ESP8266 Artnet to DMX

by **mtongnz** on September 30, 2016

Table of Contents

Intro: ESP8266 Artnet to DMX

The Problem

As a lighting technician, I often find myself having to run cables across walkways, to moving props or simply to places that are awkward to run a wire. Wouldn't it be nice to have a wireless solution? While there are many commercial units available using various protocols, they are either expensive or unreliable.

ESP8266

The ESP8266 is a super cheap but powerful WiFi chip designed to bring IOT to hobbyists. It has a fast processor, more RAM and flash memory than most Arduinos and a (small) selection of GPIOs that support a few different protocols. It is available in a variety of formats and I prefer the ESP8266 07 model as it has an external antenna. The ESP8266 is also compatible with the Arduino IDE: https://github.com/esp8266/Arduino and I use this as it's very familiar and thus easy.

Artnet and DMX

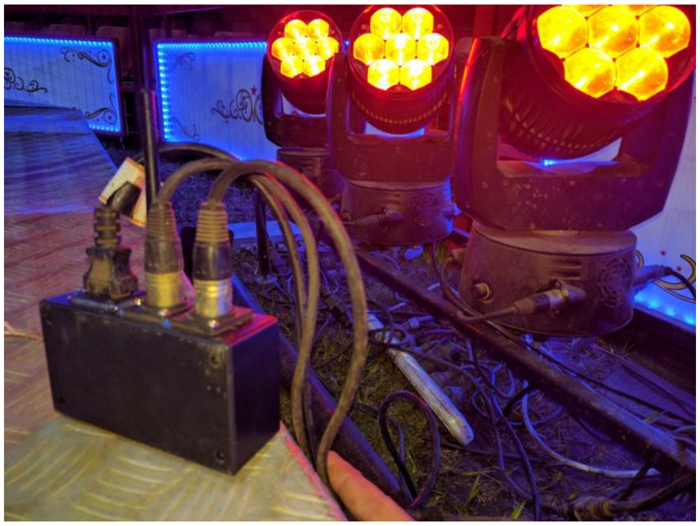
Artnet is a protocol that allows DMX to be sent over a standard IP network. This protocol is supported by pretty much every modern lighting console or software (paid and free). Read more here.

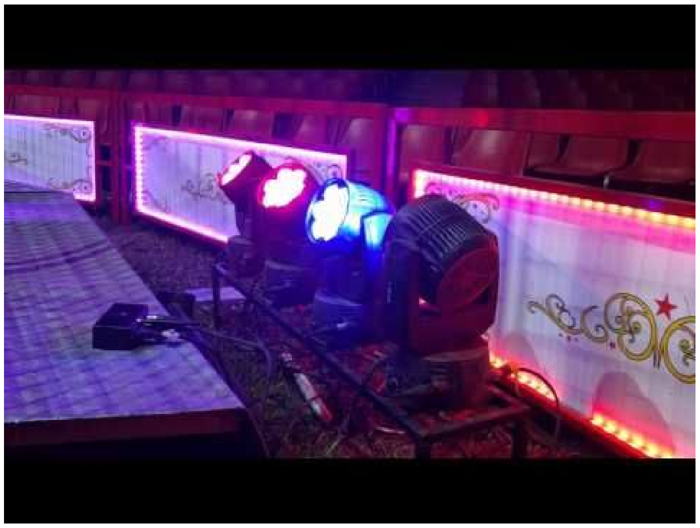
DMX is the industry standard protocol for controlling professional show and DJ lighting - it is based on the RS485 protocol. Each DMX universe can have up to 512 channels and each channel has a value from 0 - 255. Read more here.

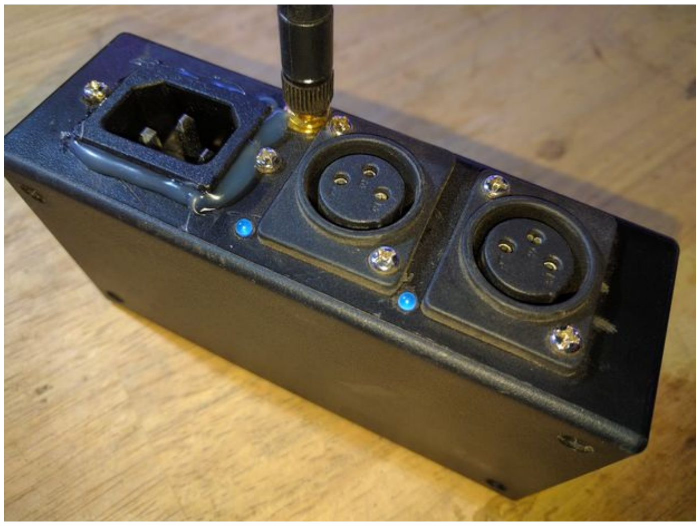
My Solution for Wireless DMX
When I first saw the cheap ESP8266 modules a year ago, I decided that I'd try to build a cheap DIY Artnet to DMX node. This Instructable will walk you through building your own.

Applications

Pro lighting (moving props, trade shows, across walkways....), DJ Lighting, christmas lighting.... Pretty much anywhere that you wish to minimize wiring, this unit will come in handy.







Step 1: â€‹Parts Required

ESP8266 - I used the 07 model but they're all pretty similar

PSU - I used a small USB power brick

Resistors - 2x 1K, 3x 390, 1x 750

Caps - I used 4x 100nF even though the schematic has 5. C1 isn't needed.

3.3V Reg - Can't remember what I used here - any 3.3V regulator will do. The ESP8266 needs 3.3V but doesn't use a huge amount of it.

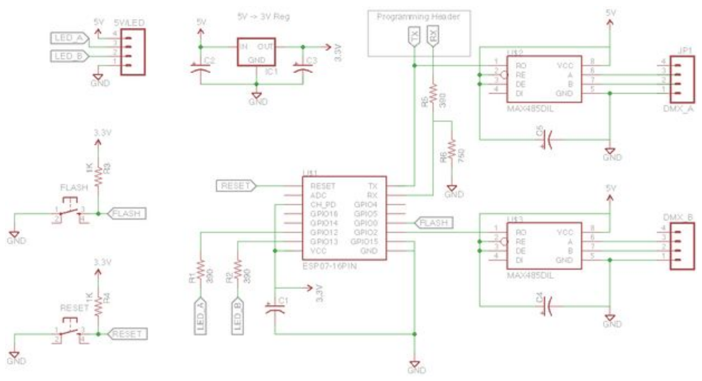
Max485 - These chips produce the DMX output (they can also do input). We need 2 - one for each universe. They are 5V chips but seem to take the ESPs 3.3V logic without any issues. There is a 3.3V version available but I had these on hand and it works...

