

Craig A. Lindley's
Micro Controller Projects
Volume 1
The Amazing ESP-8266

Craig A. Lindley
Black Forest, Colorado, USA

Important Notes

- *Please do not give away copies of these articles and/or code to other people. The meager amount of money I make on this download will be used to finance new projects that I will design and build and publish in the future.*
- *There are absolutely no guarantees about the projects published in this document. I can verify that the projects did work when the articles were originally written but things in technology change at a lightning pace. Development tools change, libraries on which the projects' depend change and all of this is outside of my control. I do, however, believe if you are a relatively competent electronic enthusiast / programmer all of these projects can be made to work with a little bit of effort.*
- *Product support is not included in the purchase of this document. Support is available, however, on a contract basis. Please email calhjh@gmail.com for more information.*
- *You are free to use code from these articles in any way you want including commercial products, but you are not allowed to use my name to endorse anything you do.*
- *Republication of any part of this document is strictly prohibited without prior written permission from Craig A. Lindley, the author.*
- *The complete contents of this document and the accompanying software / code are Copyright 2017, 2018 by Craig A. Lindley, All Rights Reserved.*

Who Am I

My name is Craig A. Lindley and I am a consummate tinkerer, maker, beer brewer and musician from Black Forest, Colorado. I have a BSEE degree in electronic engineering from Cal Poly, Pomona, California. My love of micro controllers / micro processors began around 1976 when I built my first S100 bus computer using the Signetics 2650 micro processor. Shortly after that I got a job at Jet Propulsion Laboratory (JPL) where I honed my hardware and software skills working on the central data system for the Galileo spacecraft. My name was actually etched onto the gold record which flew to Jupiter on Galileo and orbited the planet before plunging to its death into the planet. Since that time I have worked for many large tech firms including: IBM, Sun Microsystems, HP, Rolm and TRW sometimes as a hardware engineer and sometimes as a programmer. I even had my own company, Enhanced Data Technologies, for many years before returning to private industry. Late in my career I spent time writing large Java Enterprise applications for numerous startup companies. I returned to the embedded world after I retired from industry and since then I have written extensively about any aspect of micro computers that caught my attention / interest. I have over 75 publications to date including 5 books (or 6 if you include what you are now reading). My books include:

- "TRS-80/Z80 Assembly Language Library", published by Wayne Green Publishing, Inc., Peterborough, New Hampshire. ISBN 0-88006-060-3. 1983.
- "Practical Image Processing in C" published by John Wiley and Sons, Inc. in November of 1990. ISBN 0-471-54377-2. This book has been translated into Chinese, Korean and Russian.
- "Practical Ray Tracing in C" published by John Wiley and Sons, Inc. in November of 1992. ISBN 0-471-57301-9.
- "Photographic Imaging Techniques in C++ for Windows and Windows NT", published by John Wiley and Sons, Inc. in November of 1995. ISBN 0-471-11568-1.
- "Digital Audio with Java" published by Prentice-Hall in January of 2000. ISBN 0-13-087676

Information about all of my publications can be found at: <http://craigandheather.net/cwripage.html>.

I also hold five US patents in various aspects of computer engineering.

What's in this Document?

Most chapters of this document are articles I have previously published. Most of these articles were originally published by Nuts and Volts magazine and permission has been obtained for republication. I have chosen these articles for inclusion because they have generated the most email from Nuts and Volts readers or have been the most downloaded from my website: <http://craigandheather.net>. I have also included some new unpublished construction articles and other material which I call snippets. Snippets are not full construction articles but rather experiments or proofs of concepts that I thought would be of interest. All of the code that goes along with the articles in this document is also provided.

This is volume one in a series of PDF documents that deals specifically with projects for the ESP8266 family of micro controller devices. Future volumes will feature articles I have written about the Raspberry Pi and Teensy micro controller devices. A possible additional document may cover articles that don't fit neatly into the micro controller organization of the documents described above.

Introduction

There is something fun and amazing about conceptualizing an electronic device and then setting out to design and build it. There are always frustrations along the way but once it works there is really a sense of pride and accomplishment. Building projects designed by others also has its own rewards. Being able to say, "I made that" when asked about something you have built it is very fulfilling. This is especially true when the projects you build fill some need you have. You can probably buy most of the project devices I talk about in this document but why would you if you can build it yourself and probably save substantial money in the process.

That being said, not everyone has the ability to build these projects by themselves. The ability to run a personal computer, basic programming skills, the ability to read schematics, the ability to solder, determination, curiosity and self motivation are all required to make these projects work. If you lack any of these skills but wish to pursue these projects find someone who can help you. Find a friend or join a MakerSpace and you will have access to all of the skills and tools needed. Even your local library may have facilities that can help you with these projects. I know I was surprised to find out that my local library had a laser cutter machine which I used for one of the projects in this document.

In addition to required skills, you will need the Arduino Integrated Development Environment or IDE (see <https://www.arduino.cc/en/Main/Software>) to take the code presented in these articles and program the ESP8266 module on which all of these projects are based. Within the IDE you can edit the provided code, compile it over and over until it is error free and then upload the finished code it to ESP8266. You can also used the Serial Monitor from the IDE for inspecting the values of program variables to help you debug the code when things aren't working correctly.

There are many versions of the Arduino IDE available but I recommend always using the latest version. I used version 1.8.0 for macOS for most code development but you can develop on the latest version for Windows as well. Version 1.8.0 was the current version at the time of writing but maybe an out of date version in the future when you try and build one of these projects. For the most part newer versions of the IDE are backwards compatible with older versions but some tweaks to the code may be necessary to get the provided code to compile cleanly.

Before being able to compile the provided code you will need to install the ESP8266/Arduino libraries into the Arduino IDE. Instruction for how this is done are available at: <https://github.com/esp8266/Arduino>.

Speaking of libraries; many of the projects in this document make use of third party libraries for their operation. The libraries I used for development are include with the project code and should be used. Older or newer versions of these libraries may not function correctly. It is important to note that if you install a library while the Arduino IDE is running you have to exit the IDE and restart it for the library to be recognized. General instructions for installing Arduino libraries are available at: <https://www.arduino.cc/en/Guide/Libraries>.

With the Arduino IDE all setup and all required libraries installed, make sure you select a board type from the Tools menu that corresponds to the version of ESP8266 your are using in your project. Most projects in this document use the *NodeMCU 1.0 (ESP-12E Module)* board type. Most projects can use either a CPU Frequency, also setttable from the Arduino Tools menu, of 80 or 160 MHz.

If you have trouble compiling the code for any of these projects you may have to use your programming skills and determination to figure out why. If there are undefined item errors you may have forgot to install a required library or the library you are trying to use is the wrong version. Try not to get frustrated because it is usually a simple thing that is causing you problems. You may be presented with a hundred compilation errors that are all caused by a single, simple typing mistake.

So say you get your Arduino setup correctly and you have loaded the project code and compiled it correctly without error and uploaded it to your ESP8266 without error but it doesn't work or doesn't work as expected. What do you do? First thing to do would be to check and double check the project's electrical wiring. Maybe use a magnifying lamp to make sure you have connected to the proper pins of the ESP8266 since the labels are rather small and sometimes overlapping. Next, check the power supply you are using to power your project when you are absolutely sure the circuit is correctly wired. I cannot tell you how many times I have under estimated the current requirements of a project and used an inadequate power supply which caused my project to malfunction, sometimes intermittently. When all else fails, insert *Serial.print*, *Serial.println* and/or *Serial.printf* statements in the code at key location outputting the values of key variables and use the Serial Monitor in the IDE to monitor them at runtime. This is a rather crude debugging technique but it is all you got in the Arduino IDE and I use it all of the time.

Parts Suppliers

Finding the required parts for projects can sometimes be a problem. Luckily there are many vendors available that combined carry anything you might need. Suppliers I have used include:

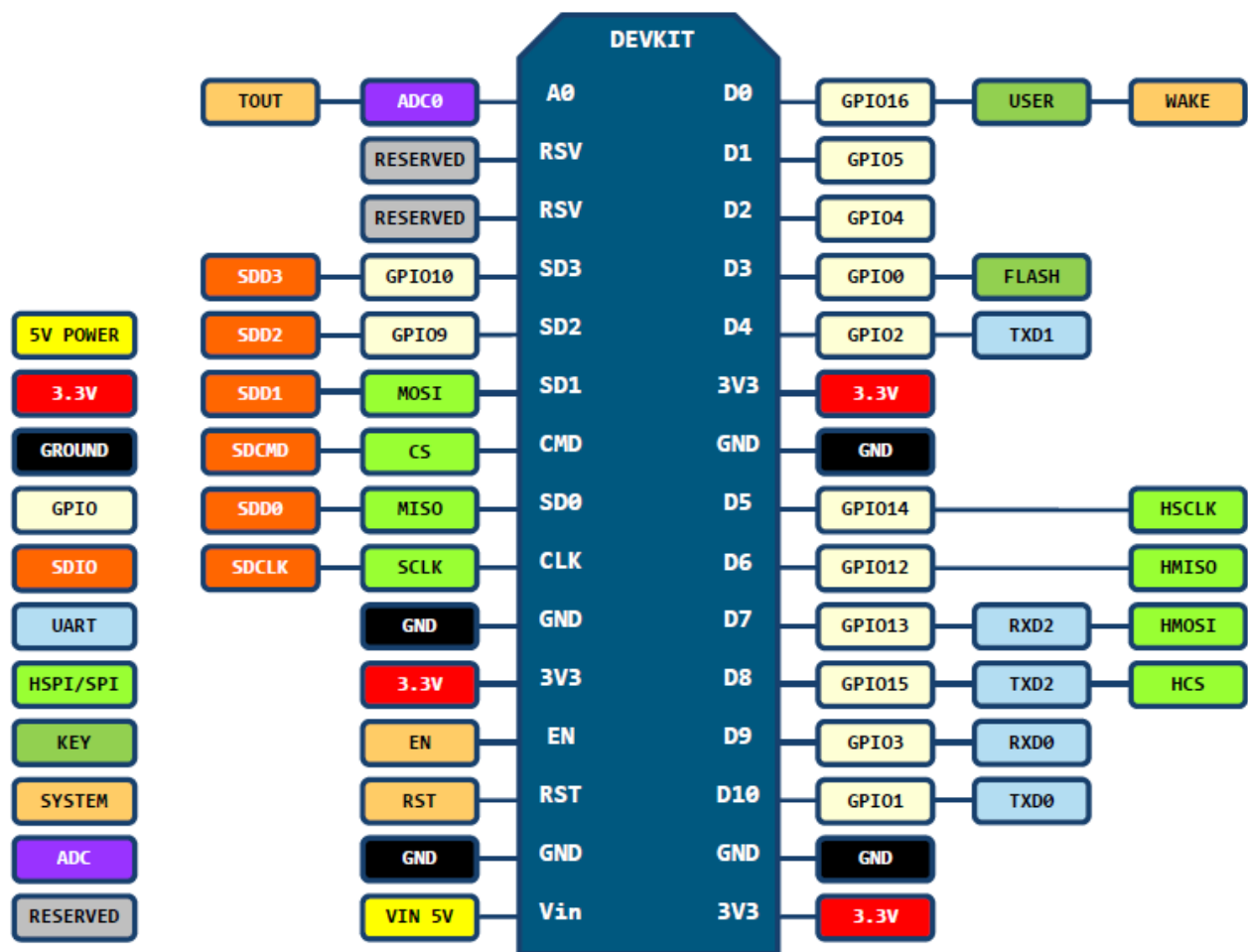
Supplier	Website
Digi-Key Electronics Corporation	www.digikey.com
Adafruit	www.adafruit.com
SparkFun	www.sparkfun.com
RadioShack	www.radioshack.com
Parts Express	www.parts-express.com
Newark, MCM Electronics, element14	www.newark.com/mcm-partnership
eBay	www.ebay.com
Jameco Electronics	www.jameco.com

Supplier	Website
Mouser Electronics	www.mouser.com
Electrodragon	www.electrodragon.com

NodeMCU Amica Pinout Diagram

Since most of the projects in this document use the NodeMCU Amica ESP8266-12 module from electrodragon.com, I've included this pinout diagram for reference.

PIN DEFINITION



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Table of Contents

Important Notes.....	2
Who Am I.....	3
What's in this Document?.....	4
Introduction.....	4
Parts Suppliers.....	5
NodeMCU Amica Pinout Diagram.....	6
Chapter One – A Tiny, WiFi Enabled, Arduino Compatible Micro Controller.....	11
Introduction.....	11
Hardware.....	11
Prototyping Hardware.....	12
Arduino IDE Version 1.6.4.....	13
Software.....	14
Teleduino.....	15
Conclusions.....	16
Resources.....	16
Chapter Two - NTP Clock.....	23
Introduction.....	23
Hardware.....	24
Software.....	26
User Configuration of the NTP Clock Software.....	28
NTP Clock Operation.....	29
Resources.....	30
Chapter Three - Weather Clock.....	35
Introduction.....	35
Hardware.....	36
Software.....	37
User Configuration of the Weather Clock Software.....	38
Weather Clock Operation.....	39
Conclusions.....	42
Resources.....	42
Chapter Four - World Clock.....	53
Introduction.....	53
Hardware.....	54
Software.....	55
User Configuration of the World Clock Software.....	57
World Clock Operation.....	57
Timezones, Timezones, Timezones.....	58
Conclusions.....	59
Resources.....	59
Chapter Five - Nixie Tube Clock.....	67
Introduction.....	67
Clock Operation.....	67
Hardware.....	68

Software.....	70
Packaging.....	72
Conclusions.....	73
Chapter Six - RSS News Reader.....	85
Introduction.....	85
Hardware.....	86
Software.....	87
RSS News Reader Software Operation.....	89
Conclusions.....	92
Resources.....	92
Chapter Seven - NeoPixel LED Tree	97
Introduction.....	97
Laser Cutting Tree Pieces.....	97
The Electronics.....	99
Building The Tree.....	100
Tree Software.....	101
Remotely Controlling The Tree.....	102
Final Thoughts.....	102
Resources.....	103
Chapter Eight - Thinking Of You.....	117
Introduction.....	117
Hardware.....	118
Configuration.....	119
Software.....	120
How Things Work.....	121
Packaging the ToY Device.....	122
Conclusions.....	123
Resources.....	123
Chapter Nine - WiFi Robot and Robot Controller.....	133
Introduction.....	133
The Robot Controller.....	134
Hardware.....	135
Software.....	136
The Robot.....	138
Hardware.....	138
Software.....	139
Conclusions.....	140
Chapter Ten – NeoPixel LED NTP Clock.....	151
Introduction.....	151
Clock Operation.....	151
Hardware.....	152
Software.....	153
Conclusions.....	154
Snippet #1 ESP8266 & VS1053B Internet Radio.....	163
Introduction.....	163
Hardware.....	163
Software.....	164
Conclusions.....	167
Snippet #2 ESP8266 & VS1053B MIDI	171

Introduction.....171

Hardware.....171

Software.....172

Conclusions.....175

Chapter One – A Tiny, WiFi Enabled, Arduino Compatible Micro Controller

Introduction

It is not very often that a new piece of hardware comes along and immediately captures the attention of the entire maker community. The Raspberry Pi and the \$9 C.H.I.P. are a couple of recent examples but the ESP8266 module from Expressif Systems (*expressif.com*) wins this prize. This little board (see Photo One) is only about the size of a nickel yet contains a powerful 32 bit micro controller and a WiFi interface and can be purchased for around \$4 in single unit quantities.

The first projects built with this module all used a micro controller to control the ESP8266 as a WiFi peripheral using an AT command set over a serial interface. While this was made to work, some of the projects suffered from stability problems as the ESP8266 firmware continued to evolve. Lately however, a group of enterprising individuals have made the ESP8266 Arduino compatible. This is important for numerous reasons:

1. It allows people familiar with the Arduino IDE to develop software for the ESP8266 module.
2. It allows the software developed in the Arduino IDE to be run directly on the 32 bit micro controller on the ESP8266 module eliminating the need in many cases for a separate micro controller altogether.
3. It allows the use of numerous third party Arduino libraries as long as they don't depend on direct access to the underlying AVR hardware.

Arduino compatibility and the low cost of the ESP8266 is a major development for the Internet of Things (IoT) movement currently sweeping the tech world. Using the ESP8266 allows for very small and inexpensive products to be created that can be controlled and/or monitored remotely. Note if you plan on putting an ESP8266 module into a commercial product you will have to pass FCC certification which can take considerable time and be rather costly.

To understand what a breakthrough this is, consider the cost and size of a traditional Arduino approach to WiFi enabled monitoring and control. First you have to have an Arduino board say an Arduino Uno from a reputable source which costs between \$20 - \$30. Then you have to purchase a WiFi Shield for around \$20 - \$40 bringing basic system cost to between \$40 - \$70. Then consider size. The Uno's dimensions are 2.1" x 2.7". Attach the WiFi shield and the sandwich is between 1.25" to 1.75" deep and a bit harder to package than the ESP8266 which is the size of a nickel.

Finally when you consider the ESP8622 has a 32 bit processor which can run at 160 MHz, 10x the speed of the Uno's 8 bit processor, and that it has 512K (minimum) of flash memory program space to the Uno's 32K, the Uno Wifi solution is looking a little dated.

Hardware

Actually the ESP8266 is a whole family of modules which vary in the number of available I/O pins, the amount of onboard memory, the types of interfaces available and in how the RF antenna is

Chapter One – A Tiny, WiFi Enabled, Arduino Compatible Micro Controller

attached/implemented. The module I will be describing in this article (and shown in Photos One and Two) is referred to as an ESP-01. This module has its RF antenna etched directly onto the circuit board.

Information on the whole family of ESP8266 devices is available here:

<http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family>.

The following attributes of the ESP8266 family were extracted from the data sheet available at:

https://nurdspace.nl/File:ESP8266_Specifications_English.pdf.

- 802.11 b / g / n
- Wi-Fi Direct (P2P), soft-AP
- Built-in TCP / IP protocol stack
- 802.11b mode + 19.5dBm output power
- Built-in temperature sensor
- Supports antenna diversity
- Off leakage current is less than 10uA
- Built-in low-power 32-bit CPU which can double as an application processor
- SDIO 2.0, SPI, UART , ADC
- Standby power consumption of less than 1.0mW (DTIM3)

In other words, the ESP8266 family of modules features low power consumption, high RF power output and is capable of supporting all of the current 802.11 standards required for WiFi connectivity. In addition, it supports many industry standard hardware interfaces and can function as the application processor in many designs. Note: the ESP8266 is a 3.3 VDC part.

Prototyping Hardware

Now that we know something about the ESP8266 module family let's talk about what we need in terms of hardware to try out the ESP8266 in the Arduino environment. In addition to the ESP8266 ESP-01 module we need some sort of USB to TTL Serial adapter, a 3.3 VDC power supply capable of at least 250 mA of output current, a couple of momentary push button switches, an LED, a 1K and a 10K ohm resistor. Do not skimp on the power supply for the ESP8266. It requires quite a bit of current and lack of sufficient current will cause the ESP8266 to appear flakey or not work at all.

