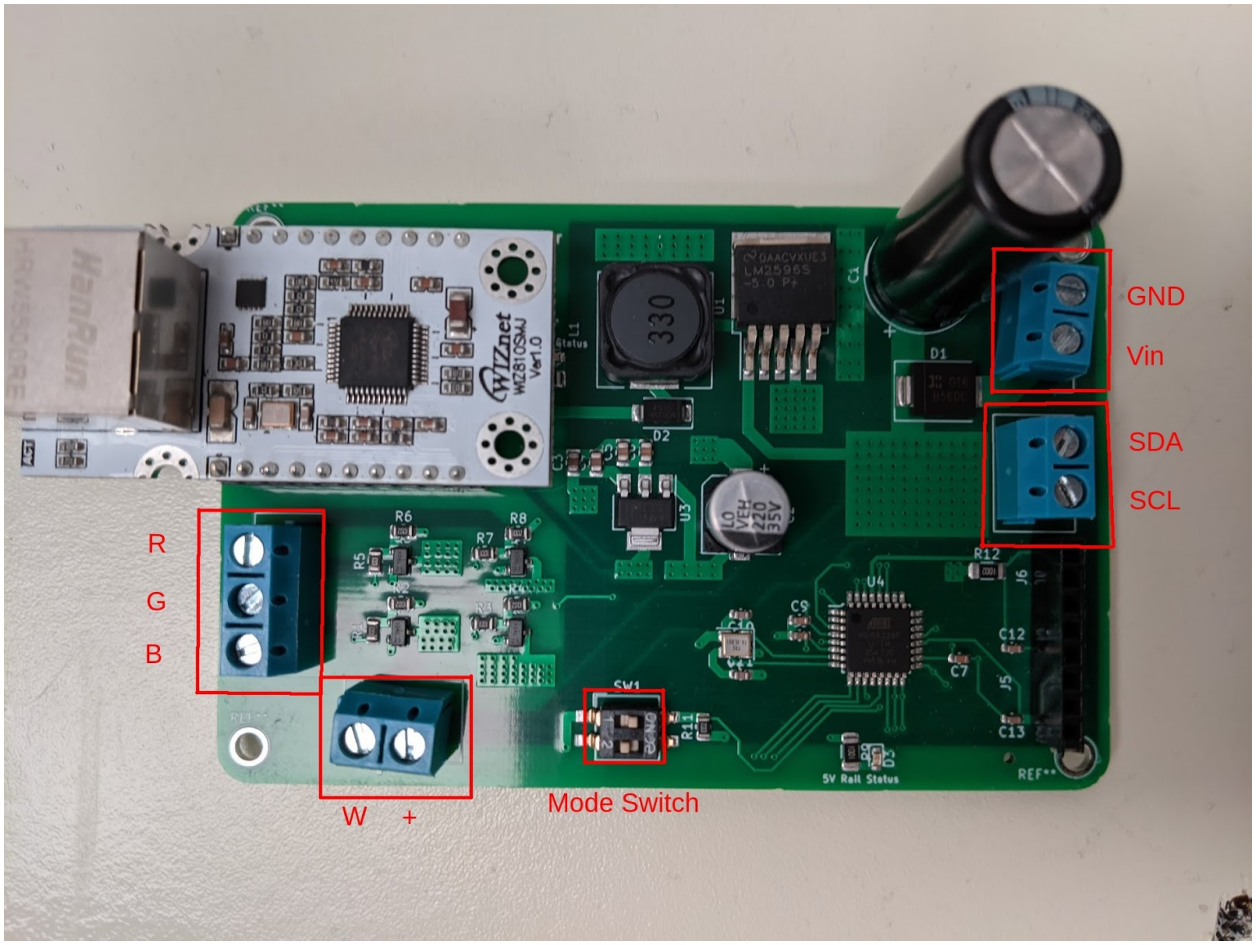

Figure 1: LED Controller General Connections

Ethernet Communication Mode

The first step in setting up ethernet communication is to plug an ethernet cable into the ethernet controller receptacle. The controller is currently set to obtain an IP address from DHCP, so make sure the network has a functional DHCP server to allow for this to work correctly. Before turning on the controller, make sure to set the switches on the mode switch to the configuration seen in figure 2. Once the board powers on, it should obtain an IP address via DHCP, which you should be able to find using your network manager tool of choice. Once you have the IP address, you can use the provided color picker website on your local network to change the color of the LEDs. Open the provided **eth/web/colorPicker.html** file in a web browser to get started. Note that the

overall RESTful API exposed by the LED controller is documented later in this document.