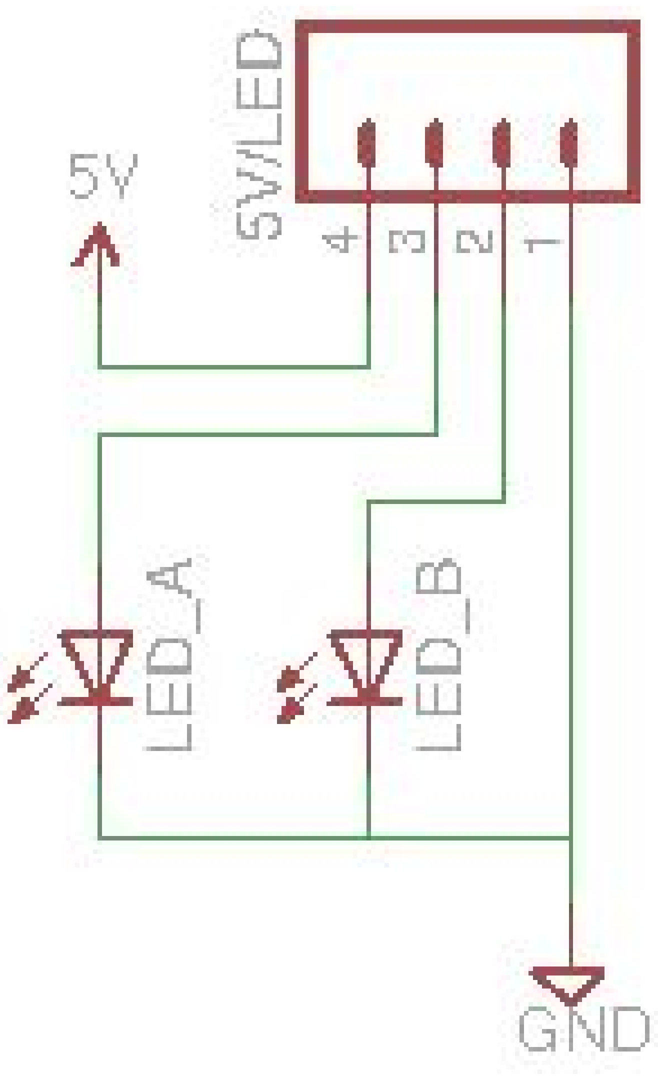
XLR sockets - I used 3 pin but the standard states 5 pin. You could use a single 5 pin for both outputs as Avolites loves to do. I use 3 pin as I have mainly cheap chinese lights with 3 pin, they are stocked more readily and they are a bit sturdier.

LEDs - I have 2 small blue LEDs, one for each DMX universe to show activity. They are optional but recommended.

Power Lead - I used an IEC inlet. They're readily available and allow the lead to be disconnected for easier storage.

Perf board, hookup wire, headers, project box, screws, glue.





Step 2: Schematic and Construction

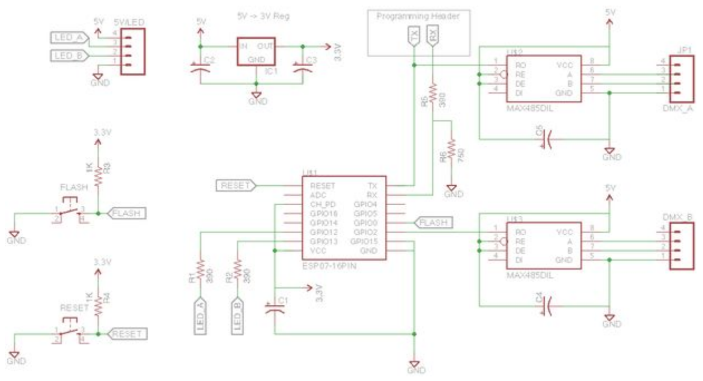
I started with a bread board. If you plan to make modifications to my schematic, I'd recommend you do the same. I haven't included a photo as I forgot to take one and my breadboards are in use for a new project I'm working on. Check out my ESP8266 Breadboard Instructable for details on making the ESP8266 breadboard friendly and then hook up your components according to the schematic.

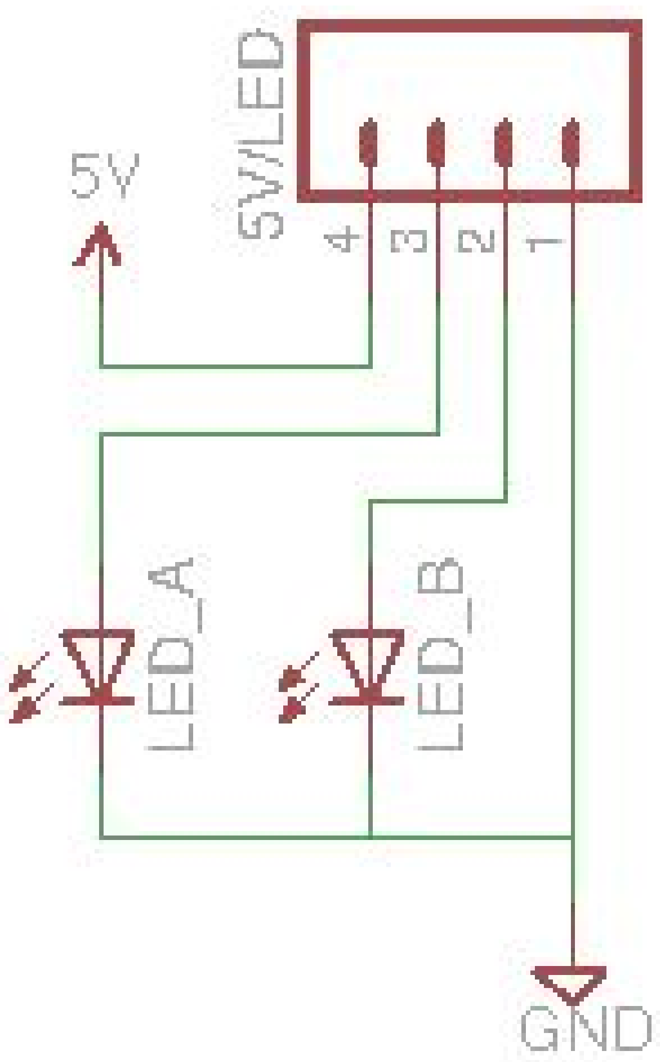
On your perf board, layout your components in a way that makes sense for the required wiring and that will fit in your case. I wanted the final unit to be as small as possible - this meant I did this step in conjunction with the case layout (next step) in order to make sure it all fits.