When I received my ESP8266 modules in the mail I was anxious to try them out but I didn't yet have the required USB to TTL 3.3V serial interface cable to proceed. Since necessity is the mother of invention and since I am not known for being a patient person, I decided to use an Arduino Uno board I had for this purpose. Note, I removed the processor from the Uno as it was not needed. The breadboard is shown in Photo Four and the Fritzing schematic in Figure One. Here, the 5V output of the Uno was used to drive a small switching mode power supply set to 3.3 VDC which in turn drives the ESP8266. The Tx and Rx signal from the Uno were connected directly to the Tx and Rx connections on the ESP8266. Yes Tx to Tx and Rx to Rx. I should point out that the ESP8266 is a 3.3V part and that direct connection to the 5V logic levels of the Uno should be avoided. With that being said, this prototype worked perfectly. I have since heard the ESP8266 has 5V tolerant pins but I have yet to have

that claim substantiated. Anyway, I figured I only had a few bucks tied up in the ESP8266 part so if it blew, oh well. As it turned out, this prototype worked splendidly.

The Reset push button on the prototype pulls the Reset pin on the ESP8266 low thereby resetting the device. The Flash push button grounds the GPIO0 pin which places the ESP8266 into firmware download mode. The CH_PD line must be pulled high for new firmware to be downloaded.

After my 3.3V USB to TTL serial cable arrived, I removed the Uno from the prototype and connected the cable directly to the ESP8266. This approach is shown in Photo Five and schematic in Figure Two. Here the cable provides 5VDC on the VCC pin which is connected to the 3.3VDC power supply. The Tx and Rx pins of the cable are at the proper 3.3V interface levels. The Tx pin of the cable is connected to the Rx pin of the ESP8266 and the Rx pin of the cable connects to the Tx pin of the ESP8266. I also added an LED and 1K resistor connected between ground and GPIO2 which will be used with Teleduino demo described later.

In either case the following series of steps must be followed to initiate successful loading of code from the Arduino IDE into the ESP8266 module.

1. Press and hold the reset button down
2. While holding the reset button down, press and hold the flash button.
3. Release the reset button while still holding the flash button down.
4. Click the Upload button in the Arduino IDE.
5. When the sketch starts to load, you can release the flash button.

Once code is successfully uploaded to the ESP8266 it will be executed every time a power up or a reset occurs.

Arduino IDE Version 1.6.4

To easily program the ESP8266 as an Arduino you must use the latest version of the Arduino IDE. As of this writing that is version 1.6.4. This version has a feature called the *board manager* which lets third party vendors add support for their Arduino compatibles that the makers of the IDE don't support directly. Adding support for the ESP8266 is a multi-step process.

1. First, you must download version 1.6.4 or newer version of the IDE from <http://www.arduino.cc/en/Main/Software> and install it. There are versions available for Windows, Mac OS X and Linux. The installation process is different for each platform but in each case is pretty straightforward. Follow the directions provided within the installation programs and you should be good to go.
2. Next bring up the Preferences page of the IDE and type http://arduino.esp8266.com/package_esp8266com_index.json into the Additional Boards Manager URLs field. With this completed click OK.
3. Next go to the Tools menu tab in the IDE and click Board and then Boards Manager. This will bring up a list of installable items. Scroll down and you should see **esp8266 by the ESP8266 Community**. Highlight this entry and an Install button should appear. Click this button to

Chapter One – A Tiny, WiFi Enabled, Arduino Compatible Micro Controller

install the ESP8266 development software. This can take awhile because a lot of software is being transferred to your computer.

Once this process has been completed, the next time you click the Tools menu and then the Board entry you should be able to select the “**Generic ESP8266 Module**”. You are now ready to program your ESP8266 as an Arduino. Make sure you make this selection for all of your projects which utilize the ESP8266 module. In addition, make sure you have a serial port selected in the IDE so the Serial Monitor can be used. If an appropriate serial port does not show up in the IDE you may have to install a driver for the USB Serial cable you are trying to use.

Software

With either variety of prototyping hardware in place and the Arduino IDE updated with ESP8266 support you are now ready to go. Right out of the box you have access to numerous example programs which illustrate some of the ESP8266 modules' capabilities. If you go to File and then Examples in the IDE you will see these three categories of example programs:

ESP8266mDNS
ESP8266WebServer
ESP8266WiFi

and each of these have one or more example sketches within them. Within the ESP8266WiFi category, for example, the following sketches are of special interest:

1. NTPClient - which shows how to get the time from a Network Time Protocol (NTP) server while demonstrating the use of UDP packets. This is the same technique your computer uses to set its time automatically.
2. WifiClient – which shows how to use the ESP8266 to talk across the Internet (as a client program) to a server. All client applications will resemble this example program including the Teleduino client discussed shortly.
3. WifiScan – which lists all of the wireless networks within range of the ESP8266. The network's name, signal strength and whether or not the network is encrypted is displayed on the Serial Monitor. The list of networks is updated every five seconds.
4. WiFiWebServer – shows how to use the ESP8266 as an HTTP type server. By locally accessing this web server with a browser, an LED connected to the ESP8266 can be toggled off and on. See the example sketch for more information.

Many of these example programs/sketches must be edited before running as you must enter the SSID of your wireless network along with your network's password. Without this information the ESP8266 will not be able to connect to your wireless network and the example programs will fail.

You may be wondering how much of the Arduino software environment has been ported to the ESP8266 and in truth the answer is quite a lot. A list of what is working is available at: <https://github.com/esp8266/Arduino>. This list is definitive as these are the people who did/are doing the Arduino port.

Teleduino

Using the ESP8266 to control an LED or some other device on your local area wireless network is cool but somewhat limiting. What if you want to control your device from anywhere in the world instead? There are multiple ways of doing this some of which require configuring your modem/router to forward messages through your firewall opening up the possibility of security breeches. Nathan Kennedy of KennedyTechnology.com has come up with a better idea that he calls Teleduino. Teleduino is designed primarily for use with an Arduino with a wired Ethernet shield and there are versions available for Uno and Mega based boards. I was interested to see if I could port some of the Teleduino functionality onto the ESP8266 as an experiment and with Nathan's help I did so. Please see www.teleduino.org for the details. If you are interested in the full Teleduino functionality on the ESP8266 you will have to wait for Nathan to port the complete code base. If however you want to experiment yourself you can grab my code (ESP8266_TeleduinoClient.ino) provided with the document.

To use the Teleduino code you must first go to: <https://www.teleduino.org/tools/request-key> and request an API key. A key will be emailed to you after you provide your name and email address. A key is a long string of hex characters which must be edited into the ESP8266_TeleduinoClient sketch along with your WiFi networks' SSID and password before downloading into the ESP8266. The API key must also be used in all API calls from your browser.

Once the sketch is downloaded into the ESP8266 it will first make a connection to your WiFi network and then it will open a TCP connection to the Teleduino server. Once this connection is made, the ESP8266 will authenticate itself to the server by providing the API key. If you bring up the Serial Monitor you will be able to watch this interaction take place. If all goes well the Teleduino server will begin sending ping messages to the ESP8266 about every 5 seconds. Because the ESP8266 establishes an outgoing connection to the Teleduino server there is no need to open any ports in your firewall and thus create any new security concerns for your network.

To summarize: once the ESP8266_TeleduinoClient is configured, it automatically connects itself to the Teleduino server when powered up. The Teleduino server translates instructions received over the internet into actions on the Teleduino device which in this case is the ESP8266.

As mentioned I ported only a (very small) subset of Teleduino functionality. In fact the ESP8266_TeleduinoClient sketch only recognizes a *setDigitalOutput* API call but that is enough to prove the concept workable. On the second prototype shown in Photo Five I have connected an LED to the ESP8266's GPIO2 pin through a 1K ohm resistor to ground. If I go to my browser and type in the following rather long URL:

https://us01.proxy.teleduino.org/api/1.0/328.php?k=<YOUR KEY GOES HERE>&r=setDigitalOutput&pin=2&output=1&expire_time=0&save=0

the LED will turn on. If I try this again but change output=0, the LED will turn off.

Keep in mind that for this demo the ESP8266 is turning an LED off and on by way of commands

Chapter One – A Tiny, WiFi Enabled, Arduino Compatible Micro Controller

entered into a browser which can be located anywhere in the world. If instead of an LED connected to the ESP8266 you connected a solid state relay (SSR) you could control devices such as a light, a heater, an alarm system, etc. The possibilities are endless.

Want to control your Teleduino device from your Android smart phone or tablet? Checkout apps like Teleduino Controller Pro V2 in the Google Play store.

Conclusions

This article doesn't begin to describe the cool things that can be done using the ESP8266. I hope after reading this article you will come up with many ideas for your own projects. I have a few projects in mind that I may share in future articles if there is enough interest. Please let Nuts & Volts know if you would like other article about using the ESP8266.

Devices like the ESP8266 make possible the idea of connecting almost anything to the Internet and controlling and/or monitoring them from anywhere in the world. The ESP8266 is a giant step forward in the IoT revolution.

Resources

The following links provide further information about ESP8266 devices.

Information about the ESP8266 Arduino port can be found at: <https://github.com/esp8266/Arduino>

An ESP8266 forum full of useful information can be found at: <http://www.esp8266.com>

The Expressif forum is available at: <http://bbs.espressif.com/>

The Teleduino client sketch (ESP8266_TeleduinoClient.ino) is available in the code provided with this document.

Photo One

The ESP8266 module (ESP-01) is about the size of a nickel

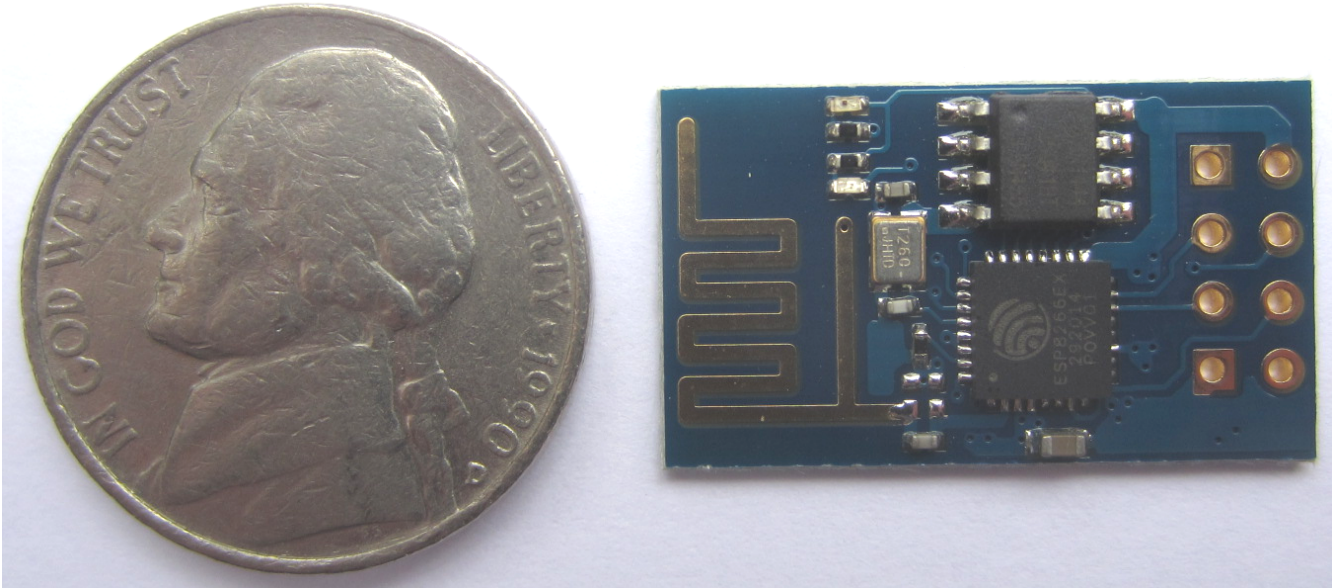
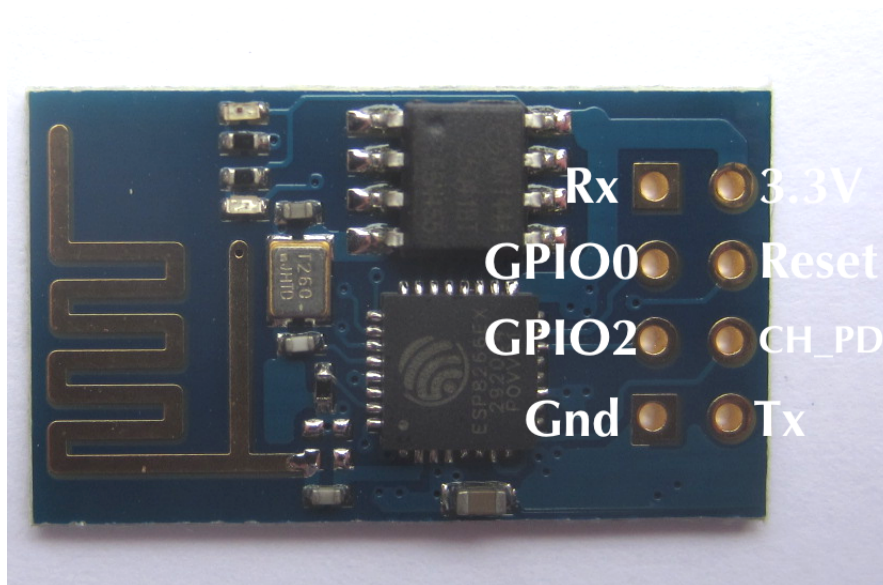


Photo Two

ESP8266 (ESP-01) Pinout

The squiggly trace is the WiFi Antenna

This device has 512K of Flash for program storage



Chapter One – A Tiny, WiFi Enabled, Arduino Compatible Micro Controller

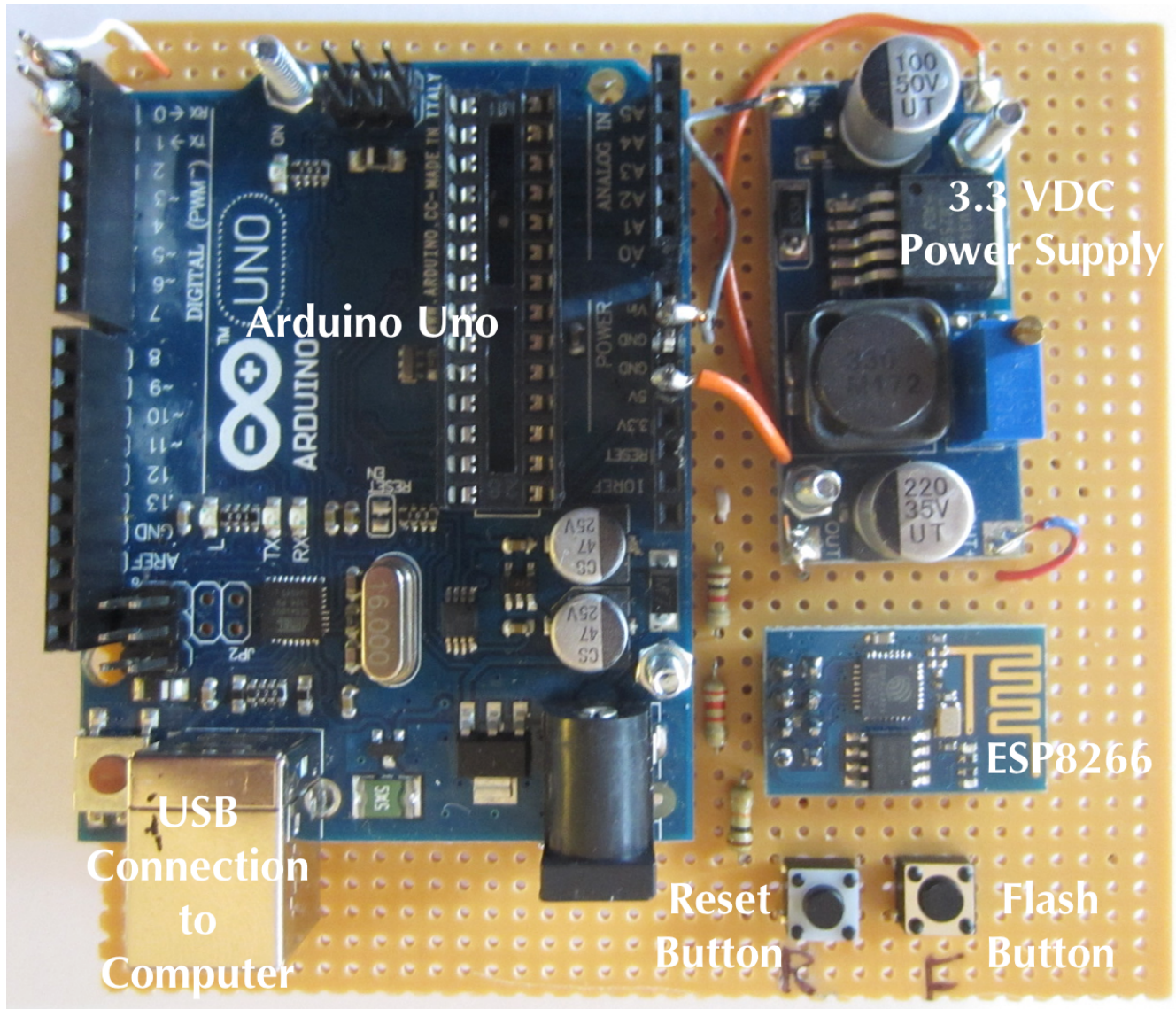
Photo Three

An ESP8266 (ESP-12) Development Module called the NodeMCU Amica with many more I/O pins and memory (4MBytes) available.