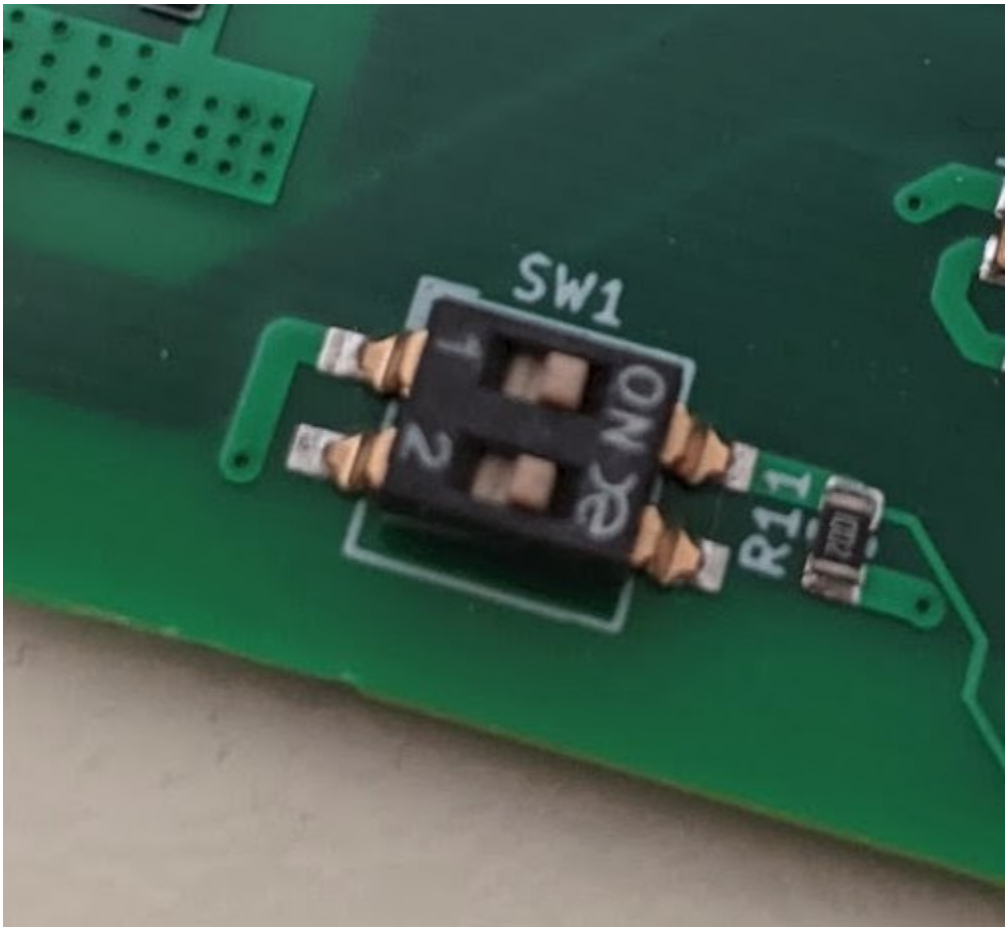

Figure 2: Ethernet Communication Mode Switch Configuration

I²C Communication Mode

For i²c communication, first change the switch to be in the configuration seen in figure 3. Then, connect wires from the SDA and SCL connection on the LED controller to your i²c master device of choice. Note that i²c communication has been tested between this board, an Arduino Uno and a Raspberry Pi 3 B+. In both cases, don't forget to establish a common ground between both the master device of choice and the LED controller. The wiring diagram for controlling the LED controller via i²c from an Arduino Uno can be seen in figure 4, and the diagram for controlling it from a Raspberry Pi 3 B+ can be seen in figure 5.

