NEW MEDIA &
COMMUNICATION
TECHNOLOGY

Raspberry PI

WS2801 LED CONTROLLER

Ilias Ismanalijev
HOWEST | 2NMCT

Contents

Getting Started

Setting up the Raspberry PI

The first step is to prepare the Raspberry PI. I chose Archlinux ARM[1] because of its faster boot times and minimal setup, it does not come with an X11 pre-installed or any other extras you may not need like on the Raspbian OS.

Follow the installation instructions on the page above, plug in an Ethernet cable, a HDMI cable and a keyboard and you're ready to go. Login with the default `root` user and the default `root` password and run `ifconfig` or `ip addr show eth0` to determine your IP address.

Now you can SSH into your PI using that address and the default root user and password. From that point you can interact with your Raspberry PI from the local internet and you can plug out the HDMI cable if you don't need it any longer.

GPIO Interface

To read and send data through the GPIO pins I've decided to use wiringPi. It's a C library, the library feels much like using the Arduino wiring system.

I've only had minimal experience with C in the past but it is a very powerful and lightweight language and focuses on speed.

There are other options. You can simply use a shell script and write data to the files in Linux, for example:

```
$ cat /sys/class/gpio/gpio7/value
```

This will read the value of the GPIO but this is not manageable in a bigger project. There are also Ruby[i], Python[ii] and Java/Scala[iii] frameworks but some of these depend on the C wiringPi library anyway and make use of C bindings.[2]

[1] http://archlinuxarm.org/platforms/armv6/raspberry-pi
[2] More libraries and examples at http://elinux.org/RPi_Low-level_peripherals

Physical	BCM GPIO	WiringPi	Component	---
7	4	7	LED	
11	17	0	Switch	
12	18	1	Pushbutton	
15	22	3	Green	Clock
16	23	4	White	Data

So, I decided on wiringPi.

Download and install wiringPi[3] and run

```
$ gpio -v
$ gpio readall
```

This will tell you if the library and tools installed correctly.

You'll be able to see the GPIO headers and other information about your General Purpose Input / Output. I've included a table with the relevant information of my setup.

[3] Installation and guide for wiringPi http://wiringpi.com/download-and-install/