It functions just like the prototype hardware described in this article and is powered directly from the USB port



Photo Four
Prototyping Hardware using an Arduino Uno
This method is not recommended, but works



Chapter One – A Tiny, WiFi Enabled, Arduino Compatible Micro Controller

Photo Five

Recommended Prototyping Hardware using a USB 3.3V TTL Serial Cable
The LED is used for the Teleduino Demo

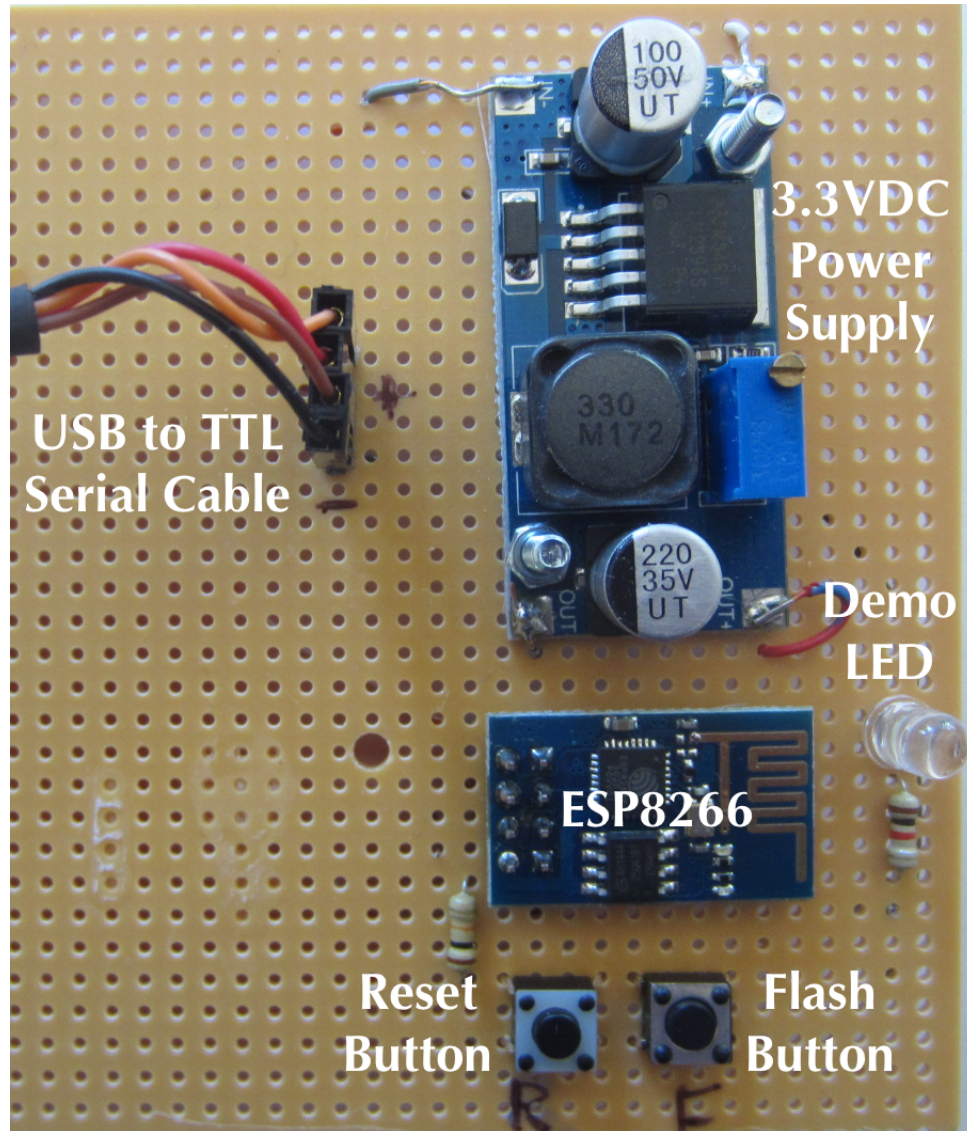


Figure One
Prototype Hardware using Arduino Uno
as USB to TTL Serial Converter

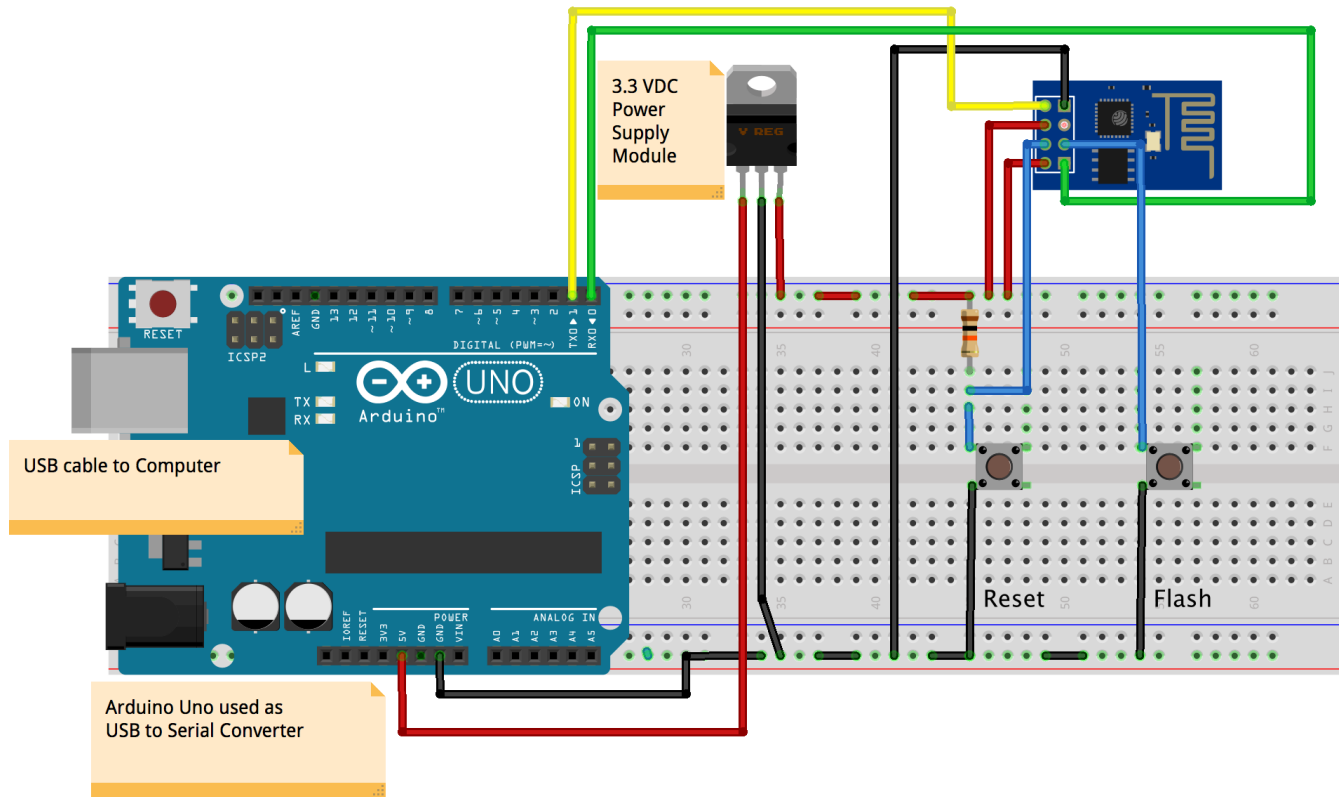
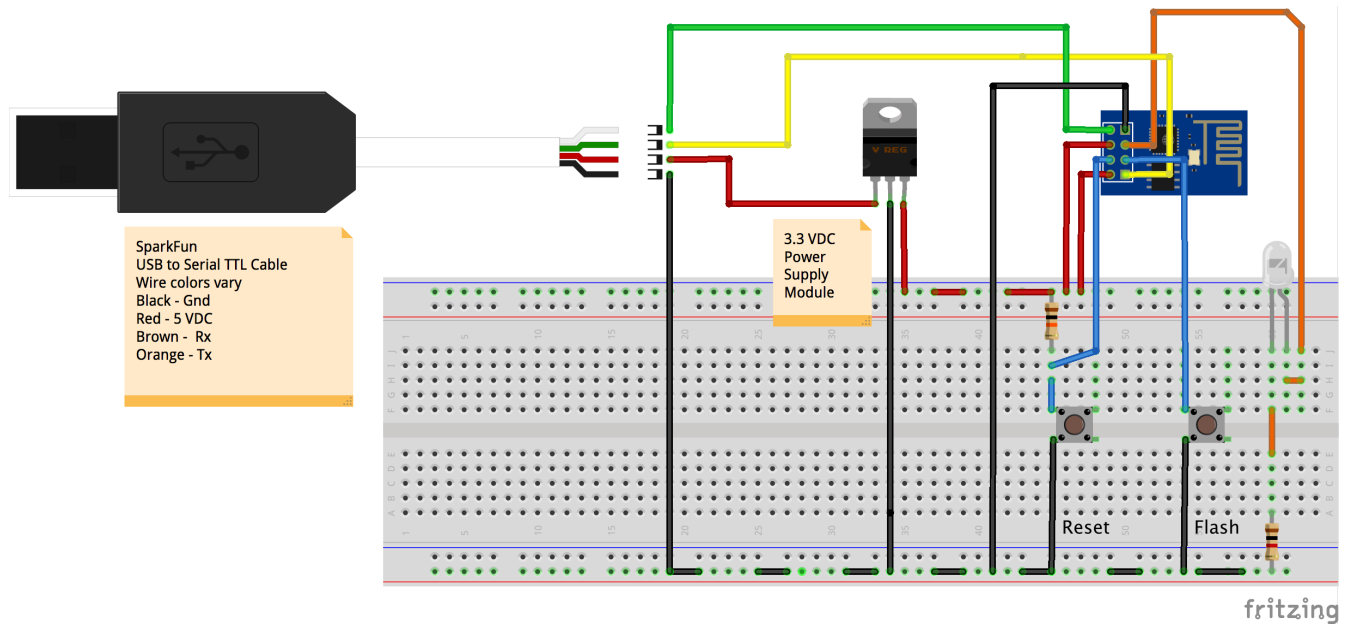


Figure Two
Prototype Hardware using a USB to 3.3 VDC TTL Serial Cable
Includes LED for the Teleduino Demo



Chapter Two - NTP Clock

Introduction

Building digital clocks is not the sexiest of DIY projects yet many people do so each year. People build these clocks in a wide variety of shapes and sizes including the weird one I designed and wrote about in the March 2014 Nuts and Volts issue called, “A Unique LED Clock”. Most home brewed digital clocks use an Arduino or other micro controller coupled to a real time clock (RTC) chip that provides the time keeping machinery and in some cases battery backup facilities. It is up to the user to set the clock to the correct time and if good quality components were used in the clock, time keeping accuracy can be pretty good. Unless the RTC chip's oscillator is temperature controlled, however, accuracy will drift over time forcing the user to perform periodic corrections to the time. Also, some RTC chips don't handle daylight savings time (DST) so it is up to the user to reset their clocks twice a year in regions that use daylight savings time.

To overcome the problems with manual time and date setting and time drift, many so called “Atomic Clocks” or “Radio-Controlled Clocks” have appeared on the market. These come in every conceivable shape and size as well. These clocks listen for WWV radio transmissions from Fort Collins, Colorado and synchronize their time keeping mechanisms to the atomic clock reference used for these transmissions thus guaranteeing their time keeping accuracy. These clocks typically require the user to select their timezone but other than that do not offer any controls for manually setting the time.

The clock mechanisms in personal computers work differently still. Personal computer usually sync their RTC using an Internet standard called Network Time Protocol or NTP. According to Wikipedia:

“NTP is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. ...

NTP is intended to synchronize all participating computers to within a few milliseconds of Coordinated Universal Time (UTC). ...

NTP can usually maintain time to within tens of milliseconds over the public Internet, and can achieve better than one millisecond accuracy in local area networks under ideal conditions. “

Basing a digital clock design on NTP requires access to the Internet which can be expensive to implement but allows for a very simple clock design for a couple of reasons. First, no battery backup circuitry is required to maintain the time setting. If clock power is lost, the connection to the Internet will automatically be re-established once power is restored and the clock will automatically set itself to the correct time. Second, no controls for manually setting the time are typically necessary because time and date setting are automatic.

The ESP8266 family of devices makes inexpensive access to the Internet a non issue so it is natural to use these devices in an NTP clock. Current readers of Nuts and Volts may remember my two previous article about using the amazing ESP8266 devices:

Chapter Two - NTP Clock

1. "Meet the ESP8266: A Tiny, WiFi Enabled, Arduino Compatible Micro Controller" in the October 2015 issue and the
2. "Thinking of You" article in the November 2015 issue.

To refresh your memory, all members of the ESP8266 device family share some basic characteristics including:

- 802.11 b / g / n
- Wi-Fi Direct (P2P), soft-AP
- Built-in TCP / IP protocol stack
- 802.11b mode + 19.5dBm output power
- Built-in temperature sensor
- Supports antenna diversity
- Off leakage current is less than 10uA
- Built-in low-power 32-bit CPU which can double as an application processor
- SDIO 2.0, SPI, UART , ADC, EEPROM
- Standby power consumption of less than 1.0mW (DTIM3)

In other words, the ESP8266 family of modules features low power consumption, high RF power output and are capable of supporting all of the current 802.11 standards required for WiFi connectivity. In addition, they support many industry standard hardware interfaces and can function as the application processor in many designs as they do in this one. The ESP8266 is a 3.3 VDC part.

Two things make using these parts even sweeter. First, many ESP8266 modules can be purchased for under \$10 in single unit quantities. Second, these modules can be programmed in the Arduino environment with the IDE so Arduino developers don't have to learn yet another programming environment to use them.

In this article I present the design and implementation of a very simple NTP digital clock based on the ESP8266 that drives a small LCD display. In actuality I used an ESP8266 variant called a NodeMCU LUA Amica in this design as it has lots of digital I/O pins available making interfacing to the display trivial.

This clock has a single pushbutton switch that, if configured for daylight saving time operation (more on this later), allows the user to put the clock into and take the clock out of daylight saving time (DST) mode.

Designing this digital clock allowed me to experiment with aspects of the ESP8266 that I had not used before including the hardware SPI interface used to run the LCD display and the onboard EEPROM for storage and retrieval of the DST state indicator.

Hardware

The hardware parts list below shows the parts required to build one of these NTP clocks and where to

get them. As you can see there isn't much to it.

Part	Source
NodeMCU LUA Amica R2 Module	Electrodragon.com
1.8" TFT SPI LCD Display	Adafruit.com - Product ID: 358
Pushbutton Switch SPST	Radio Shark or anywhere else
USB Cable - USB A to USB Micro B	Radio Shack or anywhere else
USB Power Supply capable of at least 1 amp @ 5 volts	Radio Shack or anywhere else

Figure One shows a Fritzing connection diagram/schematic for the NTP clock. Figure Two shows the design wired up and working on a breadboard. NOTE: there isn't a wire color correlation between Figures One and Two. As shown above, the clock is powered via a USB cable and a USB power supply module.

The wire by wire connections are shown below because they might not be clear from the Fritzing diagram.

NodeMCU Amica Pin	Adafruit 1.8" Display Connection	DST Pushbutton SPST Switch
D1 (GPIO 5)		SW1
D3 (GPIO 0)	LITE	
D4 (GPIO 2)	D/C	
D5	SCK	
D7	MOSI	
D8 (GPIO 15)	TFT_CS	
3V3	VCC	
GND	Gnd	SW2

The GPIO designations are shown above as that is how these digital I/O lines are referred to in the Arduino code.

The Adafruit LCD display also has a micro SD memory card connector and interface which can be used with the ESP8266 although they were not needed for this project.

Software

The software for the ESP8266 NTP clock was developed using the Arduino IDE. See my previous articles or the *Resources* section for how to set-up the Arduino IDE on your computer for targeting ESP8266 type devices. Make sure to select “*NodeMCU 1.0 (ESP-12E Module)*” as the board type in the tools menu.

The ESP8266 NTP clock software should be available in the code associated with this document. The sketch is called: *ESP8266_NTPClock.ino*. To use this software copy/move the ESP8266_NTPClock directory from the code directory into your Arduino directory.

Whereas the hardware for this clock borders on the trivial, the software/firmware for the clock is a bit more involved and complex. The seven files which make up the code are described in the table below:

File	Description
ESP8266NTPClock.ino	Main program. Initializes the hardware, logs into the local WiFi network and then installs the NTP code as the time provider. It then manages the update of the clock on the display.
ESP8266_ST7735.cpp	LCD driver code specific to the Adafruit 1.8” (blacktab) display utilizing the hardware SPI interface of the ESP8266.
ESP8266_ST7735.h	Header file for the LCD driver code above
Icons.h	Data for the WiFi, Sun and Moon icons. Data is in xbm format.
NTP.h	Functions for sending UDP packets to NTP servers and retrieving the GMT time and converting it to local time.
TextGraphicsFunctions.h	Misc functions for formatting the time data for display on the LCD.
Misc.h	Code for reading and writing the ESP8266's EEPROM

The ESP8266_ST7735 LCD driver code was adapted from the Adafruit ST7735 library to use the hardware SPI interface on the ESP8266. If you want to use a different LCD display you will have to find/develop an appropriate driver yourself.

In addition to the files above, the following Arduino libraries are also required:

Library	Source
Adafruit_GFX	https://github.com/adafruit/Adafruit-GFX-Library

Library	Source
Time	https://github.com/PaulStoffregen/Time

The version of these libraries I used to develop the NTP clock are included in the zip file for this article. Remember libraries must be installed in the *arduino/libraries* directory on your development computer and the Arduino IDE must be restarted to recognize them.

Most of the code that makes up the NTP clock is straight forward and will be easy to understand. The NTP code in the file *NTP.h*, is more complex however. To retrieve the time one must:

Get an IP address of a time server from the pool of *time.nist.gov* servers using the *hostByName* function as shown below. If you were to monitor the *timeServerIP* address during the operation of the clock you would see that the time requests rotate in a round robin fashion between the servers in the *time.nist.gov* pool.

```
// Get a server from the pool
WiFi.hostByName("time.nist.gov", timeServerIP);
```

Once you identify a server to make a request of, you must create a UDP packet configured with proper values and then send it the packet. See <https://tools.ietf.org/html/rfc5905#section-7.3> for an explanation of the fields in the UDP request packet. The *sendNTPPacket* function below does this.

```
// Send an NTP request to the time server at the given address
unsigned long sendNTPpacket(IPAddress& address) {

    // Set all bytes in the buffer to 0
    memset(packetBuffer, 0, NTP_PACKET_SIZE);

    // Initialize values needed to form NTP request
    packetBuffer[0] = 0b11100011; // LI, Version, Mode
    packetBuffer[1] = 0;         // Stratum, or type of clock
    packetBuffer[2] = 6;         // Polling Interval
    packetBuffer[3] = 0xEC;      // Peer Clock Precision
    // 8 bytes of zero for Root Delay & Root Dispersion
    packetBuffer[12] = 49;
    packetBuffer[13] = 0x4E;
    packetBuffer[14] = 49;
    packetBuffer[15] = 52;

    // All NTP fields have been given values, now
    // you can send a packet requesting a timestamp:
    udp.beginPacket(address, 123); // NTP requests are to port 123
    udp.write(packetBuffer, NTP_PACKET_SIZE);
    udp.endPacket();
}
```

Once the UDP packet is sent, you wait for a response and in that response will be a time stamp (the four bytes starting at the 40th byte of the response) indicating the time the packet was sent. The units of this time stamp is seconds since 1900 and is a very large number. This value gets converted to Unix time which is seconds since Jan 1, 1970 by the subtraction of the number of seconds between 1900 and 1970. This number is then further modified by timezone correction. This corrected value is what is used by the Time library and converted to the current time and date displayed by this clock. The *getNTPTime* function pulls this all together.