In the image, you can see 3x 4pin headers. The 2 at the top are DMX outputs and the 1 at the bottom is power and LED connections. The 2pin black header in the middle is the RX/TX for programming the ESP8266. Note that the RX pin has a voltage divider (R5 & R6). This is because I use a 5V programmer (my Arduino Mega) and so I need to drop it to 3.3V so the ESP isn't fried. These resistors can be deleted if you have a 3.3V programmer.

I neglected to include reset or program buttons to save space and I didn't have any on hand. For reset, I simply cycle the power. To enable programming, I put a small loop of wire on the reset pin. I connect an alligator clip to the wire loop and then ground it to the DMX header's GND pin. Although this is a pain, I included OTA updates in software so it's only for the first flashing.

I have also chosen not to isolate the DMX lines because I was feeling lazy :) I also wanted to save on board space and keep the cost down (DC-DC converters are expensive). I would recommend optical and electronic isolation. I did this in my DMX LED strips Instructable and the same process would work here - just reverse the optoisolators as we're outputting here as opposed to input. You will need a DC-DC and an opto on each DMX output.





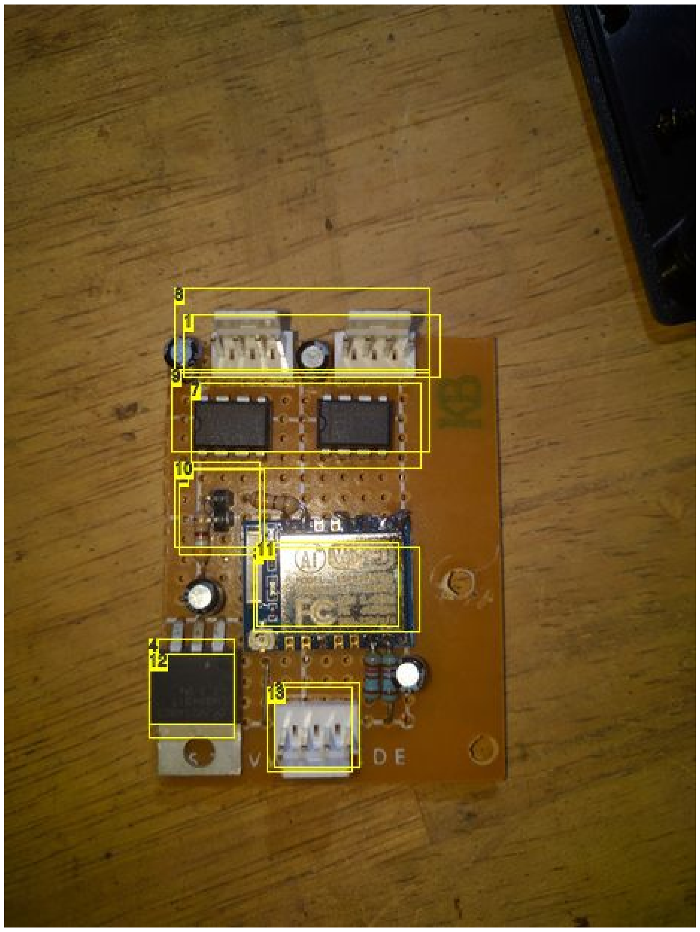


Image Notes
1. DMX Outputs
2. Serial TX/RX for flashing the ESP8266
3. ESP82366 07
4. 3.3V voltage regulator
5. 5V in and LED out
6. 5V in and LED out
7. Max485
8. DMX Out
9. Max485
10. Serial RX/TX for flashing and debug
11. ESP8266 07
12. 3.3V Regulator
13. 5V in, LED out

Step 3: Case Construction

I designed the case in conjunction with the circuit board in the previous step to ensure everything fit but used the minimum space.

I wanted all connectors on one side as it looks nicer and makes cable management easier. Because of this, the connectors were the very first thing I did. I drilled holes for the 2x XLRs, 2x LEDs and the antenna. I used a knife to cut the IEC hole. I stupidly used a female IEC connector when cutting the hole. It has a larger size than the male - hence all the hot glue around the IEC.

Once the holes are cut, I cut the pins shorter on the USB PSU to make it fit in the case. I drilled holes in the plug pins and attached wires - brown is live and blue is neutral (NZ/Australian standards).

Wire connectors to the XLRs (the blue cables).

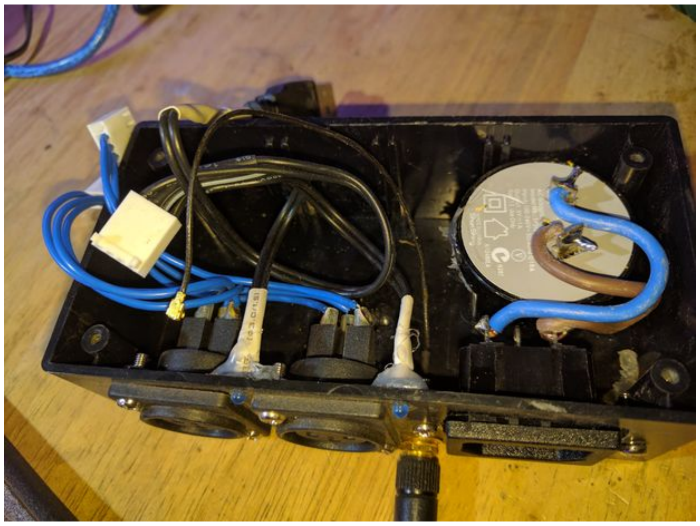
Wire LEDs and power to a 4pin connector. This is all the black cables in the photos. I used an old USB cable for power. You can find schematics on Google but I used a multimeter to check the 5V and GND wires. Check the LED polarity - the cathode (short pin) connects to GND and the anode (long pin) connects to pin 2 or 3 of our connector. Some heat shrink will make everything look pretty and stop shorts.