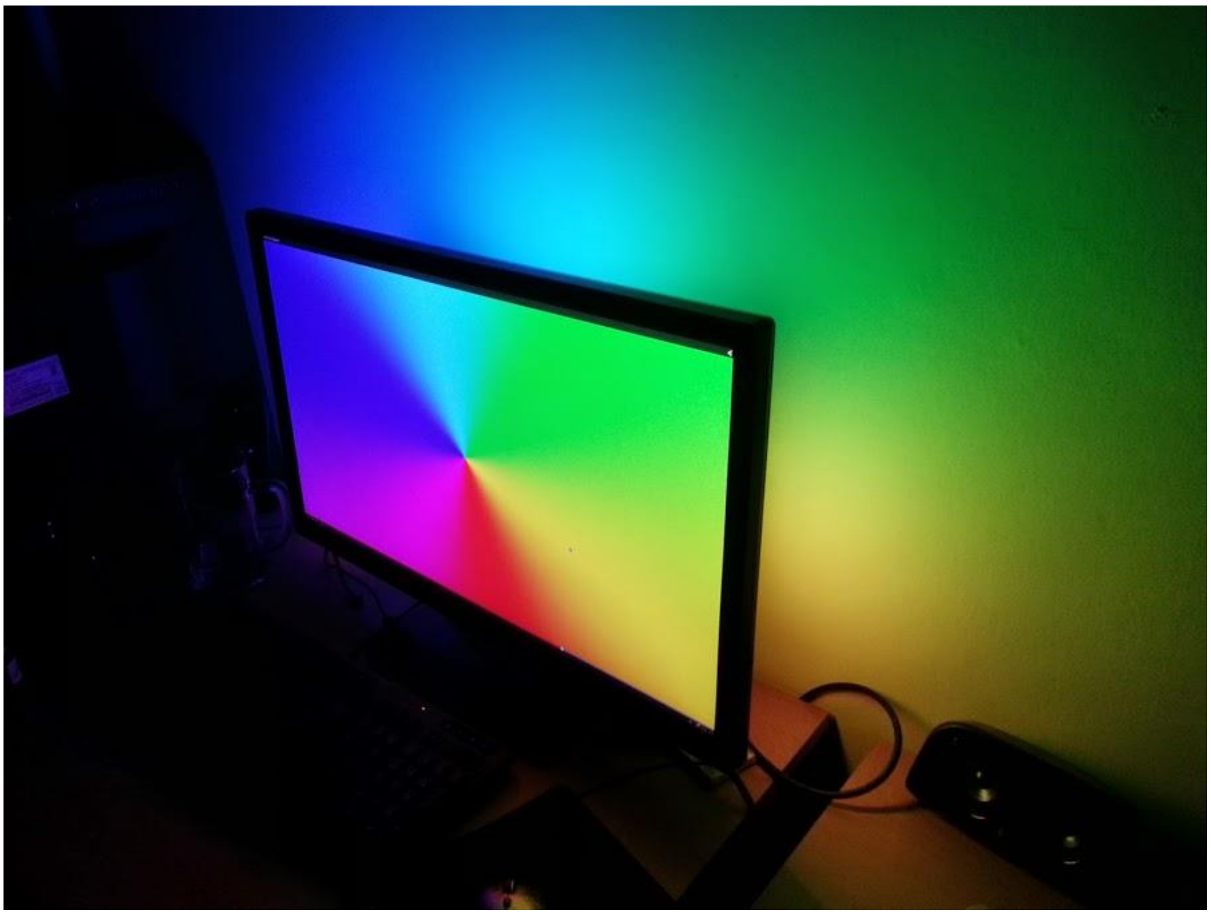
WS2081 LEDs

The Controllable LED

The WS2801 is a programmable LED driver. Every single LED on the strip has one of each. They accept input in the form of a data pin and a clock pin.

There are other LED drivers in the WS28-family like the WS2811 which only has a data pin. It is a bit more difficult to give time-sensitive input with drivers without a clock pin as you'll have to wait at least 50us per data refresh cycle.

There are a ton of great projects that use these LEDs. A fun one I've found recently is setting them up to your television by running Kodi with OpenELEC or Linux with X11. Boblight is an open-source DYI backlight controller that sends data to an Arduino through your USB port.



WS2811 LEDs using boblightd

This is all fun but it requires an advanced understanding of Linux and quite a bit of time to setup the configuration files required to run it correctly. Just like anything with these LEDs.

Specifications

In our specific case, we are using WS2801. We have an external power source connected to it on the red and blue wire and the Raspberry PI connected through the GPIO interface to the green and white wire.

Red	5V
Blue	GND
Green	Clock
White	Data

So in order to set one bit, you have to

1. Set the Clock pin on low power,
2. Set the Data pin on low or high power.
3. Set the Clock pin on high power.

Then you repeat this for

1. Every WS2801 LED on the strip, in this case 10 LEDs
    a. Every LED on the controller, 3 colors.
        i. Red, 8 bits
        ii. Green, 8 bits
        iii. Blue, 8 bits

In total, this comes down to

$$10 \times 3 \times 8 = 240 \; bits$$

Then you wait, and it'll accept the data and push it into onto the LEDs.

Programming

Testing the GPIO

The first thing I did after installation of the library is make a LED blink that's outside of the strip. It was on GPIO pin 4. It blinked, so everything worked.

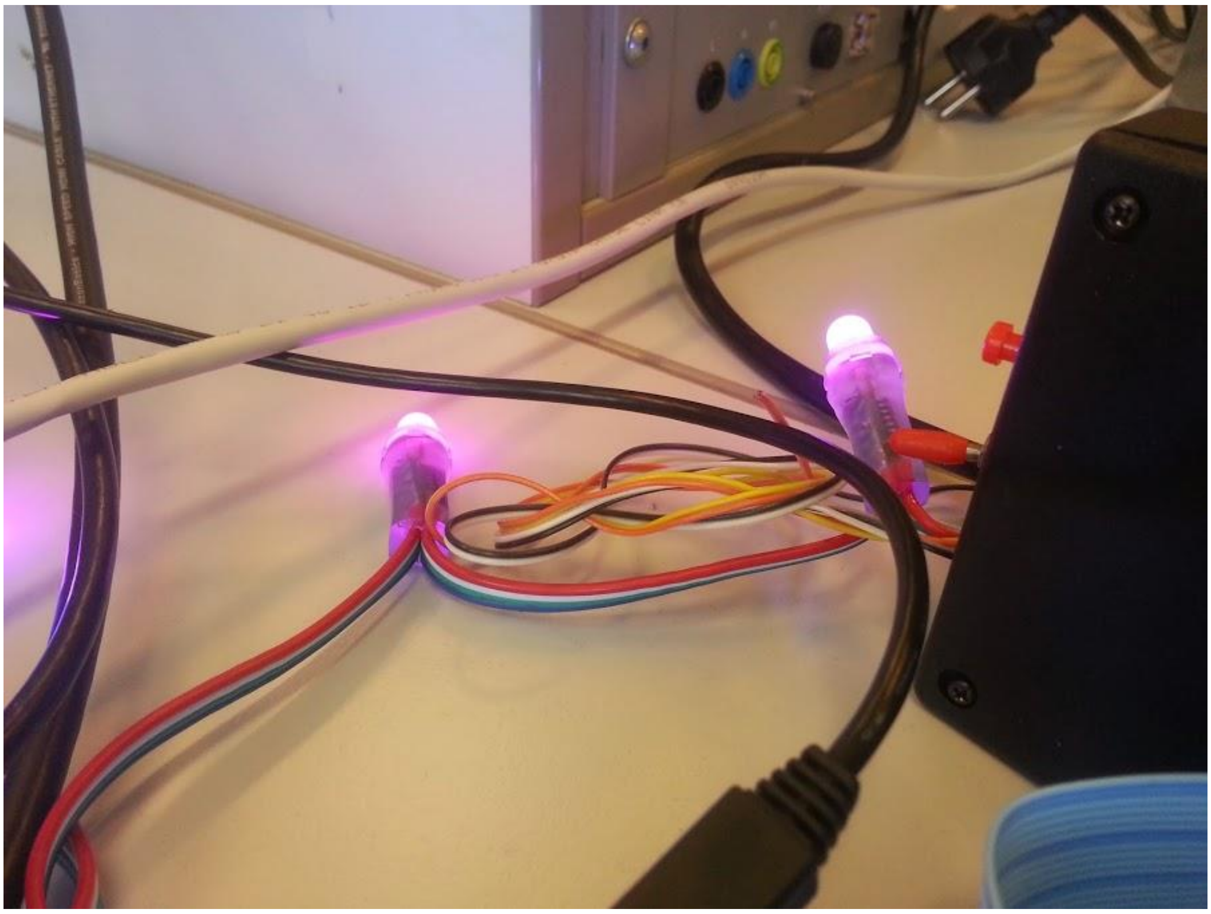
```
#include <wiringPi.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
int pin, i;
scanf("%d", &pin);
printf("pin %d \n", pin);

if (wiringPiSetup() == -1)
    exit (1);

pinMode(pin, OUTPUT);

for (i = 0; i < 10; i++) {
   printf("LED %d ON\n", pin);
   digitalWrite(pin, 1);
   delay(50);
   printf("LED %d OFF\n", pin);
   digitalWrite(pin, 0);
   delay(50);
}

return 0;
}
```