Chapter Two - NTP Clock

```
// NTP Time Provider Code
time_t getNTPTime() {

    int attempts = 10;

    // Try multiple attempts to return the NTP time
    while (attempts--> 0) {

        // Get a server from the pool
        WiFi.hostByName(ntpServerName, timeServerIP);
        Serial.print("Time server IP address: ");
        Serial.println(timeServerIP);

        while (udp.parsePacket() > 0); // Discard any previously received packets

        Serial.println("Transmitted NTP Request");
        sendNTPpacket(timeServerIP);

        uint32_t beginWait = millis();
        while (millis() - beginWait < 1500) {
            int size = udp.parsePacket();
            if (size >= NTP_PACKET_SIZE) {
                Serial.println("Received NTP Response");
                udp.read(packetBuffer, NTP_PACKET_SIZE); // Read packet into the buffer
                unsigned long secsSince1900;

                // Convert four bytes starting at location 40 to a long integer
                secsSince1900 = (unsigned long) packetBuffer[40] << 24;
                secsSince1900 |= (unsigned long) packetBuffer[41] << 16;
                secsSince1900 |= (unsigned long) packetBuffer[42] << 8;
                secsSince1900 |= (unsigned long) packetBuffer[43];

                Serial.println("Got the time");

                return secsSince1900 - 2208988800UL + realTimeZoneOffset * SECS_PER_HOUR;
            }
            delay(10);
        }
        Serial.println("Retrying NTP request");
        delay(4000);
    }
    Serial.println("No NTP Response");
    return 0;
}
```

User Configuration of the NTP Clock Software

The NTP clock's software must be configured before the clock will work correctly. All user configuration items are found in the *ESP8266_NTPClock.ino* file. Please locate the following text in that file:

```
// *****
// Start of user configuration items
// *****

// Set your WiFi login credentials
#define WIFI_SSID "xxxxxxx"
#define WIFI_PASS "xxxxxxxxxxxxx"

#define TIMEZONE_OFFSET -7 // Set your timezone offset (-7 is mountain time)
#define USE_DST true // Set to false to disable DST mode
#define HOUR_FORMAT_12 true // Set to false for 24 hour time mode
```

```
// *****  
// End of user configuration items  
// *****
```

First and most importantly you must modify the code with the SSID and Password of your WiFi network otherwise the clock won't be able to access the Internet and by extension the NTP servers that provide the time. Next, you must set the correct timezone offset for your location. Timezone offsets can be found here:

https://en.wikipedia.org/wiki/List_of_UTC_time_offsets

Then you must decide if your clock will use daylight saving time or not and whether it will operate in 12 or 24 hour format. `USE_DST` must be set true if your clock will use daylight savings time whether or not DST is currently in effect. Set `HOUR_FORMAT_12` true to run your clock in 12 hour format otherwise it will operate in 24 hour time format.

The code can be compiled and uploaded to the NodeMCU device once the configuration data is set and all of the required libraries have been installed in the Arduino environment.

NTP Clock Operation

The clock should start immediately once the software is uploaded. Figure Three shows the clock's display while a connection is being made to the local WiFi network. If this screen doesn't change to the clock display of Figure Four it means there were problems logging into the WiFi network. If this is the case go back and verify the `WIFI_SSID` and `WIFI_PASS` entries in the code and that the WiFi network is working.

As mentioned, the WiFi login display should change to the clock display of Figure Four once a WiFi connection is established. If the clock is not configured for DST mode, you are done. The clock should run as long as power is applied and it will sync its time to an NTP time server every five minutes, making the clock extremely accurate.

If the clock is configured for DST operation (`USE_DST` is true), it is up to the user to put the clock in DST mode if DST is currently in effect (mid April through November in the US). The clock doesn't default to DST mode so the user must push the DST button until the DST string is displayed in the upper right corner of the clock. You'll notice the displayed time changes when DST mode is engaged. Pressing the DST button again toggles the clock out of DST mode.

The clock will continue to run as long as power is applied. If the Internet connection is dropped, the clock will maintain the time itself. If WiFi goes down but the clock remains powered, the clock will need to be rebooted once the network issue is resolved so that NTP time syncing can be restarted. If power is lost to both the clock and the WiFi network, the clock will reboot and wait for the network to come back up and will then reconnect automatically.

Chapter Two - NTP Clock

While the clock is operational, the time and date will update once a minute. During the day, a sun icon will be displayed in the upper left corner. The sun icon will be replaced by a moon icon after about 8 PM in the evening.

As daylight savings time comes and goes, the user will be required to inform the clock by pressing the DST pushbutton which toggles the DST mode on and off. Other than that there are no other ongoing operational maintenance issues required by the user.

As a final note, the DST enabled state is written to the EEPROM in the ESP8266 every time the DST pushbutton is pressed. This was necessary to bring back the correct DST state if power to the clock was lost and then regained. See the file *Misc.h* for the EEPROM read/write code.

Resources

The following resources may be of use:

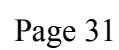
Information about NTP can be found all over the Internet. See <http://www.ntp.org/> for detailed information.

Information on WWV time broadcasts can be found at:
[https://en.wikipedia.org/wiki/WWV_\(radio_station\)](https://en.wikipedia.org/wiki/WWV_(radio_station))

Information about programming the ESP8266 in the Arduino environment can be found at:
github.com/esp8266/Arduino and in my two articles mentioned previously.

Information about the NodeMCU Amica can be found at: www.electrodragon.com/product/nodemcu-lua-amica-r2-esp8266-wifi-board/.

Information about the Adafruit 1.8" TFT SPI LCD display can be found at:
<http://www.adafruit.com/products/358>.



Chapter Two - NTP Clock

Figure Two
The ESP8266 NTP Clock Breadboard

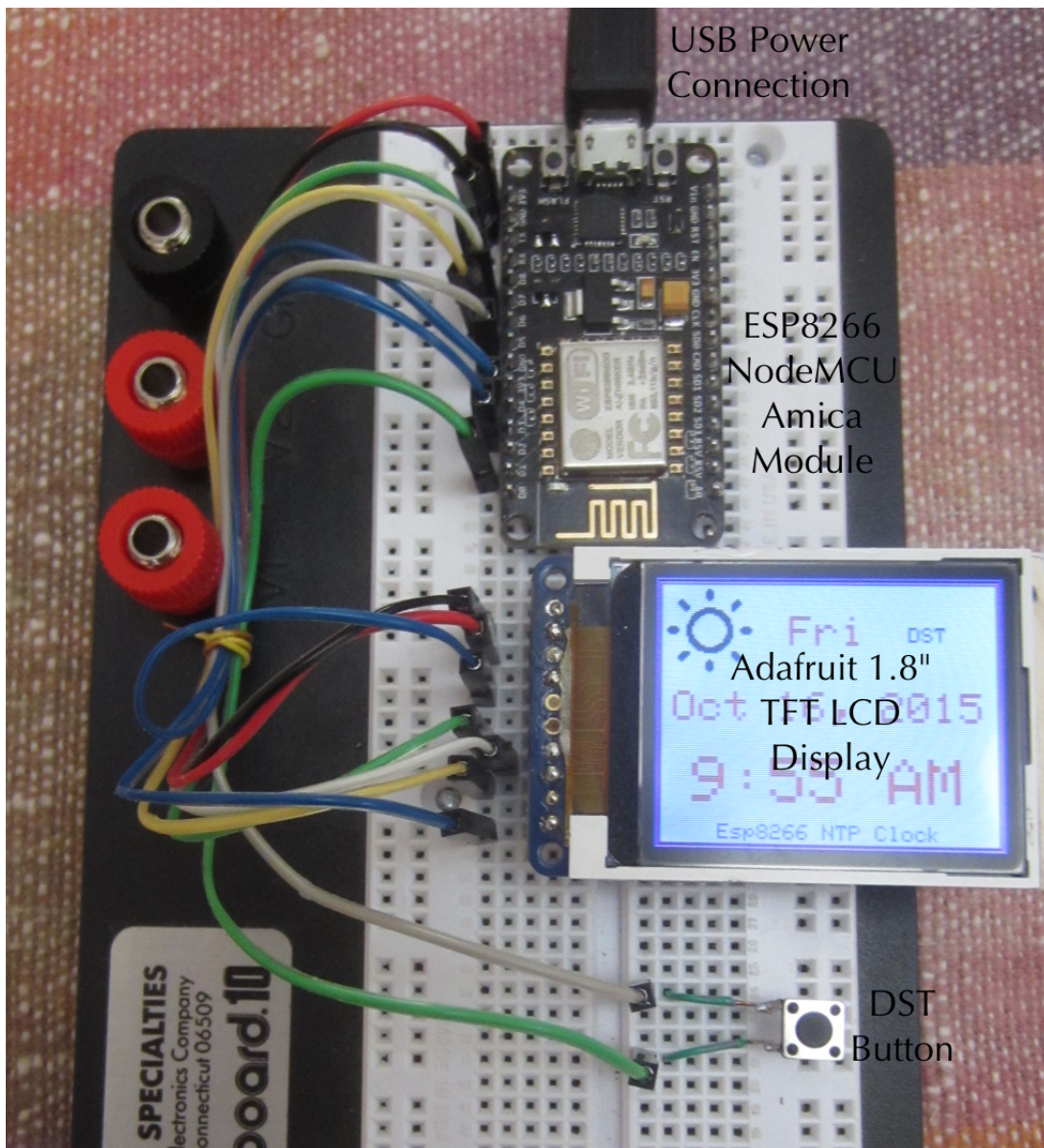
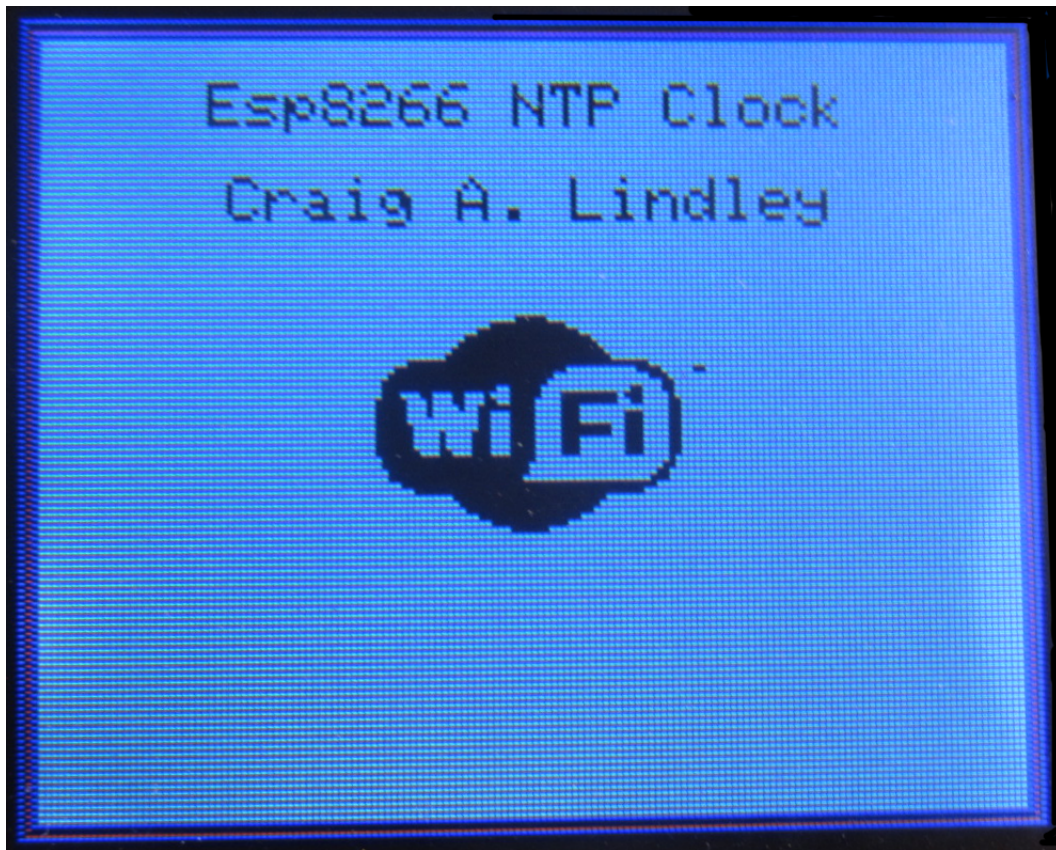


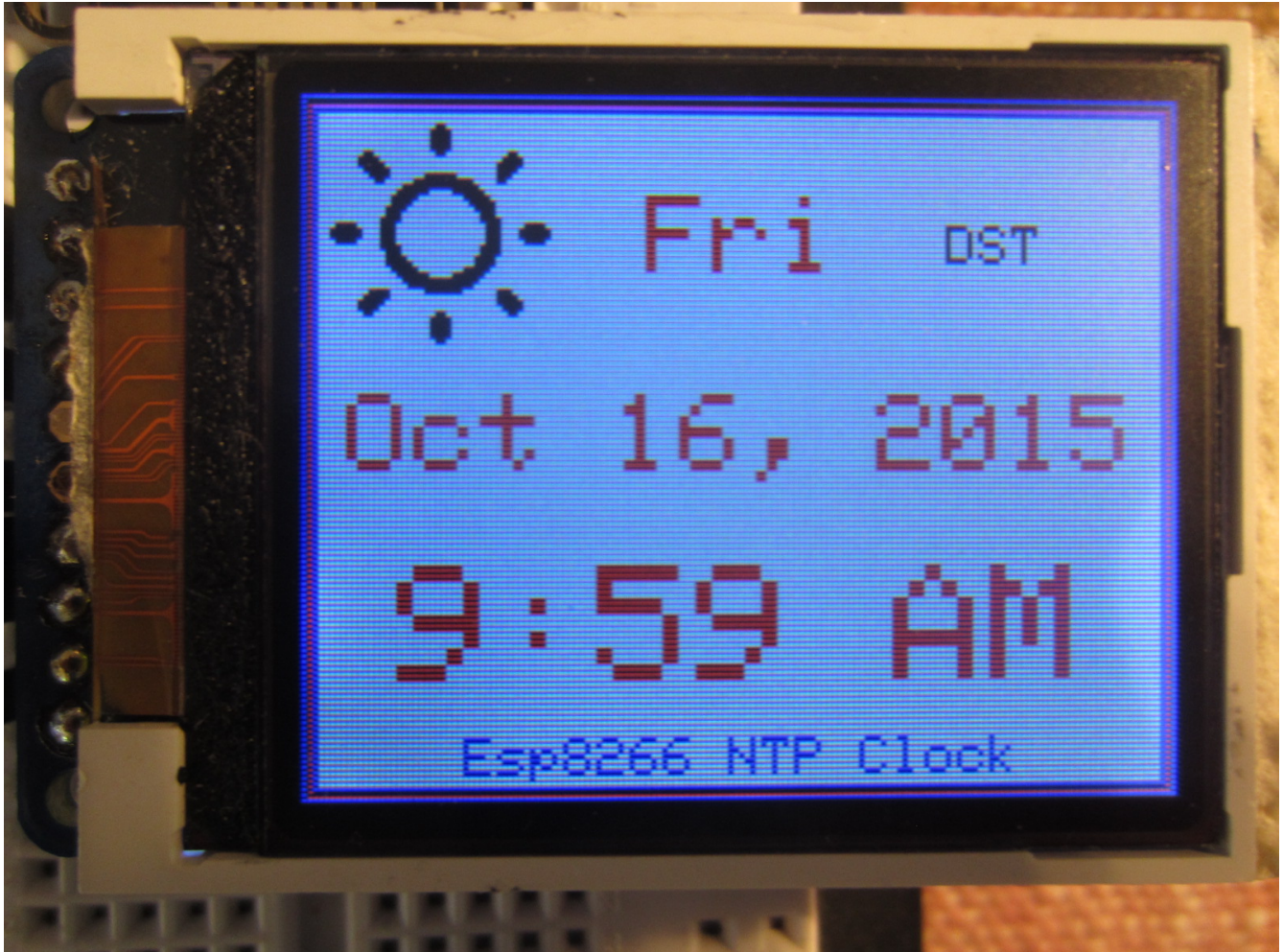
Figure Three
Initial WiFi Connection Display



Chapter Two - NTP Clock

Figure Four
Typical Clock Display

Note: daylight saving time (DST) mode is on and because the Sun icon is being displayed it is daytime.



Chapter Three - Weather Clock

Introduction

Readers of Nuts and Volts may recall my three previous article about using the amazing ESP8266 family of devices. They are:

1. "Meet the ESP8266: A Tiny, WiFi Enabled, Arduino Compatible Micro Controller" in the October 2015 issue.
2. "Thinking of You" article in the November 2015 issue.
3. "ESP8266 NTP Clock" article in the June 2016 issue.

The more I use these devices, the more I am impressed with their value proposition, capabilities and robustness. After writing the ESP8266 NTP Clock article where I coupled a NodeMCU Amica module (which contains an ESP8266-12 device) with an Adafruit 1.8" TFT LCD display I started to look around for other applications that I could use this same hardware for. I came up with two ideas.

1. Building a mini weather station that uses weather data available on the Internet (in this case from myweather2.com) for any location in the world and displaying it on the LCD display.
2. An RSS Feed reader that can display headlines from various news sources across the Internet on the LCD display.

In this article I will present the implementation of the first idea in what I call my Weather Clock which combines the display of localized weather conditions with the auto setting NTP clock from my previous article. So with extremely simple hardware with a low parts count you can get current and forecasted weather conditions for your specific location and have a clock that never needs to be manually set. Pretty sweet don't you think ?

I'll save the RSS Feed reader (which is really cool as well) for a future article. This also runs on exactly the same hardware as the Weather Clock and the NTP Clock.

Building a weather station with a micro-controller is hardly new news but most of the implementations I have seen used a PC or Raspberry Pi to access the Internet for the weather data and then messaged the data into a small enough package that it could be transferred to the micro-controller weather station for display. In other words the micro-controller based system was just a display for the previously digested weather data.

I took this as somewhat of a challenge to see if I could combine both weather data acquisition and display using a single ESP8266 device and, while I was at it, see if I could include the NTP clock functionality as well.

Chapter Three - Weather Clock

I'm pleased to say that I was able to pull this off. If you have ever wanted to build a mini weather station / clock for your home or business I don't believe you will find a simpler or cheaper solution than the one presented here.

Hardware

As mentioned, the Weather Clock uses the same hardware as used in my ESP8266 NTP Clock. To save you from going back and (re)reading the previous article, the hardware information is repeated here starting with the minimalist parts list.

Part	Source
NodeMCU LUA Amica R2 Module	Electrodragon.com
1.8" TFT SPI LCD Display (blacktab)	Adafruit.com - Product ID: 358
Pushbutton Switch SPST	Radio Shack or anywhere else
USB Cable - USB A to USB Micro B	Radio Shack or anywhere else
USB Power Supply capable of at least 1 amp @ 5 volts	Radio Shack or anywhere else
Hook up wire and breadboard	Radio Shack or anywhere else

Figure One shows a Fritzing connection diagram/schematic for the Weather Clock. Figure Two shows the design wired up and working on a breadboard. NOTE: there isn't a wire color correlation between Figures One and Two. As shown above, the Weather Clock is powered via a USB cable and a USB power supply module.