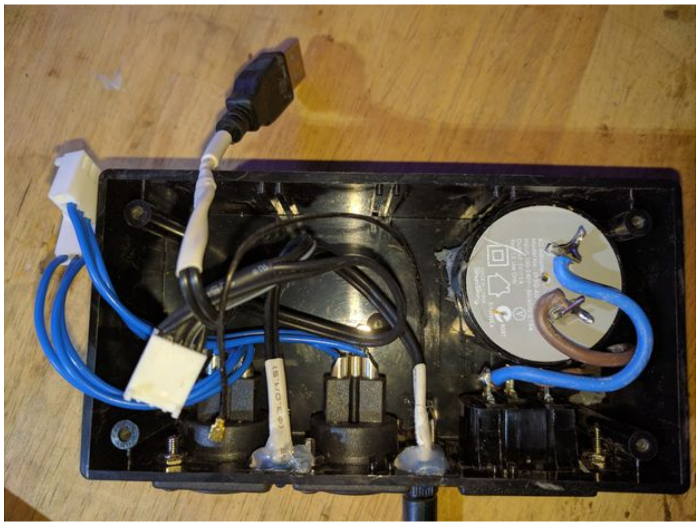
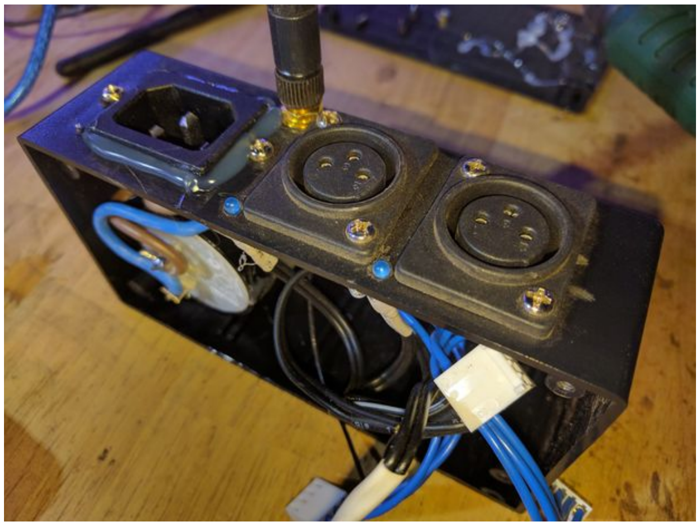
Screw in the XLRs, clip/glue in the IEC, screw in your antenna, and hot glue the LEDs into their holes.

Slide the circuit board into place. I drilled a single hole through the case and the board for an M4 screw and nut. You could glue it if you prefer. This ensures it wont move around.

Finally insert the USB connector into the PSU. I put some hot glue on it to ensure it stays in place. Plug in your 3 headers and antenna.

Now it's time to flash our firmware....



Step 4: Flash Firmware

Connect Programmer

Connect your USB -> Serial adaptor. I use my Arduino Mega... on the Mega, simply short Reset to GND and connect megaTX -> espTX, megaRX -> espRX, megaGND -> espGND. Ensure you have a 3.3V serial - if it's 5V (as with the Mega) then use a voltage divider as shown on the schematic wiring page.

Hold the flashing pin 0 to GND - I use alligator clips as explained on the construction page.

Connect power (or press reset). Your device should now be in flash mode.

Get Files Needed

I use the Arduino IDE with the Arduino core for ESP8266 for coding and flashing the firmware. Install everything according to the instructions on their git.

Download my espDMX library to your Arduino IDE library folder. I wrote this library to output 2 universes of DMX using interrupts to ensure precise timing. The DMX output has a refresh rate of approximately 44Hz - the maximum you can get with a full 512 channels. When looking at the output with an oscilloscope, it is picture perfect on both outputs. I used the timings from Ujjal's DMX512 Pages. The library is heavily based on the hardware serial library from the core files linked above. The hardware serial library was a bit bloated for this purpose and didn't utilize hardware interrupts - hence why I modified it to suit. In the future, I may enable receiving DMX but I doubt it - I see no use case for this.

Download my esp8266_artnetnode_dmx code to your projects folder. You can use the bin file provided and flash OTA for the latest updates. OTA updates means no compiling of code is required. More on OTA later.

Flash Firmware

Open ESP8266_ArtNetNode_DMX.ino in the Arduino IDE.

In the Tools menu, choose Generic ESP8266 Module.

Set the settings according to the image above. Note that the **flash size** needs to be set according to the size of your chip. Use the CheckFlashConfig example (under File > Examples > ESP8266 in Arduino IDE) to check your chip size. The first number is the total flash chip size. The SPIFFS number is how much to assign to the file system - we don't use the file system so it doesn't matter what this number is. If your flash size is smaller than 1M then I'd recommend making the SPIFFS smaller to allow more space for our program.

Select your USB -> Serial programmer under Port.

Click the upload arrow. If the library and core files are correctly installed, it should compile and flash without too much fuss.

Settings

Most of the settings are done via the web browser. I'll go over this in the next step.