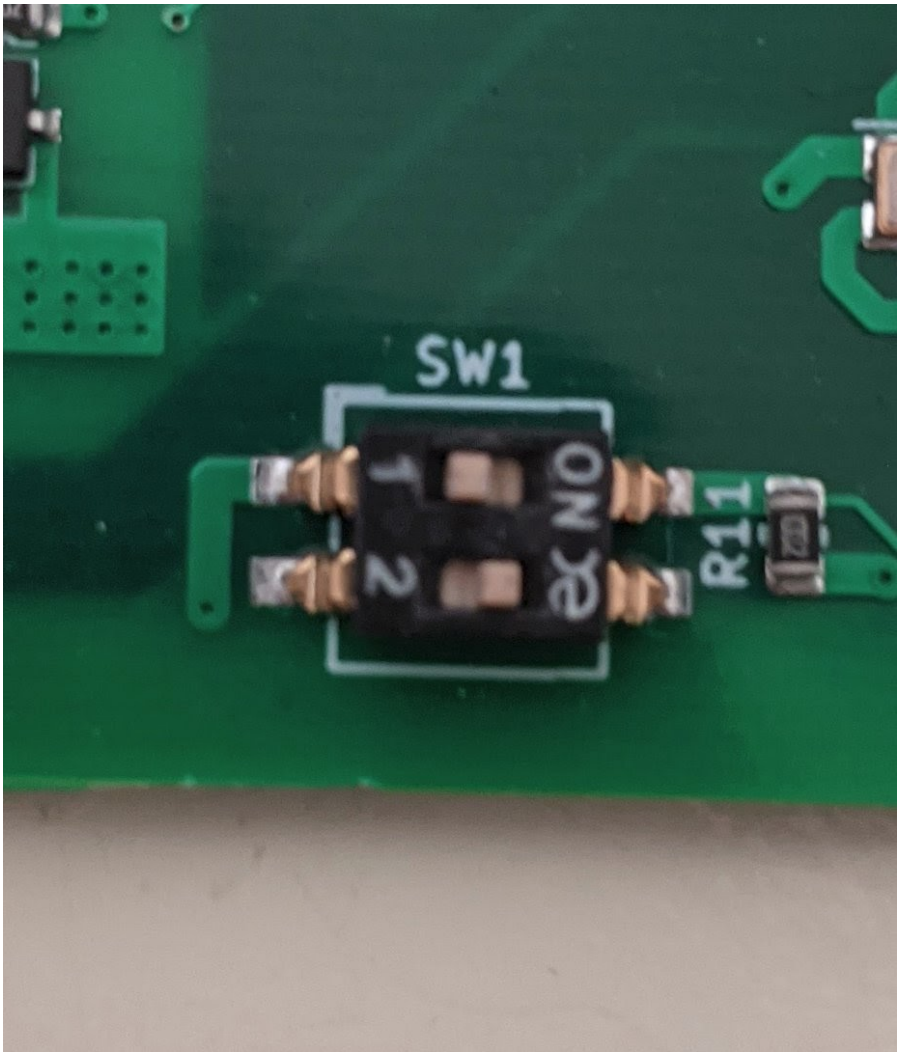

Figure 3: I²C Communication Mode Switch Configuration

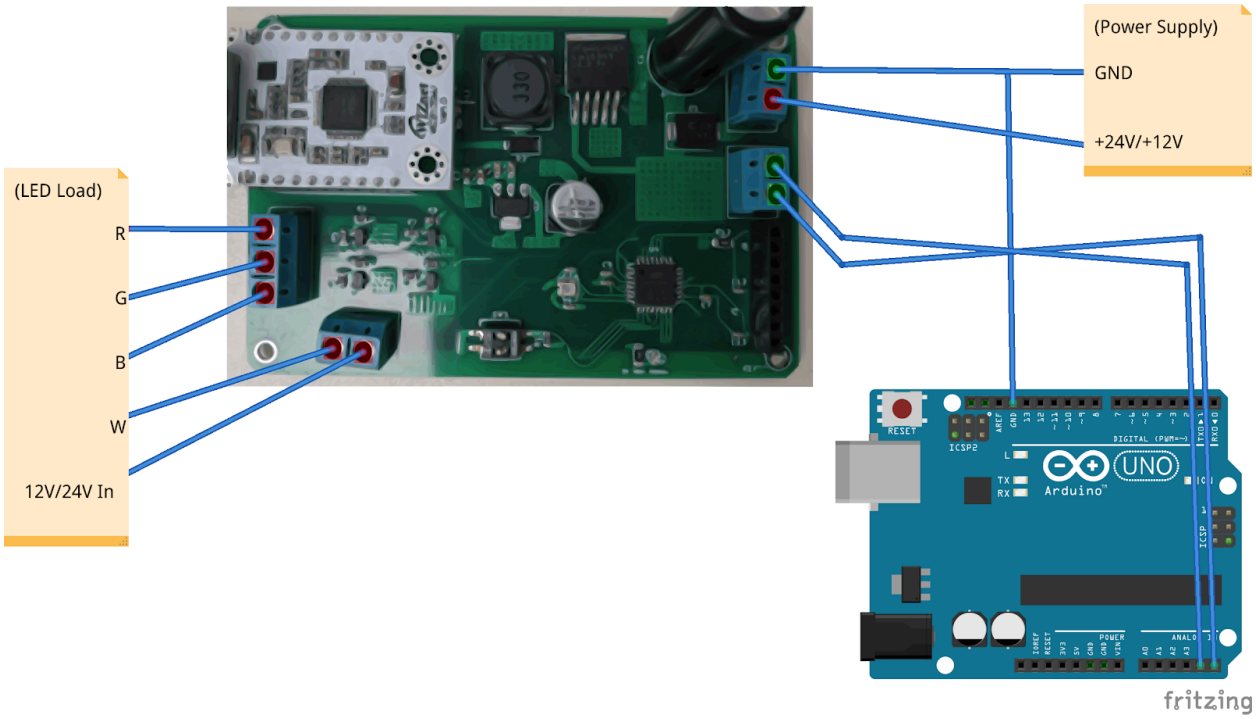

Figure 4: Arduino Uno I²C Master Wiring Diagram

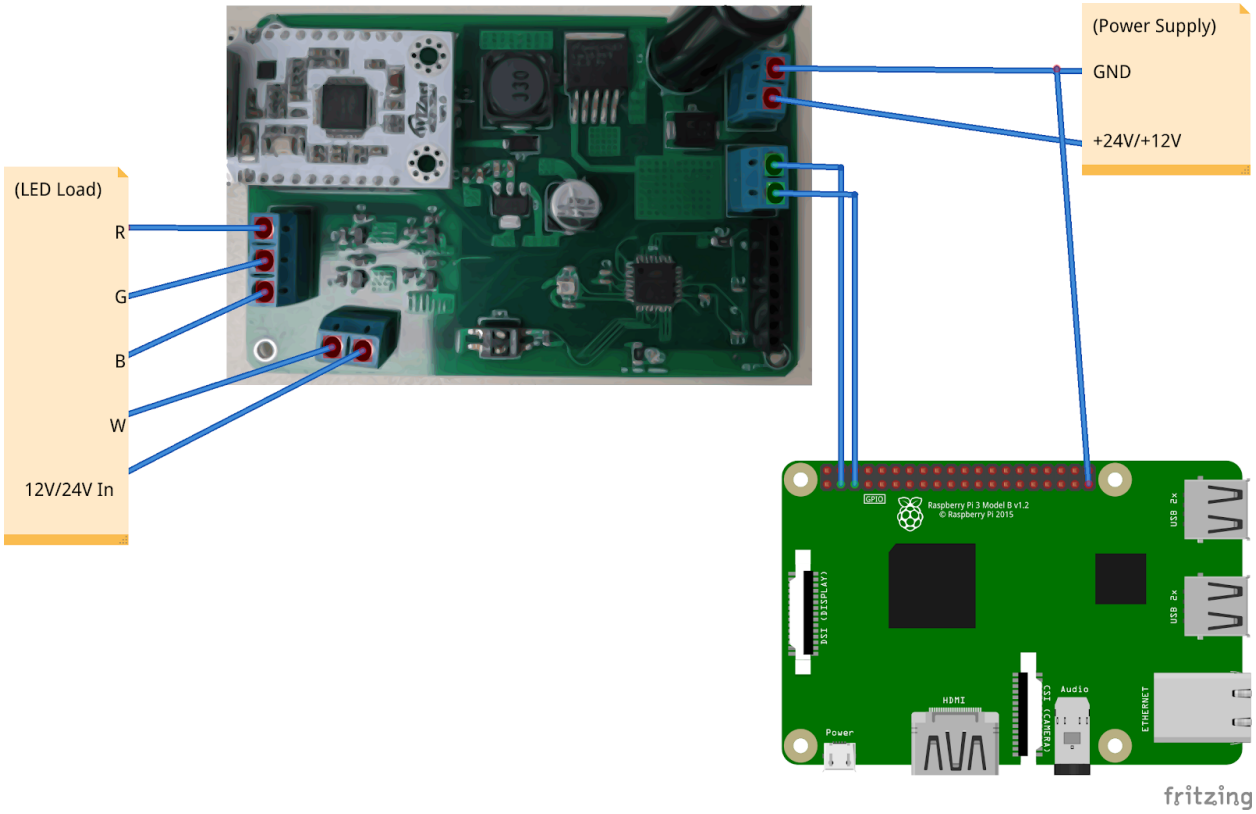

Figure 5: Raspberry Pi 3B+ I²C Master Wiring Diagram

Example software was developed for both cases. The sketch for the Arduino Uno to control the LED controller via i^2c can be found in the **i2c/arduino/arduino_i2c_interfacing_demo** directory. A python script was developed to communicate with the LED controller via i^2c and can be found in the **i2c/RPi** directory. For additional configuration information for setting up i^2c on the Raspberry Pi 3 B+, please refer to the guide here: https://dronebotworkshop.com/i2c-arduino-raspberry-pi/

Ethernet Communication Mode: RESTful API

When the LED controller is in ethernet communication mode, it exposes a RESTful API that can be used to change the state of the controller. These requests are of the form: **http://<IPAddr>/<CMD>?param=<RRGGBBWW>** where <IPAddr> is the IP address of the LED controller, and <RRGGBBWW> is the concatenation of all respective red, green, blue and white channels. Please change these color values as you see fit for your application. <CMD> instructs the controller on how to control the lights themselves. Available commands are outlined in table 1 in this section.

Command	Description