The wire by wire connections are shown below because they might not be clear from the Fritzing diagram.

NodeMCU Amica Pin	Adafruit 1.8" Display Connection	DST Pushbutton SPST Switch
D1 (GPIO 5)		SW1
D3 (GPIO 0)	LITE	
D4 (GPIO 2)	D/C	
D5	SCK	
D7	MOSI	
D8 (GPIO 15)	TFT_CS	
3V3	VCC	
GND	Gnd	SW2

The GPIO designations are shown above as that is how these digital I/O lines are referred to in the

Arduino code.

The Adafruit LCD display also has a micro SD memory card connector and interface which can be used with the ESP8266 although they were not needed for this project.

Software

The software for the ESP8266 Weather Clock was developed using the Arduino IDE. See my previous articles and/or the *Resources* section for how to set-up the Arduino IDE on your computer for targeting ESP8266 type devices. Make sure to select “*NodeMCU 1.0 (ESP-12E Module)*” as the board type in the tools menu.

The ESP8266 Weather Clock software should be available in the code associated with this document. The sketch is called *ESP8266_WeatherClock.ino*. To use this software, copy/move the ESP8266_WeatherClock directory from the code directory into your Arduino directory.

While the hardware is about as simple as possible, the software is quiet complex and is made up of the following files:

File	Description
DisplayPages.h	Code for each of the seven display pages
ESP8266_ST7735.cpp	LCD driver code specific to the Adafruit 1.8” (blacktab) display utilizing the hardware SPI interface of the ESP8266.
ESP8266_ST7735.h	Header file for the LCD driver code above
ESP8266_WeatherClock.ino	Main program. Initializes the hardware, logs into the local WiFi network and then installs the NTP code as the time provider. It then manages the display of the display pages on the LCD.
Icons.h	Data for the WiFi, Sun and Moon icons. Data is in xbm format.
Misc.h	Code for reading and writing the ESP8266's EEPROM
NTP.h	Functions for sending UDP packets to NTP servers and retrieving the GMT time and converting it to local time.
TGFunctions.h	Misc functions for formatting text and graphical data for display on the LCD.
Weather.cpp	Weather class for sending weather data requests to myweather2.com, for retrieving the JSON data stream returned and then parsing the data to

Chapter Three - Weather Clock

File	Description
	extract the pertinent weather attributes for display by the various display pages.
Weather.h	Header file for the Weather class above

In addition to the files above, the following Arduino libraries are also required:

Library	Function	Source
Adafruit_GFX NOTE: this library had to be modified for use with the ESP8266 so it is not the stock library.	Text and graphics functions for the LCD display driver	https://github.com/adafruit/Adafruit-GFX-Library
Time	Updated and improved version of the Arduino library	https://github.com/PaulStoffregen/Time
ArduinoJson	JSON parser	https://github.com/bblanchon/ArduinoJson

The version of these libraries I used to develop the Weather Clock are also included in the code directory for this article. Remember libraries must be installed in the *arduino/libraries* directory on your development computer and the Arduino IDE must be restarted to recognize them.

User Configuration of the Weather Clock Software

The Weather Clock's software must be configured before it will work correctly. All user configuration items are found in the *ESP8266_WeatherClock.ino* file. Please locate the following text in that file:

```
// *****
// Start of user configuration items
// *****

// Set your WiFi login credentials
const char * WIFI_SSID = "xxxxxxxxxx";
const char * WIFI_PASS = "xxxxxxxxxxxx";

const int    TIMEZONE_OFFSET = -7;      // Set your timezone offset (-7 is mountain time)
const bool   USE_DST = true;           // Set to false to disable DST mode
const bool   HOUR_FORMAT_12 = true;    // Set to false for 24 hour time mode

const char * API_KEY = "yyyyyyy";      // Key/Access code from myweather2.com
const char * LOCATION_STRING = "zzzzzz"; // Location indicator

// *****
// End of user configuration items
// *****
```

First and most importantly you must modify the code with the SSID and Password of your WiFi

network otherwise the Weather Clock won't be able to access the Internet and by extension the weather data feed from myweather2.com nor the NTP servers that provide the time. Next, you must set the timezone offset for your location to make the clock display the correct time. Timezone offsets can be found here:

https://en.wikipedia.org/wiki/List_of_UTC_time_offsets

Then you must indicate whether your clock will use daylight saving time or not and whether it will operate in 12 or 24 hour format. `USE_DST` must be set true if your clock will use daylight savings time whether or not DST is currently in effect. Set `HOUR_FORMAT_12` true to run your clock in 12 hour format otherwise it will operate in 24 hour time format.

You must get an api key from myweather2.com (which they refer to as the Unique Access Code) to access their service and retrieve weather data from them. To get a key/code go to:

<http://www.myweather2.com/ilogin.aspx>

and fill out their registration form. After you have created an account you need to login and go to your *Developer Zone* page and you will see the key/code they have assigned you. This key/code must be transferred to the `API_KEY` entry in the configuration data shown above. Note, only one key/code is available per email address.

You must also tell the myweather2.com service the location you want the weather data for. This is what the `LOCATION_STRING` in the configuration data does. This string has three possible formats:

1. A UK Postcode
2. A US zip code
3. A latitude,longitude

For my Weather Clock I used my zip code.

The code can be compiled and uploaded to the NodeMCU Amica device once the configuration data is set and all of the required libraries have been installed in the Arduino environment.

Weather Clock Operation

Figures Three through Nine show the Weather Clock in operation. Each of these images show what I refer to as a display page. Each display page shows different information, shown below, but all pages have a series of seven small circles at the bottom of the page to indicate which display page is currently being shown.

Chapter Three - Weather Clock

Display Page Number	Information Displayed
1	WiFi logo and Credits
2	Current Weather Conditions
3	Day Conditions Forecast
4	Night Conditions Forecast
5	Next Day Conditions Forecast
6	Next Night Conditions Forecast
7	NTP Time and Data Display

When the Weather Clock software first starts there is (usually) a short delay while three things happen. First, the Weather Clock logs into the local WiFi network. Next, the NTP clock code initializes and makes a request over the Internet to retrieve the time from a NTP time server. Finally, weather data is requested and retrieved from myweather2.com. If all is well, the first display page with the WiFi icon and credits is selected for display (Figure Three) and after a programmable length of time, the other six display pages are sequentially displayed. After display page seven, the NTP Clock, is displayed the sequence repeats starting with display page one.

By default, time data is retrieved every five minutes; weather data is retrieved every 15 minutes; display pages are changed every 12 seconds. Of course each of these time intervals can be changed in the software. I should point out that the acquisition of weather and time data is completely disjoint from its display. That is, these two processes happen completely independently of each other.

The operation of the Weather Clock can best be understood by examining the code in the file *ESP8266WeatherClock.ino*. A Finite State Machine (FSM) contained in the *loop()* function orchestrates everything. Technically a FSM is

“A model of a computational system, consisting of a set of states, a set of possible inputs, and a rule to map each state to another state, or to itself, for any of the possible inputs”.

Sounds daunting but don't let the definition scare you, the operation is really quite simple.

The Weather Clock's FSM is defined by this series of states:

INIT, CHECK_EVENTS, ACQUIRE_DATA, ADVANCE_DISPLAY, UPDATE_DST_STATUS

which will each be described shortly. In addition, there is a variable called *state* which tells the state machine which state it is currently in. Transitions between the states happen when certain events or inputs occur. The FSM is guaranteed to be in one of the defined states every time through the *loop()* function.

The INIT state is the initialization state for the state machine and it is only entered once when the

ESP8266 device is first powered up. In this state, the display page number variable is set to one; display page one is displayed on the LCD and the *dataAcquisitionCount* and the *displayAdvanceCount* variables are initialized to a time in the future when new weather data is to be acquired and when the next display page is to be shown. The CHECK_EVENTS state is then selected for the next iteration of the *loop()* function.

The FSM will stay in the CHECK_EVENTS state until one of the following things happen:

1. Activity is detected on the DST pushbutton switch
2. The weather data acquisition count has expired and new weather data needs to be acquired.
3. It is time to advance to the next display page.

The state of the FSM will change to UPDATE_DST_STATUS if number one occurs; to ACQUIRE_DATA if number two occurs and to ADVANCE_DISPLAY if number three occurs.

In the UPDATE_DST_STATUS state the Daylight Savings Time or DST status is toggled. If DST was on, it is turned off and if it was off, it is turned on. Every change to the DST state is stored in the ESP8266's EEPROM so that it survives power outages. Then the display page variable is set so that display page seven, the NTP Time and Date page, will be displayed the next time the ADVANCE_DISPLAY state is entered. Finally, the state is changed to ADVANCE_DISPLAY for the next trip through the loop.

In the ACQUIRE_DATA state the *retrieveWeatherData* function in the *Weather* class is called to acquire new weather data. A call to this function causes a whole chain of events to occur. First, an HTTP GET request is built up using the API_KEY and the LOCATION_STRING and it is sent to myweather2.com. The returned weather data in JSON format is stored line by line in a buffer for later processing. JSON or (Javascript Object Notation) is a text-based, human-readable data interchange format used for representing simple data structures and objects in Web browser-based code. See *Resources* below for information about JSON if you are interested.

After all the data has been retrieved, it is passed to the ArduinoJson parser which makes the various data items easily accessible. All of the weather attributes displayed on the five weather display pages are extracted from the returned JSON data.

Once new weather data has been acquired the *dataAcquisitionCount* variable is reinitialized to a time in the future when new weather data will again be required. Finally the state is changed to CHECK_EVENTS for the next pass through the loop.

The ADVANCE_DISPLAY state is entered when it is time to change the displayed page. In this state the LCD display is cleared and the frame which is common to all display pages is drawn around the perimeter of the display. The *displayPageNumber* variable is then incremented and wrapped around if necessary and the new display page is displayed. Finally *displayAdvanceCount* is reinitialized and the state variable is set back to CHECK_EVENTS for the next pass through the loop. In case it is not obvious, the ESP8266 spends most of its time spinning in the CHECK_EVENTS state only changing states occasionally when one of the three events occur.

Chapter Three - Weather Clock

Astute readers may be wondering how and when the NTP time data gets updated since there are no references to time update in the FSM. That is because time update is handled behind the scenes in the Time library and so doesn't need to be explicitly performed in the Weather Clock's FSM code.

Conclusions

To build a Weather Clock of your own, connect the Adafruit 1.8" LCD display to the NodeMCU Amica module and then connect a DST pushbutton switch. Next connect a USB cable from the NodeMCU Amica module to your development computer. After installing the required libraries bring up the Arduino IDE and load the software for this article. Edit the User Configuration data as described previously and then compile and upload the code. If you did everything correctly, you should have a fully functioning, stand alone Weather Clock of your own.

Resources

The following resources may be of use:

Information about myweather2.com's weather data feed can be found at:
www.myweather2.com/developer.

Information about JSON can be found at: <http://www.json.org/>

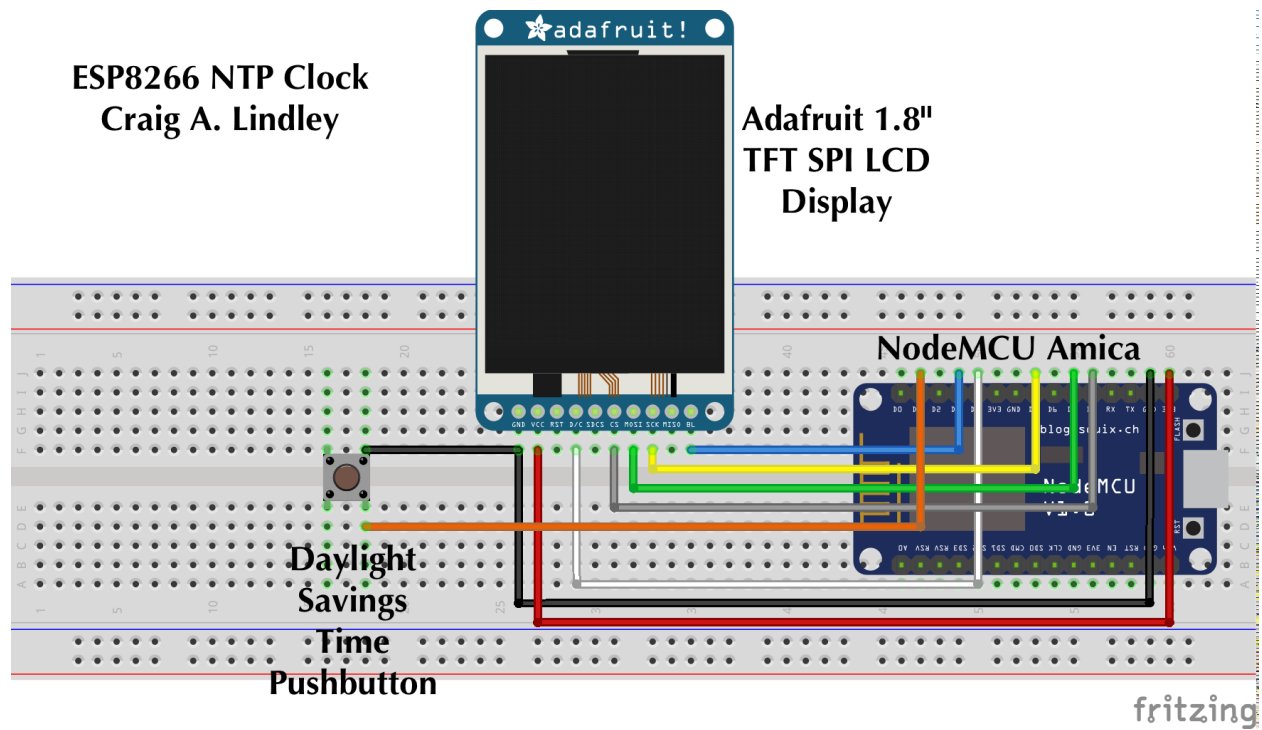
Information about NTP can be found at: <http://www.ntp.org/>.

Information about programming the ESP8266 in the Arduino environment can be found at:
github.com/esp8266/Arduino and in my three articles mentioned previously.

Information about the NodeMCU Amica can be found at: www.electrodragon.com/product/nodemcu-lua-amica-r2-esp8266-wifi-board/.

Information about the Adafruit 1.8" TFT SPI LCD display can be found at:
<http://www.adafruit.com/products/358>.