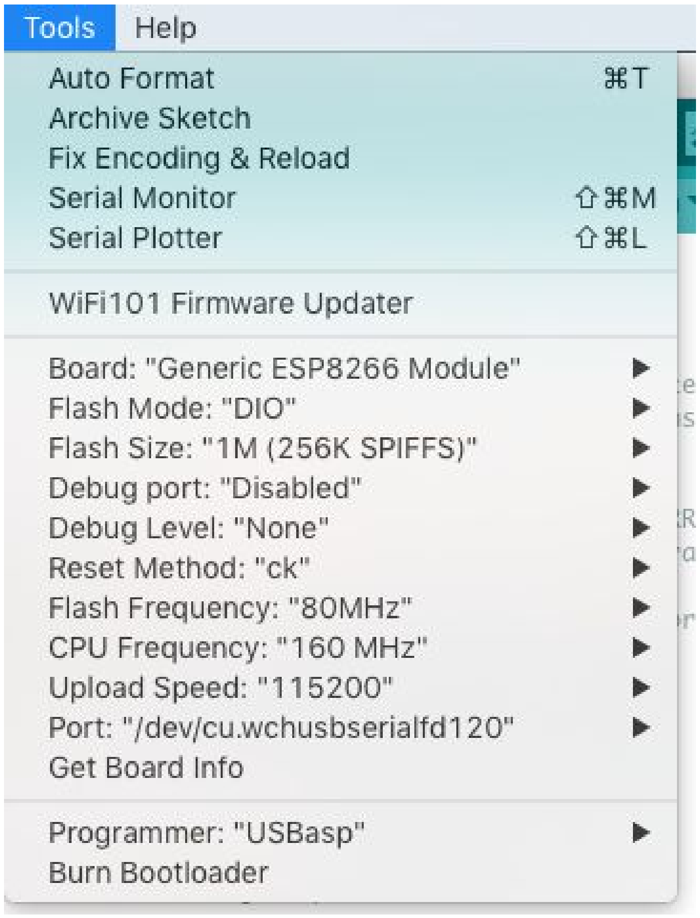
The LED pins are set in firmware. They default to pins 12 & 13 but you can change them if you want.

Troubleshoot

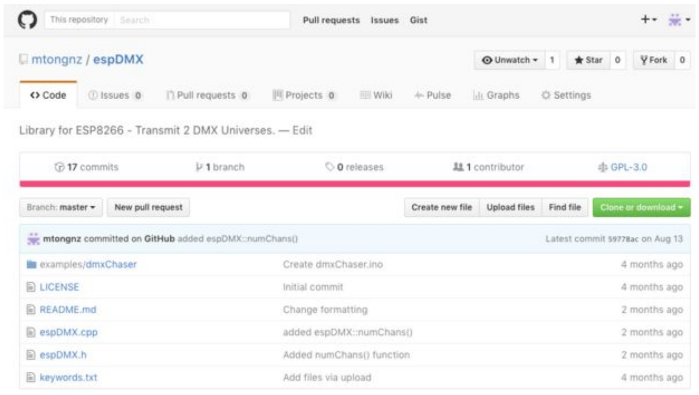
There are 2 define statements commented out at the top of ESP8266_ArtNetNode_DMX.ino

Uncomment #define VERBOSE to have the device spit out information about what is happening at any given time via serial0 (the serial port you have already connected via USB -> serial above). This will disable one of your DMX outputs and display wifi passwords in plain text so make sure it's disabled before deployment!

Uncomment #define LOAD_DEFAULTS to clear any settings and reload factory defaults. Make sure you comment this line out and reflash or you cant change any settings - each time the device starts, it'll clear them all.



```
Tools   Help
Auto Format                              ⌘T
Archive Sketch
Fix Encoding & Reload
Serial Monitor                          ⇧⌘M
Serial Plotter                          ⇧⌘L
WiFi101 Firmware Updater
Board: "Generic ESP8266 Module"           ▶
Flash Mode: "DIO"                         ▶
Flash Size: "1M (256K SPIFFS)"            ▶
Debug port: "Disabled"                    ▶
Debug Level: "None"                       ▶
Reset Method: "ck"                        ▶
Flash Frequency: "80MHz"                  ▶
CPU Frequency: "160 MHz"                  ▶
Upload Speed: "115200"                    ▶
Port: "/dev/cu.wchusbserialfd120"         ▶
Get Board Info
Programmer: "USBasp"                      ▶
Burn Bootloader
```



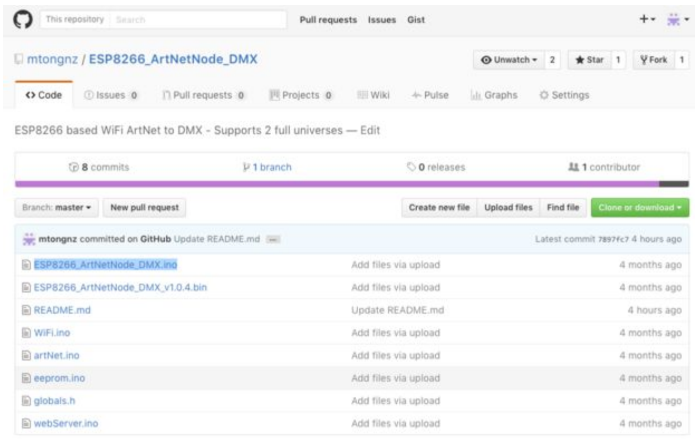
mtongnz / espDMX

Library for ESP8266 - Transmit 2 DMX Universes. — Edit

17 commits · 1 branch · 0 releases · 1 contributor · GPL-3.0

mtongnz committed on GitHub added espDMX::numChans() · Latest commit 59778ac on Aug 13

examples/dmxChaser	Create dmxChaser.ino	4 months ago
LICENSE	Initial commit	4 months ago
README.md	Change formatting	2 months ago
espDMX.cpp	added espDMX::numChans()	2 months ago
espDMX.h	Added numChans() function	2 months ago
keywords.txt	Add files via upload	4 months ago