Controlling the LED strip

The next step is to try to get the LED array to work. This is a part I've been stuck on for a while. I simply couldn't get it to work properly. The timing would be off or perhaps I had an incorrect interpretation of the specsheet.

After stripping out the delays and setting the clock to go from LOW to HIGH it worked. So in the end it turned out to be simple.

As a starting point, the LEDs would receive a full white[4] color. Then a random[5] color for each LED.



Making a Socket

To interact with the C code, I decided to use a network socket. This seemed like a good, proven way to interact with the C code. The other option was to run it as an argument but this isn't feasible to do with high-speed animations or interactions back-and-forth.

The socket would accept a binary stream and forward the same bits out to the data pin. So with this code[6] I would be able to keep the C code alive, send and receive data and get rid of the low-level logic of working the clock pin.

[4] https://github.com/Illyism/Raspberry-PI-Led/blob/master/src/white.c
[5] https://github.com/Illyism/Raspberry-PI-Led/blob/master/src/random.c
[6] https://github.com/Illyism/Raspberry-PI-Led/blob/master/src/led.c

Making a Server

To interact with the socket, I decided on adding another server written in JavaScript[7] to make it easier to interact with the LEDs, I wanted to have an easy to control API [8]so I could send data in HEX colours, RGB colors and add in effects such as loops, rainbows, timers and so on.

I used restify to make a REST API for the server. This opens an HTTP port and listens to GET or POST requests and forwards them to the socket or responds with data it can keep in memory.

I made the API feel similar to the Philips Hue API. It is an easy and obvious way to connect to a LED server and it would be easier to allow multiple connections to multiple LED strips in the future.

Making a Client

The final part is to connect to the JavaScript server. You can use anything you want as long as it can send HTTP requests. I made a sample application using JavaScript (again) with some examples in both node and in the browser with jQuery.

You could make a command line interface for this or use any programming language you want similar to the Philips Hue. This is the power of separating the code in so many ways.

Again, the idea is that the microcontroller runs the compiled C code, for speed.
On top of that, it runs a REST API server to make interactions easier.
Then clients connect to the REST API.

[7] I'm using io.js. I've had extensive experience with this programming language and I can write in it very quickly. After wasting a lot of time with the C code I had to finish everything as soon as possible.
[8] https://github.com/Illyism/Raspberry-PI-Led/tree/master/server

Conclusion

It was an interesting project, I've learned about network sockets and how to implement them in C and it gave me some experience in making a REST API. I didn't have enough time and a few more hours to fix the C code to work even faster and to add more examples for timers and special light effects would have been great.

I bought a WS2811 for myself to play with and to mount on my television for a backlight but there is a lot more you can do with LEDs and that is what excites me.

The code for the socket, server and client can be found on GitHub:

il.ly/ws2801

---

[i] https://github.com/jwhitehorn/pi_piper
[ii] http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/
[iii] https://github.com/jkransen/framboos
Code on making sockets, http://www.linuxhowtos.org/C_C++/socket.htm