Figure One
Fritzing connection diagram/schematic



Chapter Three - Weather Clock

Figure Two
The design wired up and working on a breadboard

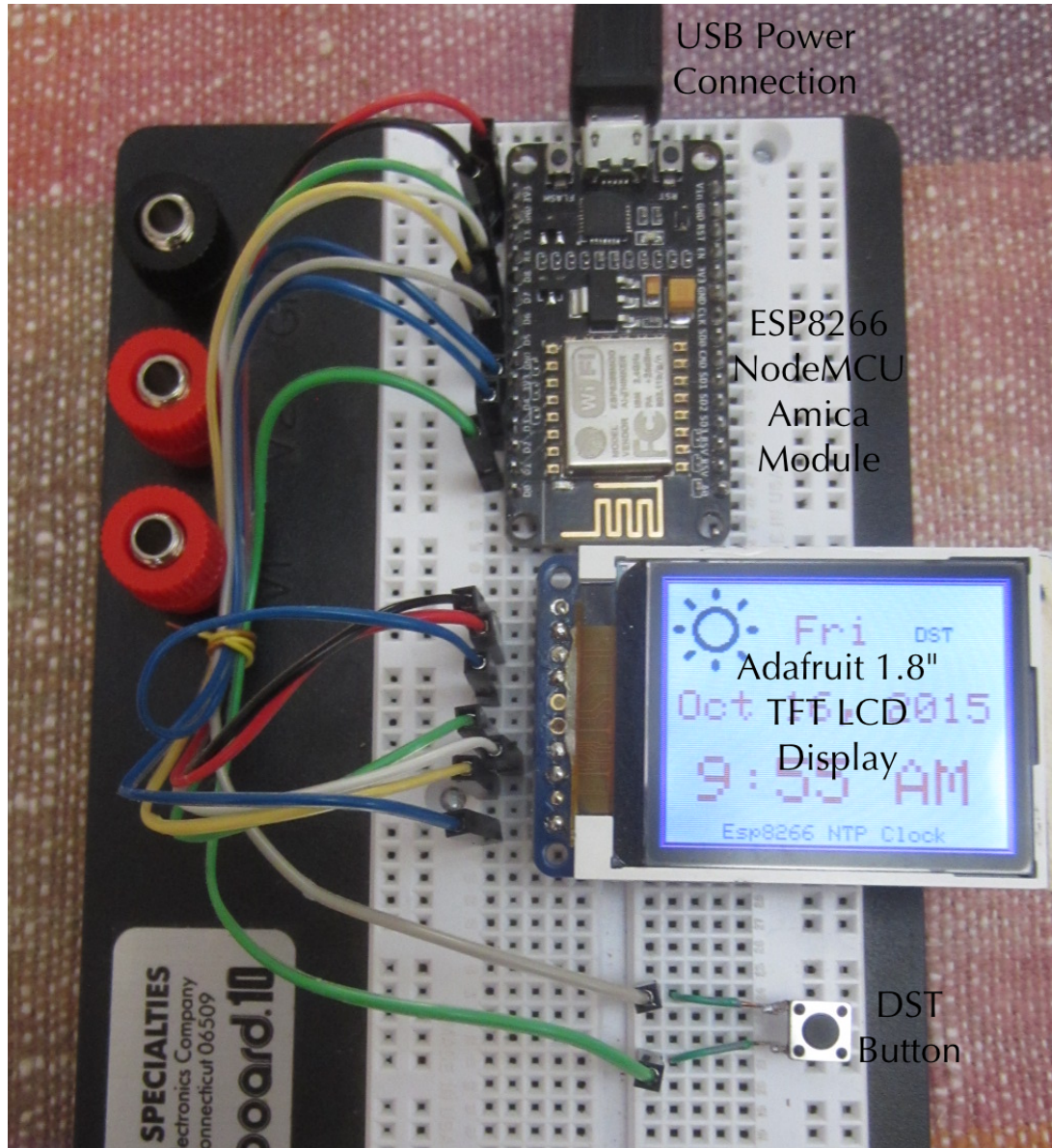


Figure Three
Display Page One
WiFi Login and Program Credits



Chapter Three - Weather Clock

Figure Four
Display Page Two
Current Weather Conditions

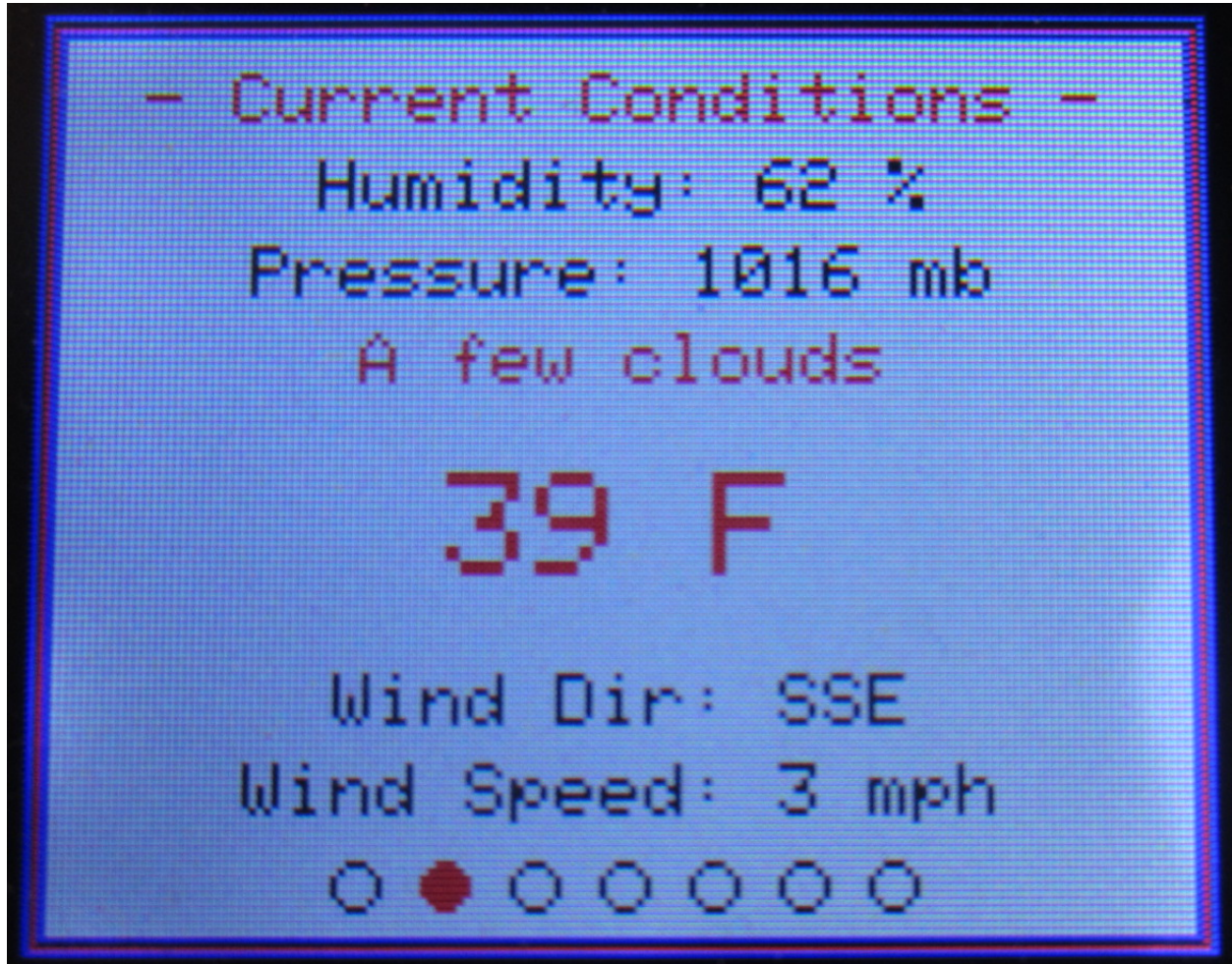
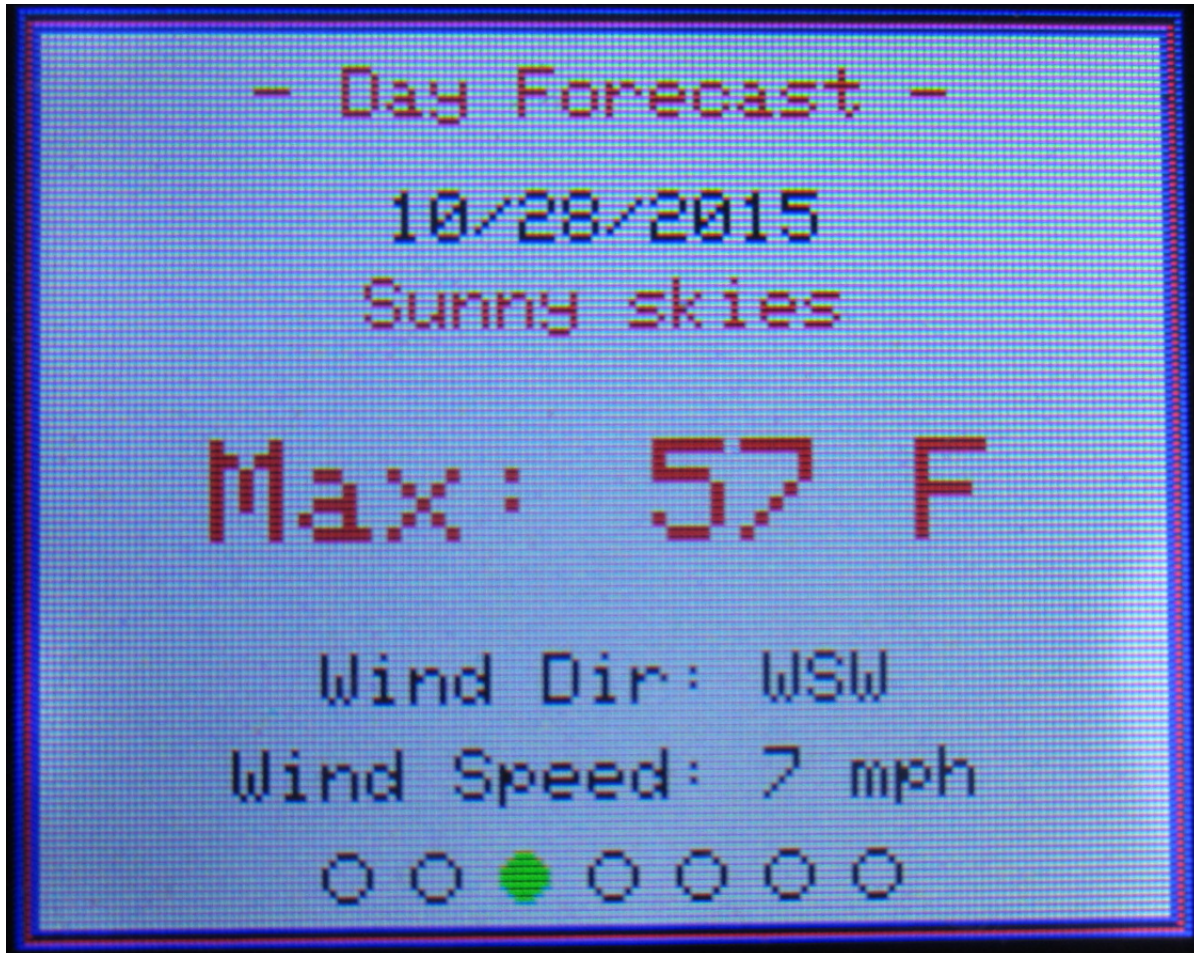


Figure Five
Display Page Three
Day Conditions Forecast



Chapter Three - Weather Clock

Figure Six
Display Page Four
Night Conditions Forecast

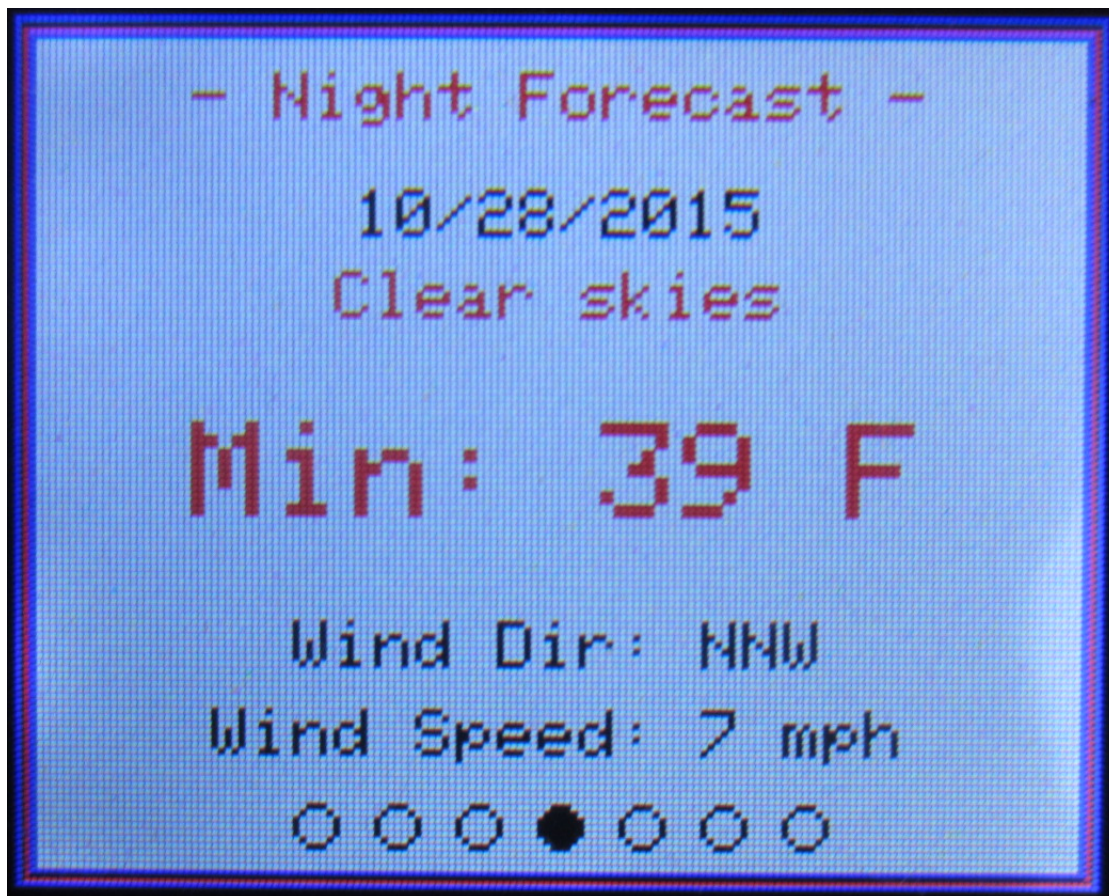
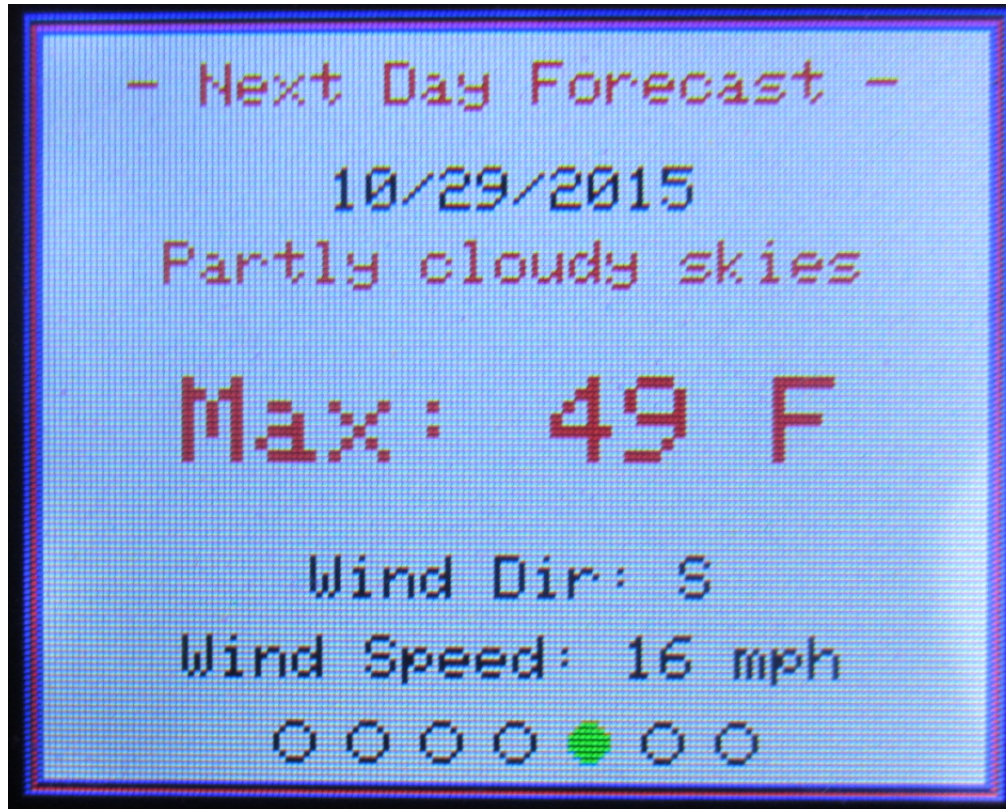


Figure Seven
Display Page Five
Next Day Forecast



Chapter Three - Weather Clock

Figure Eight
Display Page Six
Next Night Forecast

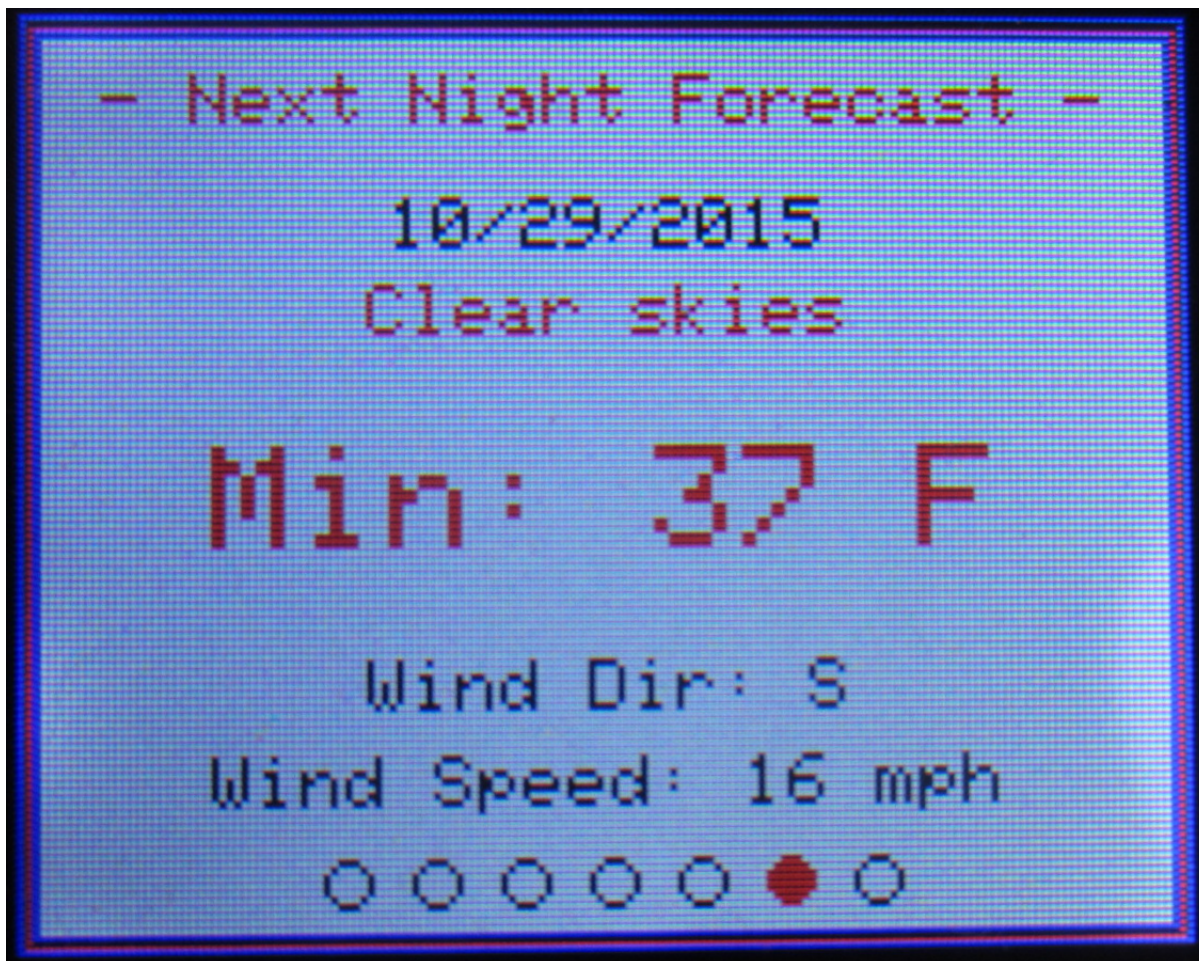
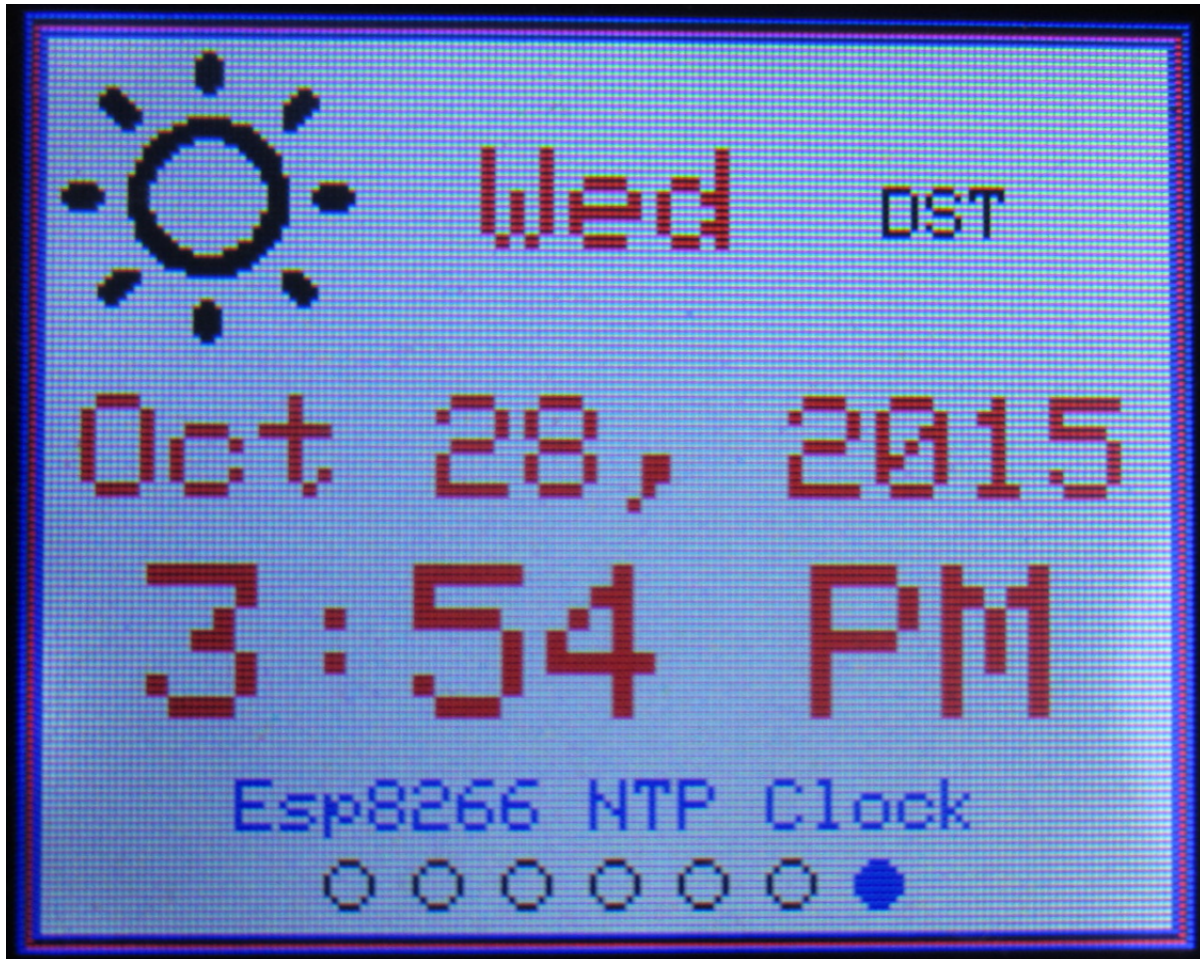


Figure Nine
Display Page Seven
NTP Time and Date Display



Chapter Four - World Clock

Introduction

What do most movie scenes of the stock market trading floor and most high end international hotels have in common? They both have world clocks that allow people to see the time and date in remote locations in the world. World clocks are important to business as they allow companies and/or individuals to have an idea of the time of day and date where their customers reside. There is nothing more embarrassing than calling an international client in the middle of the night because you figured the time difference incorrectly. World clocks can even be important to families that have members distributed across the US or the world so they know when would be a good time to call.