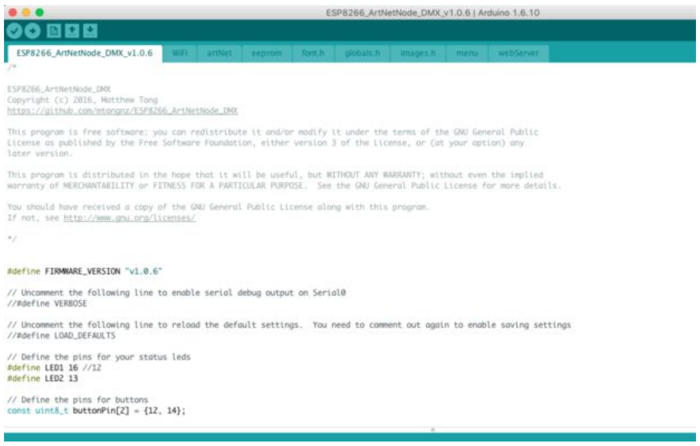
mtongnz / ESP8266_ArtNetNode_DMX

ESP8266 based WiFi ArtNet to DMX - Supports 2 full universes — Edit

8 commits · 1 branch · 0 releases · 1 contributor

mtongnz committed on GitHub Update README.md · Latest commit 7897fc7 4 hours ago

ESP8266_ArtNetNode_DMX.ino	Add files via upload	4 months ago
ESP8266_ArtNetNode_DMX_v1.0.4.bin	Add files via upload	4 months ago
README.md	Update README.md	4 hours ago
WiFi.ino	Add files via upload	4 months ago
artNet.ino	Add files via upload	4 months ago
eeprom.ino	Add files via upload	4 months ago
globals.h	Add files via upload	4 months ago
webServer.ino	Add files via upload	4 months ago



ESP8266_ArtNetNode_DMX_v1.0.6 | Arduino 1.6.10

```
/*

ESP8266_ArtNetNode_DMX
Copyright (c) 2016, Matthew Tong
https://github.com/mtongnz/ESP8266_ArtNetNode_DMX

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any
later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program.
If not, see http://www.gnu.org/licenses/

*/


#define FIRMWARE_VERSION "v1.0.6"

// Uncomment the following line to enable serial debug output on Serial0
//#define VERBOSE

// Uncomment the following line to reload the default settings.  You need to comment out again to enable saving settings
//#define LOAD_DEFAULTS

// Define the pins for your status leds
#define LED1 16 //12
#define LED2 13

// Define the pins for buttons
const uint8_t buttonPin[2] = {12, 14};
```

Step 5: Settings

Connect to WiFi Hotspot

When you first start the device, It will attempt to connect to the WiFi network defaultSSID (which I doubt exists). If it can't connect within 30 seconds, it will start a hotspot. The hotspot is nodeName_XXXX (nodeName by default is artNetNode and XXXX is a random number). Connect to this hotspot using the password artNodeXXXX (XXXX is the number from the hotspot SSID).

Settings Web Page

Point your browser of choice to the settings page. If you connect via the hotspot, the address of the web page will be 192.168.4.1 but if the ESP has connected to your WiFi then use it's IP address (look at your router if it's DHCP assigned or use a static address you've assigned).

At the top of this page, you will see the MAC address. Use this to allow your device on a MAC filtered network (this is highly recommended).

Next is the Node Name. Change this to anything (less than 30 characters) to make the device easy to identify in your lighting software. It will also change the WiFi hotspot name - password stays the same however.

Next is your WiFi settings. Enter the SSID and password of your show network.

Hotspot timeout is the delay before the hotspot is started (when the WiFi network wont connect). The hotspot only starts when the device is power cycled, not if the WiFi connection drops after being connected.

Now for the Artnet settings. Subnet is your Artnet subnet, not to be confused with your network subnet. Set the required universe addresses - these can be the same if you want both outputs to be the same.

IP Settings allows you to see the current IP settings (DHCP by default). You can set static addresses if you wish. Artnet networks should run in the 10.0.0.0 or 2.0.0.0 ranges but will work on others.

Certain settings can be saved and will take immediate effect such as changing the Artnet settings. Other settings need a reset (IP and nodeName). Use the buttons provided to do this.

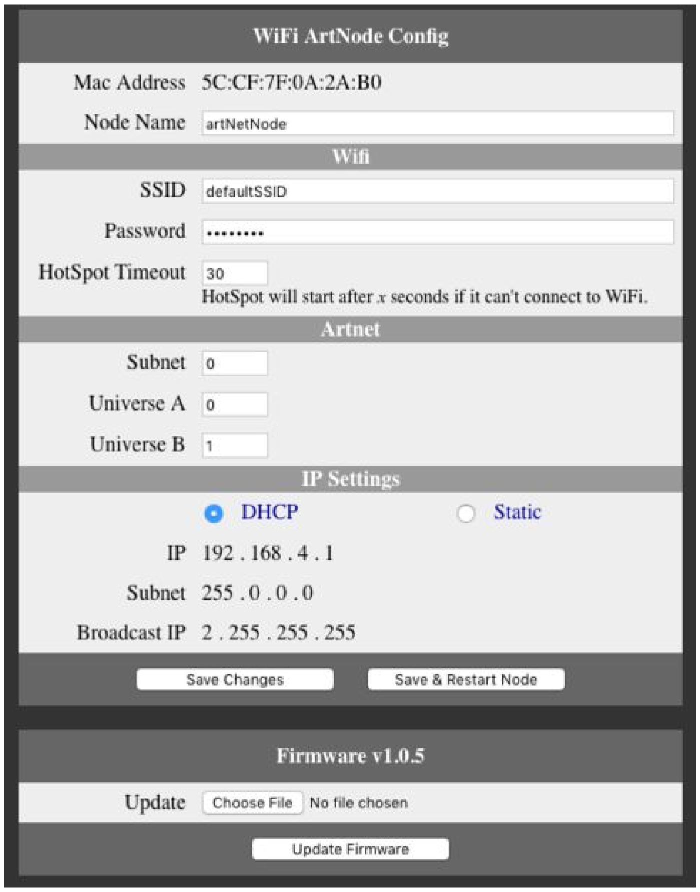
Firmware Update

The last item on this page is the firmware info and update box. This allows you to easily see which firmware version the unit is running and upload a new version via WiFi. Uploading new firmware will terminate all DMX output and stop responding to artnet until complete. Occasionally it fails so check the expected firmware version is displayed after an OTA update.

In order to produce the OTA binary, select Export compiled Binary from the Sketch menu of the Arduino IDE. This will create a .bin file in your project directory.

OTA Fails

If you find the OTA constantly fails, the WiFi wont connect or the hotspot wont start, check the Flash Size setting in the Tools menu. If this doesn't match your chips flash size, it causes weird issues. Use the CheckFlashConfig example (under File > Examples > ESP8266 in Arduino IDE) to check you are using the correct settings.