clear	sets each respective color channel to zero (i.e. turns off all LEDs)
hold	sets each respective color channel to the provided value (i.e. <RRGGBBWW>).
strobe	rapidly flashes each respective color channel on (to the value set from <RRGGBBWW>) and off.

Table 1: API CMD Parameter Descriptions

The LED controller also exposes two additional endpoints (flash and ramp) that take an additional parameter of number of flashes or ramp time. Requests to either of these endpoints can be made in the form **http://<IPAddr>/<CMD>?param=<RRGGBBWW:HH>** Where <IPAddr> is the IP address of the LED controller, and <RRGGBBWW> is the concatenation of all respective red, green, blue and white channels, and HH is either the number of flashes or the total ramp time, depending on the endpoint being used. Please see table 2 in this section for more information on these two endpoints.

Command	Description
ramp	Ramps the light to the provided color value (i.e. <RRGGBBWW>) in HH multiples of 100ms. I.e. a HH value of 10 would result in a one second ramp.
flash	Flashes the light at the provided color value (i.e. <RRGGBBWW>) for HH two second periods. I.e a HH value of 05 would result in five one second on, one second off flash sequences.

Table 2: API Specialized Parameter Descriptions

Please note that the HH value and individual color channel values need to be left padded with zeros. For example, an HH value of 5 should be 05, and a color channel value of ‘A’ should be ‘0A’.

I²C Communication Mode: Data Packet Format

The i²c communication modality expects data in the same information structure as seen in the previous section. That is, command and color string combinations of the form **<CMD><RRGGBBWW>** for clear, hold and strobe, and a form of **<CMD><RRGGBBWW:HH>** for ramp and flash. This data is to be sent to the LED controller acting as an i²c slave device as ASCII encoded byte representation of the respective characters. For an example of how to convert standard string to their respective character arrays in arduino C++ and python, please see the included i2c examples outlined in the previous i²c section. Table 3 in this section outlines some example character arrays and the respective behaviour of the LED controller once it receives said command.

Command Char Array	Expected Controller Behavior
hFF000000	Controller should turn the LED load to a red color and latch the state.
f00FF0000:05	Controller should flash the LED load on and off slowly with a green color for five one second on, one second off flash sequences.
r0000FF00:05	Controller should fade the LED load from off to a bright blue color in 0.5 seconds.
sFFFFFFFF	Controller should rapidly flash the LED load on and off at a full white color.
c00000000	Turns the LED load off.

Table 3: I^2C Command Examples and Expected Behavior