In order for world clocks to be useful, they must be accurate, they must show the time and date correctly for each location and they must take world wide daylight saving time into account. To assure the accuracy of this world clock design I have opted to again use Network Time Protocol or NTP as the provider of time information. NTP can provide time accurate to within a second to devices over the Internet which is good enough for our application here.

Basing a digital clock design on NTP requires access to the Internet which can be expensive to implement but allows for a very simple clock design for a couple of reasons. First, no battery backup circuitry is required to maintain the time setting. If clock power is lost, the connection to the Internet will automatically be re-established once power is restored and the clock will automatically set itself to the correct time. Second, no controls for manually setting the time and date are typically necessary because the time and date are set automatically.

The ESP8266 family of devices makes inexpensive access to the Internet possible so it is natural to use these devices in digital and world clock applications. Current readers of Nuts and Volts may remember my previous articles about using the amazing ESP8266 devices:

1. "Meet the ESP8266: A Tiny, WiFi Enabled, Arduino Compatible Micro Controller" in the October 2015 issue.
2. "Thinking of You" article in the November 2015 issue.
3. "ESP8266 NTP Clock" article in the June 2016 issue
4. "ESP8266 Weather Clock" article in the November 2016 issue
5. "ESP8266 RSS News Reader" article in the January 2017 issue

I should mention that the hardware used in articles 3, 4 and 5 above is exactly the same. That is, with one hardware setup (be it a breadboard or a PCB) but different software, you can have an NTP clock, an NTP clock and weather station combination or an RSS news reader. Now, with this article, the exact same hardware can function as a World Clock as well.

The World Clock design presented here has some rather interesting attributes including:

Chapter Four - World Clock

- It uses NTP so it sets itself automatically whenever it is powered up. There is no need for switches or buttons for setting the time or date. In fact there are no switches or buttons at all in this design.
- It uses a timezone library that automatically deals with daylight savings time (DST) world wide so there is no need to reset the clock when the time changes or to click a button to enter/exit daylight saving time.
- It currently has a round robin display of time and date for Sydney, Frankfurt, London, New York, Houston, Denver, Phoenix and Los Angeles and you can change these for other locations if you want to modify the code provided.
- The clock can run in 12 or 24 hour formats.
- It runs a finite state machine that will recover automatically if network connectivity is lost and then regained and it will also recover from power outages automatically as well.
- The design consists of two components only. A NodeMCU Amica ESP8266 module and a 1.8" LCD display. One cannot build many circuits simpler than this.
- This World Clock can be built for as little as \$22.

As soon as you power up this design, it accesses NTP time wirelessly via the Internet and then calculates and displays the time and date in the cities mentioned above. There is nothing to set or configure and it will continue to display the time and date until it is powered down. You might call this a no fuss world clock because there is nothing users have to do. When daylight saving time arrives in one or more of the target cities/timezones, it will automatically be taken into consideration. It even knows how to deal with places like Arizona which don't use DST.

The World Clock presented in this article is an adaption of the World Clock example program included with the Timezone library described later. My contribution is the use of NTP as the source of the time data, the state machines that allow the clock to recover from WiFi, Internet or power disruptions and the formatting and display of the time and date information on the LCD display.

Hardware

The hardware parts list below shows the components required to build a World Clock and where to get them. As you can see there isn't much to it.

Part	Source
NodeMCU LUA Amica R2 Module	Electrodragon.com
1.8" TFT SPI LCD Display	Adafruit.com - Product ID: 358

Part	Source
	or SainSmart.com 1.8 ST7735R TFT LCD Module with MicroSD SKU:20-011-920
USB Cable - USB A to USB Micro B	Radio Shack or anywhere else
USB Power Supply capable of at least 1 amp @ 5 volts	Radio Shack or anywhere else

Figure One shows a Fritzing diagram for the World Clock. Figure Two shows the design wired up and working on a breadboard. As implied above, the clock is powered via a USB cable and a USB power supply module or alternatively, it can be plugged into a USB port on your computer.

The wire by wire connections are shown below as they might not be clear from the Fritzing diagram.

NodeMCU Amica Pin	Adafruit 1.8" Display Connections
D4	D/C
D5	SCK
D7	MOSI
D8	TFT_CS
3V3	RESET
3V3	LITE
3V3	3V3
GND	Gnd

Both the Adafruit and the Sainsmart displays have a micro SD memory card connector and interface which can be used with the ESP8266 although they are not required for this project.

Software

The software for the ESP8266 World Clock was developed using the Arduino IDE. I used version 1.8.0 for MacOS but you should be able to use the Arduino IDE on Windows as well. See my previous articles or the *Resources* section for how to set-up the Arduino IDE on your computer for targeting ESP8266 type devices. Make sure to select “*NodeMCU 1.0 (ESP-12E Module)*” as the board type in the tools menu.

The ESP8266 World Clock software should be available in the code associated with this document. The file is called: *ESP8266_WorldClock.ino*. To use this software, copy/move the

Chapter Four - World Clock

ESP8266_WorldClock directory into your computer's Arduino directory. The code directory contains the versions of the libraries I used during program development. It is important to use these versions as newer or older versions may not function correctly. These library files should be moved into your arduino/libraries directory for use. Remember the Arduino IDE, if running, must be restarted to recognize new libraries after you install them.

Whereas the hardware for this World Clock borders on the trivial, the software/firmware for the clock is a bit more involved and complex. The files which make up the program are described in the table below:

File	Description
ESP8266_WorldClock.ino	Main program which initializes the hardware for operation and runs a finite state machine that insures the WiFi connection to the Internet is continually maintained.
DisplayFSM	Code for formatting and displaying time and date information on the LCD and a finite state machine that controls the order that timezones are displayed in, which timezone is currently being displayed and how long each timezone is displayed. <i>Time Change Rules</i> (described shortly) for each timezone are also defined in this file and the conversion of NTP's UTC time to local timezone time is perform here as well.
NTP.h	Functions for sending UDP packets to NTP servers on the Internet and retrieving the returned UTC time.
TextGraphicsFunctions.h	Misc functions for formatting text data for display on the LCD.
TimezoneExt.h	A subclass of the <i>Timezone</i> class that contains the <i>Time Change Rules</i> in addition to a name assigned to the timezone.

In addition to the files above, the following Arduino libraries are required:

Library	Source
TFT_ST7735	https://github.com/sumotoy/TFT_ST7735
Time	https://github.com/PaulStoffregen/Time
Timezone	https://github.com/JChristensen/Timezone

User Configuration of the World Clock Software

The World Clock's software must be configured before the clock will operate correctly. All user configuration items are found in the file, *ESP8266_WorldClock.ino*. Please locate the following text in that file:

```
// *****  
// Begin user configurable items  
// *****  
  
// Set your WiFi login credentials  
#define WIFI_SSID "XXXXXX"  
#define WIFI_PASS "YYYYYY"  
  
#define USE_12_HOUR_FORMAT true  
#define PAGE_DISPLAY_TIME_SECS 20  
#define PAGE_DISPLAY_TIME_MS (PAGE_DISPLAY_TIME_SECS * 1000)  
  
// *****  
// End user configurable items  
// *****
```

First and most importantly you must modify the code with the SSID and Password of your WiFi network otherwise the clock won't be able to access the Internet and by extension the NTP servers that provide the time. Next you must decide if your clock will operate in 12 or 24 hour format. Set `USE_12_HOUR_FORMAT` true to run your clock in 12 hour format otherwise it will operate in 24 hour time format. Finally you can configure how long each timezone display page is displayed. By default the time is 20 seconds but you can make this longer or shorter depending upon your preference.

The code can be compiled and uploaded to the NodeMCU device once the configuration data is set and all of the required libraries have been installed in the Arduino environment.

World Clock Operation

The clock should start immediately once the software is compiled and uploaded. Figure Three shows the clock's display while a connection is being made to the local WiFi network. If this screen doesn't change to a timezone display similar to Figure Four it means there were problems logging into the WiFi network. If this is the case, go back and verify that the `WIFI_SSID` and `WIFI_PASS` entries in the code are correct and that the WiFi network is working.

The WiFi login display will be replaced by the first timezone display page once a WiFi connection is established. The clock should run as long as power is applied and it will sync its time to an NTP time server every five minutes, making the clock very accurate. Each timezone display page will be displayed for the configured time interval in a round robin fashion. By default this means time and date will be displayed in the following order: Sydney, Frankfurt, London, New York, Houston, Denver, Phoenix and Los Angeles.

Chapter Four - World Clock

The clock will continue to run as long as power is applied. If the Internet connection is dropped, the clock will maintain the time itself. If WiFi goes down but the clock remains powered, the clock will continually try to reestablished Internet connectivity and will re-sync with NTP as soon as possible. If power is lost to both the clock and the WiFi network, when power is restored the clock will reboot and wait for the network to come back up and will then reconnect automatically.

Timezones, Timezones, Timezones

Wrapping my head around world wide timezones was the most difficult part of writing this code. Think about it. Its a somewhat easy task to calculate offsets from the time in your location to any timezone in the world but what happens when the timezone you are interested in changes to or from daylight savings time or whatever they call it in their location. To compound the problem there is no world standard that I am aware of that says the time changes from standard to daylight savings time and back have to occur on the same date and time in all locations. Luckily the timezone library used in this application, written by J Christensen, has this kind of smarts built in. It is our responsibility, however, to define *Time Change Rules* for each timezone that describe to this library the rules for each location.

To define the appropriate *Time Change Rules* for each timezone requires gathering the following information. The day of the week the time change occurs, which week of the month the change occurs in, at what hour the change occurs and the offset in minutes of the timezone from universal coordinated time or UTC when the change occurs. Two rules are required for each timezone. One for the change to daylight savings time and another for the change back to standard time. Since I live in Colorado, I'll use the *Time Change Rules* for the mountain timezone as an example. The following code fragment was extracted from the file *DisplayFSM.h*.

```
// US Mountain Time Zone (Denver, Colorado Springs, Salt Lake City)
TimeChangeRule usMDT = {"MDT", Second, Sun, Mar, 2, -360};
TimeChangeRule usMST = {"MST", First, Sun, Nov, 2, -420};
TimezoneExt usMT("Denver", usMDT, usMST);
```

The first rule, *usMDT*, is the rule for when daylight saving time starts in my location. It says the time change occurs the second Sunday of March at 2 AM and that during DST we are offset a negative 360 minutes or -6 hours from UTC time. The second rule, *usMST*, is for standard time that starts the first Sunday of November again at 2 AM. During standard time we are -7 hours from UTC time. Note: there is no mention of the day of the month that time changes as that is different every year.

These two *Time Change Rules* are then combined into a *TimezoneExt* object called, *usMT*, and given the name “Denver” which will be displayed for the mountain time zone on the LCD. Using these rules UTC time returned from NTP is converted to local time for the timezone and that is what gets displayed on the World Clock.

Arizona is unique in that they don't use daylight savings time in their state (they may have the right idea). Arizona is on standard Mountain time the whole year. To accommodate this, the *TimezoneExt* object specifies *usMST* for both of their *Time Change Rules* as shown below:

```
// Arizona is US Mountain Time Zone but does not use DST
```

```
TimezoneExt usAZ("Phoenix", usMST, usMST);
```

After all the *Time Change Rules* and the *TimezoneExt* objects are defined in the file *DisplayFSM.h* the *TimezoneExt* objects are placed into an array called *timeZones* as shown below.

```
// All of the timezone we want to display time for
TimezoneExt timeZones [] = {
    ausET, CE, UK, usET, usCT, usMT, usAZ, usPT
};
```

The World Clock code walks through this array sequentially and displays the time and date information calculated for each timezone. Because you have access to the software you could change the order of the timezones displayed, remove timezones you are not interested in or add new timezones of your choice. The World Clock code should automatically adapt to any changes you make to the *timeZones* array. If you have a large distributed family you might want to have your World Clock display the time and date where each family member lives.

You'll want to check the time and date for any timezones you add and this can easily be done by googling, “time in XXXX” where XXXX is the name of a city in the timezone you added.

Conclusions

The World Clock presented in this article is one of the simplest projects one could build and it can be built for around \$22. Two components connected with 6 wires is all it takes. Plug your World Clock into your computer, update the user configuration info, compile the code and upload it to your World Clock and you should be good to go. Now you can feel like you are living in a high end international hotel as you have access to times and dates all over the world.

As always, have fun !

Resources

The following resources may be of use:

Information about NTP can be found all over the Internet. See <http://www.ntp.org/> for detailed information.

Information about timezones can be found here:
https://en.wikipedia.org/wiki/List_of_UTC_time_offsets

Information about programming the ESP8266 in the Arduino environment can be found at:
github.com/esp8266/Arduino and in my articles previously mentioned.

Information about the NodeMCU Amica can be found at: www.electrodragon.com/product/nodemcu-lua-amica-r2-esp8266-wifi-board/.

Chapter Four - World Clock

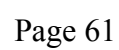
Information about the Adafruit 1.8" TFT SPI LCD display can be found at:

<http://www.adafruit.com/products/358>. Information about the Sainsmart display can be found at:
<http://www.sainsmart.com/sainsmart-1-8-spi-lcd-module-with-microsd-led-backlight-for-arduino-mega-atmel-atmega.html>.

The ST7735 display driver library can be found at: https://github.com/sumotoy/TFT_ST7735

The Time library can be found at: <https://github.com/PaulStoffregen/Time>.

The Timezone library can be found at: <https://github.com/JChristensen/Timezone>.