WiFi ArtNode Config

Mac Address 5C:CF:7F:0A:2A:B0

Node Name artNetNode

Wifi

SSID defaultSSID

Password ••••••••

HotSpot Timeout 30

HotSpot will start after *x* seconds if it can't connect to WiFi.

Artnet

Subnet 0

Universe A 0

Universe B 1

IP Settings

[✓ DHCP] Static

IP 192 . 168 . 4 . 1

Subnet 255 . 0 . 0 . 0

Broadcast IP 2 . 255 . 255 . 255

Firmware v1.0.5

Update No file chosen

Step 6: Let there be light!

By now, your device should be connected to your show WiFi network and you should be able to connect to the settings page. Now to setup the lighting console/software...

First, ensure your console is connected to the network with an IP in the same subnet range as your artnet devices. This should be the 10.0.0.0 or 2.0.0.0 ranges.

All lighting software is different in the specifics of how these settings are implemented but all follow the same basic principles... Set a certain universe of data to be output to a certain artnet port (set by the subnet and universe). This data can be multicast (sent to every device on the network) or unicast (sent to a specific device). I recommend using unicast as it has less network overheads so should perform better. Our espArtnetNode will receive either provided the subnet and universe settings match. It responds to ArtPoll requests so it should show up correctly in your lighting software of choice.

Jands Vista Setup

I use Jands Vista as my console of choice so I'll show screenshots of the process and explain where the settings are. You can download and try Vista for free (Mac and PC) - but it will do random blackouts if you don't purchase a channel dongle.

I'm assuming you have some lights patched already - I'm not going to cover this as Jands has a bunch of great tutorials on this and other basics available on their YouTube channel.

On the Vista, goto your Patch view and select Connect Universes in the top right of the screen. There will be a list of available outputs for any consoles or wings that are detected.

Your espArtnetNode should also be listed with 2 available ports. Assign which universe of data you want to send to each of the ports. This will setup a unicast artnet connection. As soon as this is done, the activity LEDs on your espArtnetNode should start flashing to indicate that DMX is being output.

You could also select Add Network Connection on the bottom left of the Connect Universe dialog. This will allow you to add an artnet broadcast port. Use the subnet and universe setting from your device. Now assign the data universe to the broadcast port. This will establish an artnet broadcast. Any devices on the network can be set to receive this subnet and universe of artnet data.

Plug It In

I'm pretty sure you've already figured this out.... Plug the DMX outputs into your light(s) of choice and take it for a spin. I'd highly recommend these awesome RGB LED strip controllers that I designed a couple of years ago. #shamelessPlug

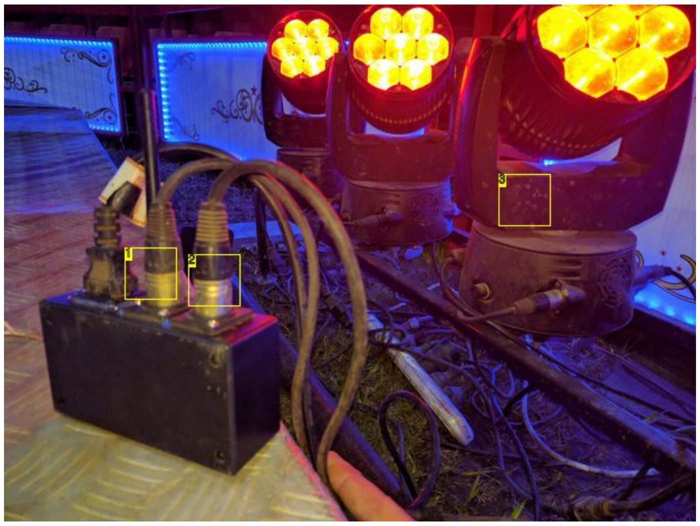


Image Notes
1. Universe A - RGB LED Strips. See my other Instructable to see how to make them!
2. Universe B - LED Moving Heads
3. Ignore the dirty state of the lights. As you can see, they're cheap chinese junk sitting on grass - they get very dirty!

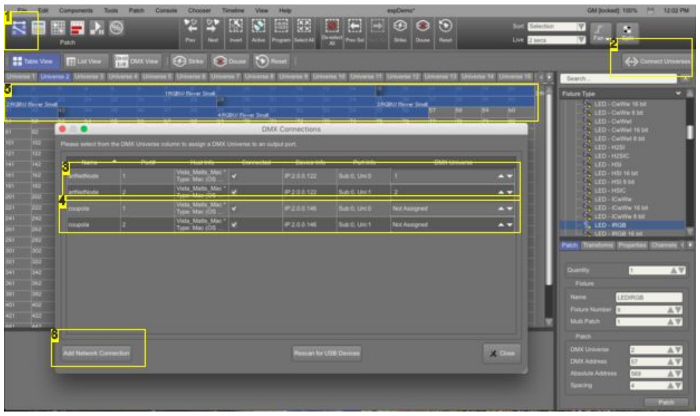


Image Notes
1. Patch View
2. Connect Universes
3. My new device
4. Another ESP device already in use
5. My LED movers patch to universe 2
6. Add a multicast connection

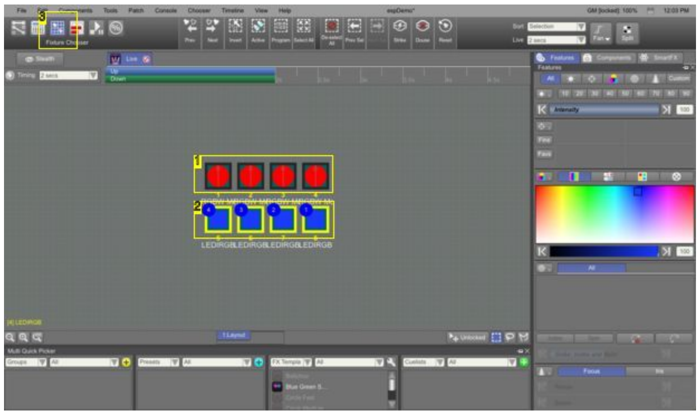


Image Notes
1. My moving heads - in red with full intensity
2. My LED strips - in blue with full intensity
3. Fixture Chooser View

Step 7: LED Status

This is an overview of what the 2 status LEDs do.

On power up, both LEDs will light to indicate the device has started. They will stay fully lit while the node attempts to connect the WiFi.

Once the hotspot timeout is reached (30 second default), the LEDs will alternate about twice per second. This will continue until a client connects to the hotspot. If a client connects, they both turn off. If no client connects within a minute, the device resets and it all starts again.

Once a WiFi network has been connected to, both LEDs will turn off until artnet data arrives.

When data starts arriving, the device buffers it. Once the first artnet packet is received, the device will start outputting DMX and the LED will flicker to indicate activity. A healthy artnet stream will have a constant flicker. Each output is handled separately.

A slow flash (about once per second) indicates that no new data is being received. This could indicate that the network has dropped, the console has stopped sending data, or simply that there isn't any new data. The device will output at least one full DMX universe each second to ensure devices remain stable and any new devices connected to the DMX chain will get the data needed.

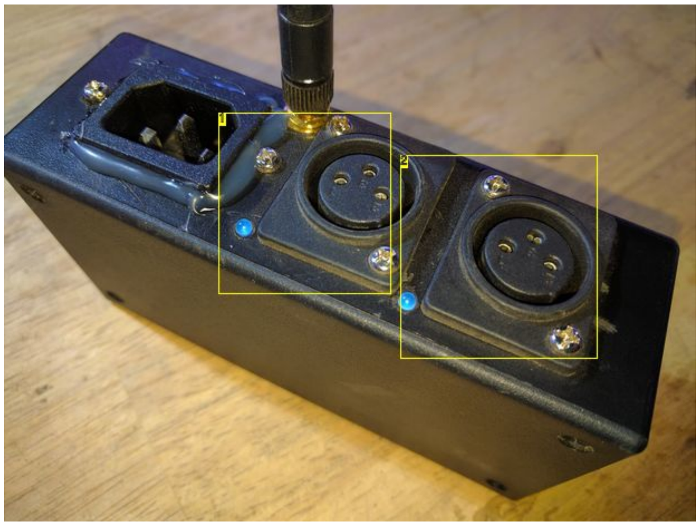


Image Notes
1. DMX Universe A and Status LED
2. DMX Universe B and Status LED

Step 8: Troubleshooting

I have run into very few issues. If you find any, please let me know.

WiFi Wont Connect

The ESP8266 is 2.4Ghz B,G,N capable. I found it didn't like one network I tried it on but as that was so long ago, I can't remember specifics.

Triple check SSID and password. I plan to add a plain view switch for password entry to a future release.

Check your MAC is on the allowed list for your network.

Ensure the flash size is correct in Arduino IDE when compiling/flashing the firmware. See the flash firmware step for more details and the fix.

Hotspot Not Starting

Check your hotspot timeout setting (30 seconds by default). If it's really high, your hotspot could take hours to start.

Check the device isn't connecting to a WiFi network. If the WiFi set on the settings page is available, it will connect to this before starting the hotspot.

Ensure the flash size is correct in Arduino IDE when compiling/flashing the firmware. See the flash firmware step for more details and the fix.

OTA Not Working

Ensure the flash size is correct in Arduino IDE when compiling/flashing the firmware. See the flash firmware step for more details and the fix.

No DMX Output

DMX is only output when connected to WiFi and valid artnet packets are received. Once this happens, it will output at least 1 full DMX universe each second of the last received data.

Check your lighting console and artnet node have matching artnet settings. Also check the network settings to ensure they can talk to each other. Also check that you have a licence for the software that allows it to output data. Jands Vista will do random blackouts if you don't purchase enough channels.

If the LEDs show activity but there's no DMX output, check your Max485 chips have power and are wired correctly. Also check that the LED and DMX output match (i.e LED_A represents DMX_A, not DMX_B). It's easy to poke the LEDs into the wrong hole.

Step 9: Ideas for the Future

Web Settings

I plan to add a plain view switch for password entry to a future release.

I am also looking into using AJAX and JSON to save settings rather than submitting the entire page.

LCD Menu

I am currently developing an LCD menu for this device. It will consist of a small 128x64 LCD and either 2 or 3 buttons. The menu will allow local setting changes and display critical info locally, without the use of a laptop or phone.

The main issue I am having at present is debouncing the buttons. It is very difficult to do this in code as I want to maintain the precise DMX timing I currently have - receiving artnet packets and DMX output is the priority. When I get a bit more time, I will be looking into hardware debouncing. I am thinking of using a small AVR running a loop with software debouncing and then sending button states to the ESP - this is simple and I have heaps of small AVRs on hand. I could also use the AVR to run the LCD, taking the load off the ESP.

Store Scenes

With the large amount of flash storage available, I was thinking that storing scenes on the ESP would be a great idea. This would allow simple scenes or chases to be run without a lighting console being present. This would tie in with the LCD menu, allowing selection of scenes from the menu or web interface.

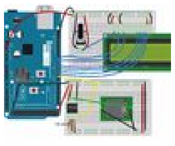
One DMX universe is 512 bytes so 2 universes would take 1K. I guess I could store 500+ scenes on the 1M flash available to me and more with the 4M versions of the ESP8266. This would allow some cool chases to be recorded.

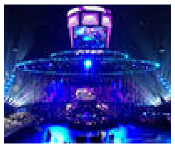
DMX is 44Hz so 44K space would be needed per second. This would mean you could fit a chase about 11.5s long. You could also record every second or third packet to allow for longer chases.

Captive Portal

I plan to add captive portal code to the hotspot shortly. This will mean that you don't need to enter an IP address - you'll be automatically sent to the settings page when the hotspot starts. Obviously this wouldn't work when connected to a WiFi network as the ESP8266 wouldn't be the DHCP server.

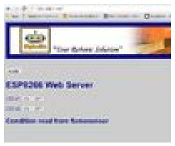
Related Instructables

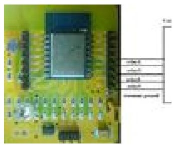

Arduino Artnet Node by DavidB56


Ethernet to DMX512 by ë³´í˜„ë

Arduino DMX 512 Tester and Controller ENG by daniel3514

Arduino DMX 512 Tester and Controller by daniel3514


How to build a ESP8266 Web Server by mybotic


Automate your Home Lights from web browser using Esp8266 by Magesh Jayakumar

Comments