Chapter Four - World Clock

Figure Two
The ESP8266 World Clock Breadboard

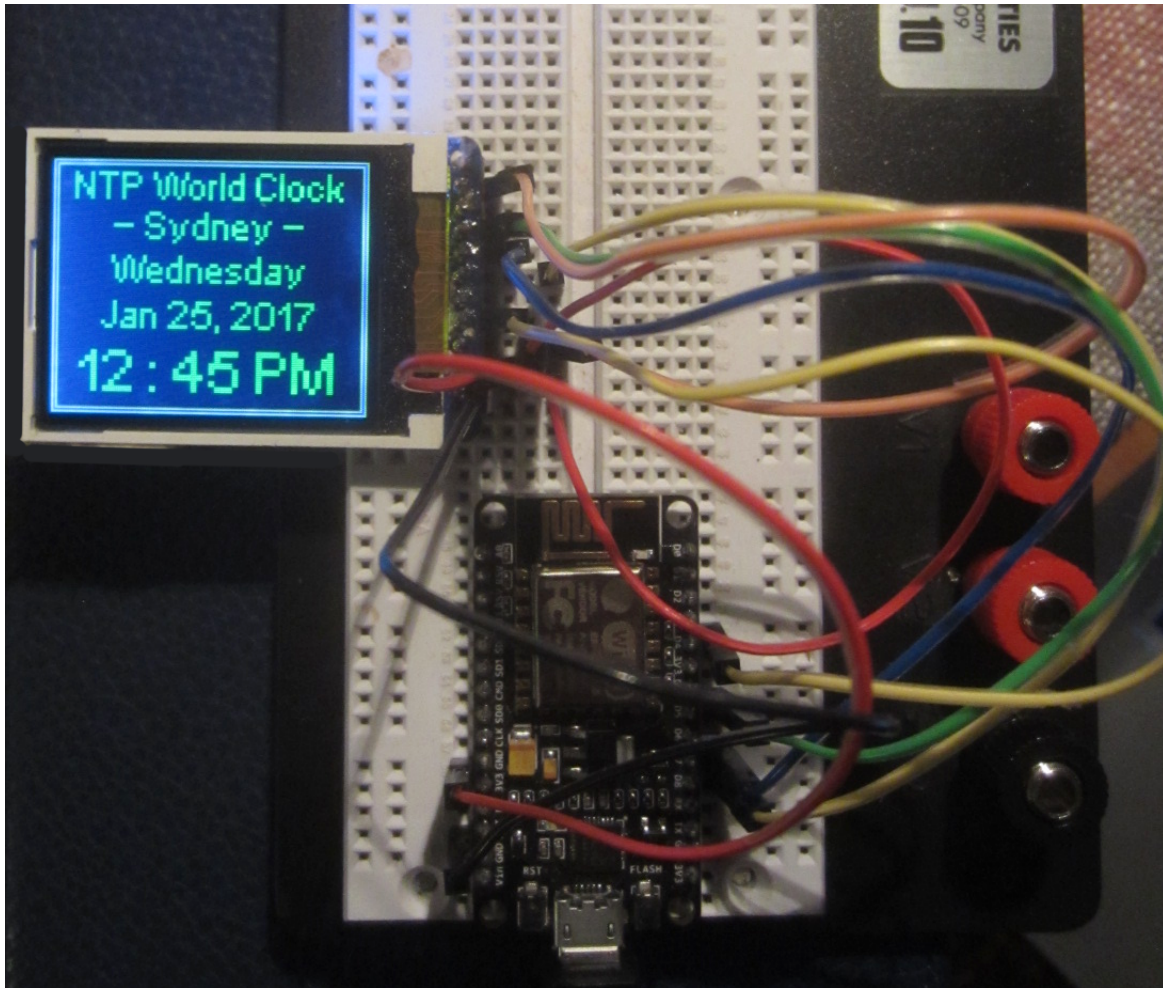


Figure Three
Initial WiFi Connection Display



Chapter Four - World Clock

Figure Four
Typical World Clock Timezone Display Page

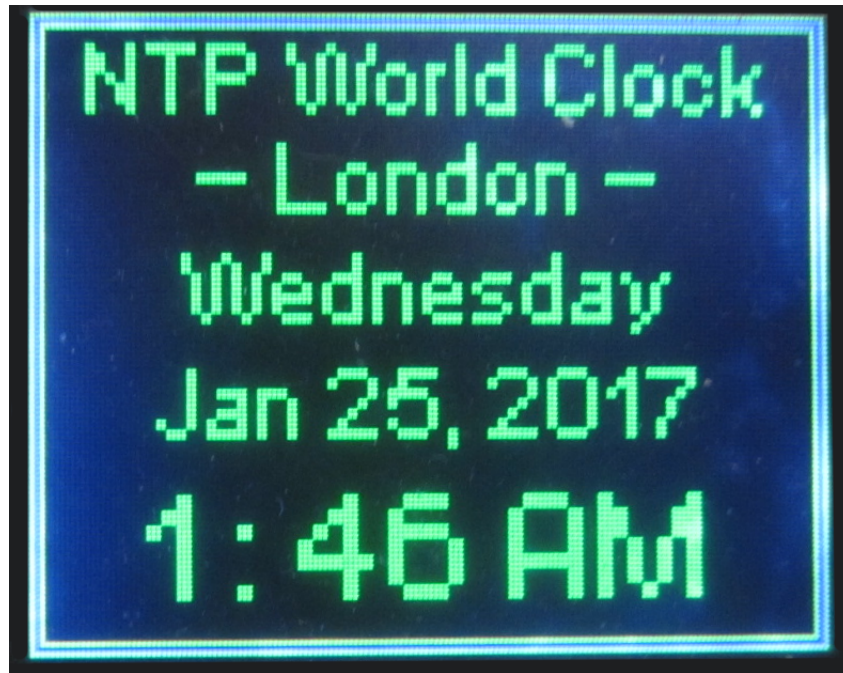
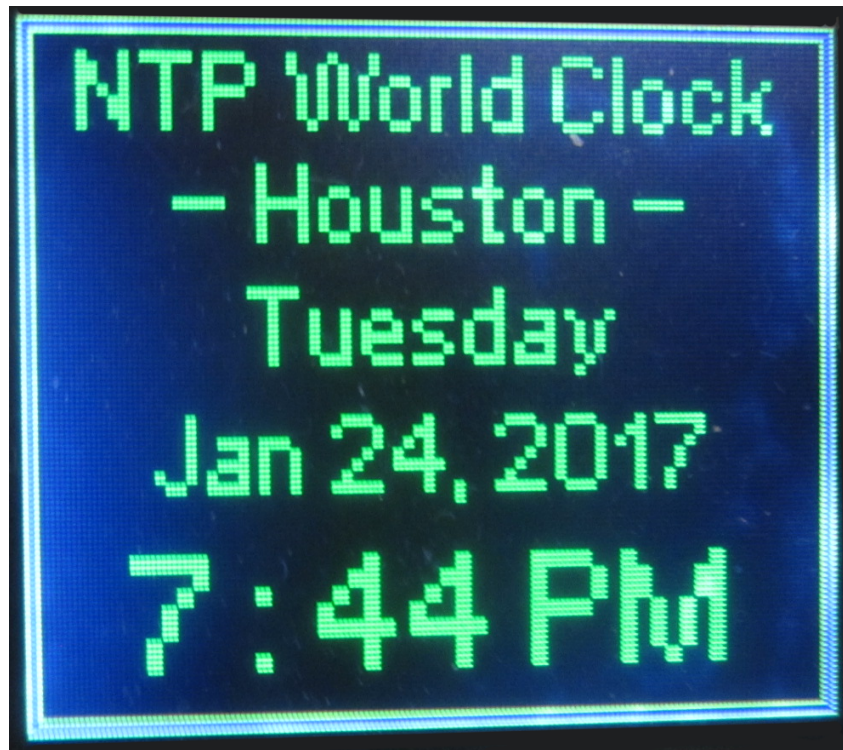


Figure Five
Another World Clock Timezone Display Page



Chapter Five - Nixie Tube Clock

Introduction

For some unknown reason I have always wanted to build a Nixie Tube clock and now I finally have. For a long time I looked around for a nice kit but they always cost more than I was willing to pay. While on eBay recently I saw an Arduino Nixie Tube Shield for a reasonable cost so I decide the time to build a Nixie Tube clock was now. This is a fully assembled and tested shield which was meant for connection to an Arduino Uno. It had all the required hardware like a high voltage power supply for the tubes, a real time clock chip with battery backup and the switches necessary to set the time and date. I decided to get the clock working and then decide how to package it.

I, however, have had my fill of clocks that require any kind of ongoing maintenance on my part like setting the time / date and changing the time for Daylight Savings Time (DST). Each time DST occurs I have to go around my home and change the time on all of the clocks and then do the same for my cars. No more. Any clocks I build going forward will perform these tasks automatically.

To this end, I decided to use an ESP8266 (a NodeMCU Amica module actually) in place of the Arduino Uno this shield was designed for. This would give my clock access to the Internet as the ESP8266 has built in WiFi capabilities. With Internet access my clock could get the time by polling Network Time Protocol (NTP) servers on the Internet. This is the same technique personal computers use to set their time. From NTP time the clock could calculate the current date so my Nixie Tube clock would always display accurate time and date information. Going further my software incorporates a Timezone library that understands DST changes in my location so my clock automatically adjusts for that as well. Perfect, a clock that requires absolutely no ongoing maintenance on my part.

Maybe you should build a Nixie Tube clock of your own. The design incorporates display technology from the past with state of the art modern embedded electronics, a nice combination.

Clock Operation

Before getting into the details of this build I want to give you an idea of how the clock works.

After power is applied to the Nixie Tube clock there will be a short delay as the clock accesses the Internet for NTP time. After that delay the clock does what is called the anti-poisoning function which runs each tube through all the numeric digits four times. If this is not done periodically, the unused digits will slowly darken to the point of being unusable. As a matter of fact the clock runs this anti-poisoning function at the top of each hour.

The Nixie Tube shield has RGB LEDs under each Nixie Tube which are controlled via the custom software I have written. During the normal operation of the clock the color of the LEDs change over the course of time. The complete spectrum of colors is displayed in a twelve hour period if the clock is running in 12 hour mode and over a twenty four hour period if the clock is in 24 hour mode. The LEDs

Chapter Five - Nixie Tube Clock

change colors in a subtle way over the course of a day.

Continually displaying just the time on the Nixie Tube clock can get a little boring so I have built events into the software to liven things up a bit. There are three defined events: the 10 minute event causes the clock to change momentarily from the display of time to the display of the date; the 15 minute event which causes the clock to perform a little light show by running the RGB LEDs under the Nixie Tubes through all the colors of the rainbow; and finally the top of the hour event which performs the anti-poisoning function described above. After the events have completed, the normal display of time is resumed.

Finally, the clock's software turns off the Nixie Tubes at a predefined time in the evening and turns them back on in the morning. This is done to lengthen the lives of the Nixie Tube's by turning them off when no one is around to see the time.

Hardware

I bought the Arduino Nixie Tube Shield from gra-afch.com on eBay for about \$94 US. See Figure One. It came fully assembled and tested. It has all of the high voltage circuitry required for the tubes, a RTC chip with backup battery and switches used to set the time and date. It is meant to be used with an Arduino Uno which is not supplied. This is one of the cheapest Nixie Tube clock assemblies I have ever come across.

To verify that the shield worked I plugged on an Arduino Uno and loaded the Arduino Uno code from gra-afch. As expected it worked perfectly. The shield seems to be well constructed with SMT components and uses quality parts throughout. Figure Two shows the shield in operation with an Arduino Uno.

As mentioned I wanted to drive my Nixie Tube clock with an NodeMCU Amica module so I built the NodeMCU module onto a small piece of pref board that has the same form factor as an Arduino Uno. See Figure Three. This allows the NodeMCU module to plug directly onto the Nixie Tube shield in place of the Arduino Uno with no additional wiring being necessary. Since I was going to use NTP as my time source I would not be using the RTC chip or any of its support circuitry on the shield other than the high voltage power supply and the Nixie Tube drive circuitry.

I built a 5 volt DC voltage regulator onto the pref board to reduce the 12 volt power from the shield down to 5 volts for the NodeMCU. This was probably unnecessary but meant the onboard NodeMCU voltage regulator would not have to dissipate so much power allowing the module to run cooler which is always a good thing for electronic circuitry.

Parts list follows:

Part	Description	Source
NodeMCU Amica Module	32 bit CPU with WiFi	electrodragon.com

Part	Description	Source
Power Adapter	12 Volt DC 2 Amp minimum	amazon.com
5 Volt Voltage Regulator (optional)	7805 or equivalent voltage regulator in TO-220 case	adafruit.com
2x 100 uF 25 volt capacitors (optional)	Filter capacitors	Radioshack
2x 0.01 uF capacitors (optional)	Filter capacitors	Radioshack
Pref board shaped like an Arduino Uno	For building circuit onto	Radioshack
Connectors	2.54mm 40 Pin Female Single Row Pin Header Strip	Ebay
Power connector	5.5mm X 2.1mm DC Power Female Socket Panel Mount	amazon.com

Additional materials for packaging depend upon how you package the clock.

The connections between the NodeMCU module and the Nixie Clock shield are as follows:

NodeMCU Pin	Shield Pin	Signal / Function
D5	SCK	SPI clock
D7	MOSI	Master Out Slave In data from NodeMCU to shield
D8	LE	Latch Enable
D9	DOT1	Neon dot 1
D9	DOT2	Neon dot 2
D0	PWM1	Green LED drive
D1	PWM2	Red LED drive
D2	PWM3	Blue LED drive
D3	SHTDN	High voltage enable
GND	GND	Ground connection
Vin	VIN This voltage is reduced by an	12 volts from shield

Chapter Five - Nixie Tube Clock

	onboard regulator. 5 volt output of regulator feeds Vin pin of the NodeMCU module.	
--	--	--

Software

To gain access to the Internet the NodeMCU ESP8266 must be configured to talk to the local WiFi network. Whereas the SSID and password of the WiFi network could be hardcoded into the clock's software I decided to use the WiFi Manager library (see <https://github.com/tzapu/WiFiManager> for details) which allows the WiFi credentials to be set via a web interface. Use of the WiFi Manager library means the clock can be moved between WiFi networks without changes to the code being necessary. Here is how it works.

When the program is first started the ESP8266 creates a wireless access point (AP) called *NixieClock* that the user needs to connect to. Then if the user points his/her browser to 192.168.4.1, a page is presented that allows the credentials for the actual WiFi network to be entered. This only needs to be done once since the credentials will be stored in the ESP8266's EEPROM and will be used from that point forward. If, at some point in the future, the ESP8266 cannot connect to the WiFi network using the stored credentials, it will again create the *NixieClock* access point to allow new WiFi credentials to be entered. Pretty slick if you ask me.

All of the Nixie clock software was developed using the Arduino IDE version 1.8.0. The sketch is called *ESP8266_NTPNixieClock.ino* and when you load it into the IDE you will be presented with four tabs for the four files that make up the sketch. The purpose of each file is spelled out below:

File	Function
ESP8266_NTPNixieClock.ino	This is the main sketch file which coordinates the operation of the clock. The setup() function initialized the shield driver, sets up SPI, runs the WiFi Manager if required, initializes the NTP code and then runs the anti-poisoning function for the Nixie tubes.
LEDControl.h	Has code for controlling the RGB LEDs which are located underneath each of the Nixie tubes on the shield. PWM via <i>analogWrite</i> is used to control the RGB LEDs giving a full 24 bit range of colors. Eight of the most common colors are defined in this file.
NTP.h	This file contains the code necessary to poll a NTP provider (time.nist.gov) across the Internet for the current time expressed in seconds since Jan 1, 1970 midnight UTC/GMT. UDP networking is used for this communication.
NixieTubeShield.h	This is the driver for the shield. It controls the high voltage power

File	Function
	supply, controls driving the two neon bulbs, and most importantly controls sending 60 bits of serial data to the Nixie tubes (10 bits for each tube) every time the <i>show()</i> function is called. The <i>show()</i> functions uses a finite state machine (FSM) for its operation.

Before you can use the Nixie Tube clock sketch you must configure it for your use. At the top of the main sketch file there is a section for user configurable items shown below:

```
// *****
// Start of user configuration items
// *****
// Name of AP when configuring WiFi credentials
#define AP_NAME "NixieClock"
// Checks WiFi connection. Reset after this time, if WiFi is not connected
#define WIFI_CHK_TIME_SEC 60
#define WIFI_CHK_TIME_MS (WIFI_CHK_TIME_SEC * 1000)
// Set to false for 24 hour time mode
#define HOUR_FORMAT_12 true
// Nixie tubes are turned off at night to increase their lifetime
// Clock off and on times are in 24 hour format
#define CLOCK_OFF_HOUR 23
#define CLOCK_ON_HOUR 07
// Suppress leading zeros
// Set to false to having leading zeros displayed
#define SUPPRESS_LEADING_ZEROS true
// Define the timezone in which the clock will operate
// See the Timezone library for details
// US Mountain Time Zone (Denver, Salt Lake City)
TimeChangeRule usMDT = {"MDT", Second, Sun, Mar, 2, -360};
TimeChangeRule usMST = {"MST", First, Sun, Nov, 2, -420};
Timezone TZ(usMDT, usMST);
// *****
// End of user configuration items
// *****
```

Chapter Five - Nixie Tube Clock

Most of these entries should be self explanatory from the comments provided and you should make any changes necessary to make your Nixie clock operate as you want it to. The suppress leading zeros define turn off the leading digits for time and date display if the values to be displayed are less than ten. Information about *TimeChangeRules* and *Timezones* can be found in the World Clock article.

Once you have made your configuration changes you can compile and upload the resultant code to the NodeMCU device via the USB connection.

Packaging

I did not want to build my state of the art Nixie Tube clock into a boring rectangular box so I decided instead to build it into an elliptical shape.

The process started by printing out an ellipse of the proper size (4 3/8" x 11") and gluing it to a piece of 1/8" MDF. I then cut the MDF to shape and drilled a series of holes around the ellipse equal distant from the edges. This became a template for my router for machining the various parts I needed. See Figure Four.

I then attached the template to a piece of 3/4" Baltic Birch which was slightly larger than the template using double sided tape and used a flush trim router bit with a bearing to cut the wood to shape. The bearing runs along the template making the router cut out the exact shape. I then drilled all of the hole in the template through the wood and then used a jig saw to saw out the middle of the ellipse. See Figure Five. I made two of these that I glued together as the clock's base needed to be about 1 1/2" deep.

I again used the template and double sided tape to create the top out of black 1/8" plastic. After carefully measuring the positions of the Nixie Tubes, I drilled holes in the top for the six tubes and the two smaller neon bulbs. See Figure Six. I used the template once more to create the bottom as well out of the same plastic material.

After gluing the two sections of the chassis together and sanding them to 240 grit I stained them with mahogany stain. I attached four cabinet knobs to the bottom to use as feet to give my clock a modern look. Figure Seven show all of the chassis pieces together to get an idea of how things fit. The knob feet screw onto the bottom plastic piece which in turn screws onto the wooden chassis but there are no exposed screws anywhere else.

Figure Eight shows two views of the finished clock. Figure Nine shows the rear view of the clock with only a power connector. As noted there are no switches or buttons to set the time or date or to indicate daylight savings time. Setting of the clock is all done automatically via NTP.

Figure Ten shows the clock in operation. My clock runs in 12 hour time and I suppress leading zeros which is why the left most tube is off. The displayed time is 5:58:14. The clock can also run in 24 hour time by making a small change in the software.

Of course this is just one packing possibility. If you use your imagination I am sure you can come up with other cool packing ideas.

Conclusions

The use of the Nixie Tube shield makes building a Nixie Tube clock a lot simpler. The use of NTP time and the custom software provided with this article make the clock essentially maintenance free from the user's perspective. And as I stated earlier this design incorporates display technology from the past with state of the art modern embedded electronics, how cool is that?

Chapter Five - Nixie Tube Clock

Figure One

The Nixie Tube Shield

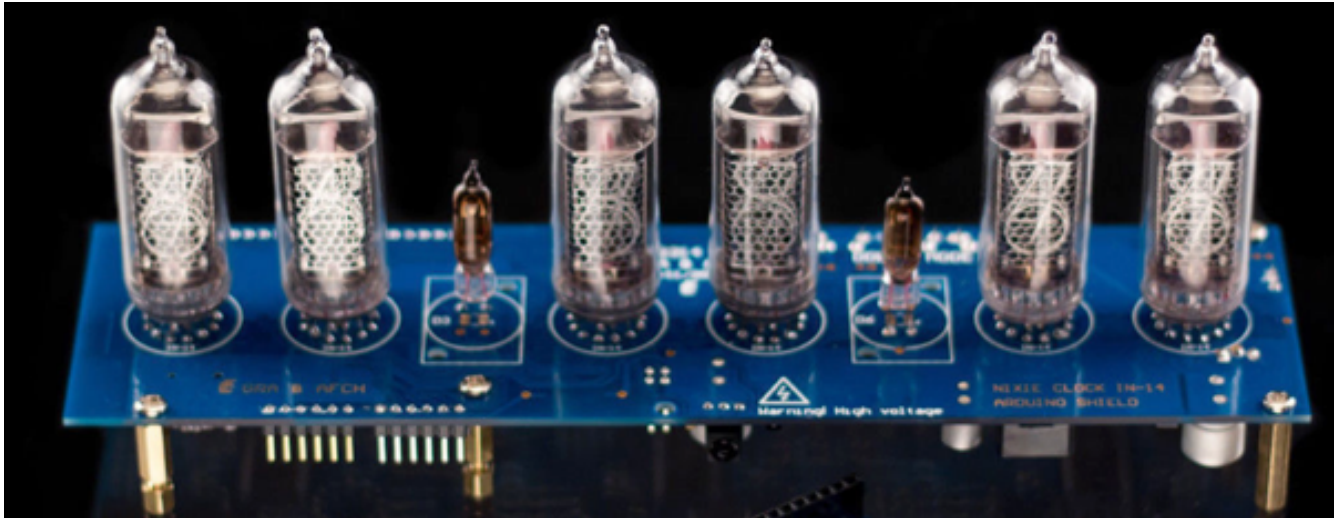
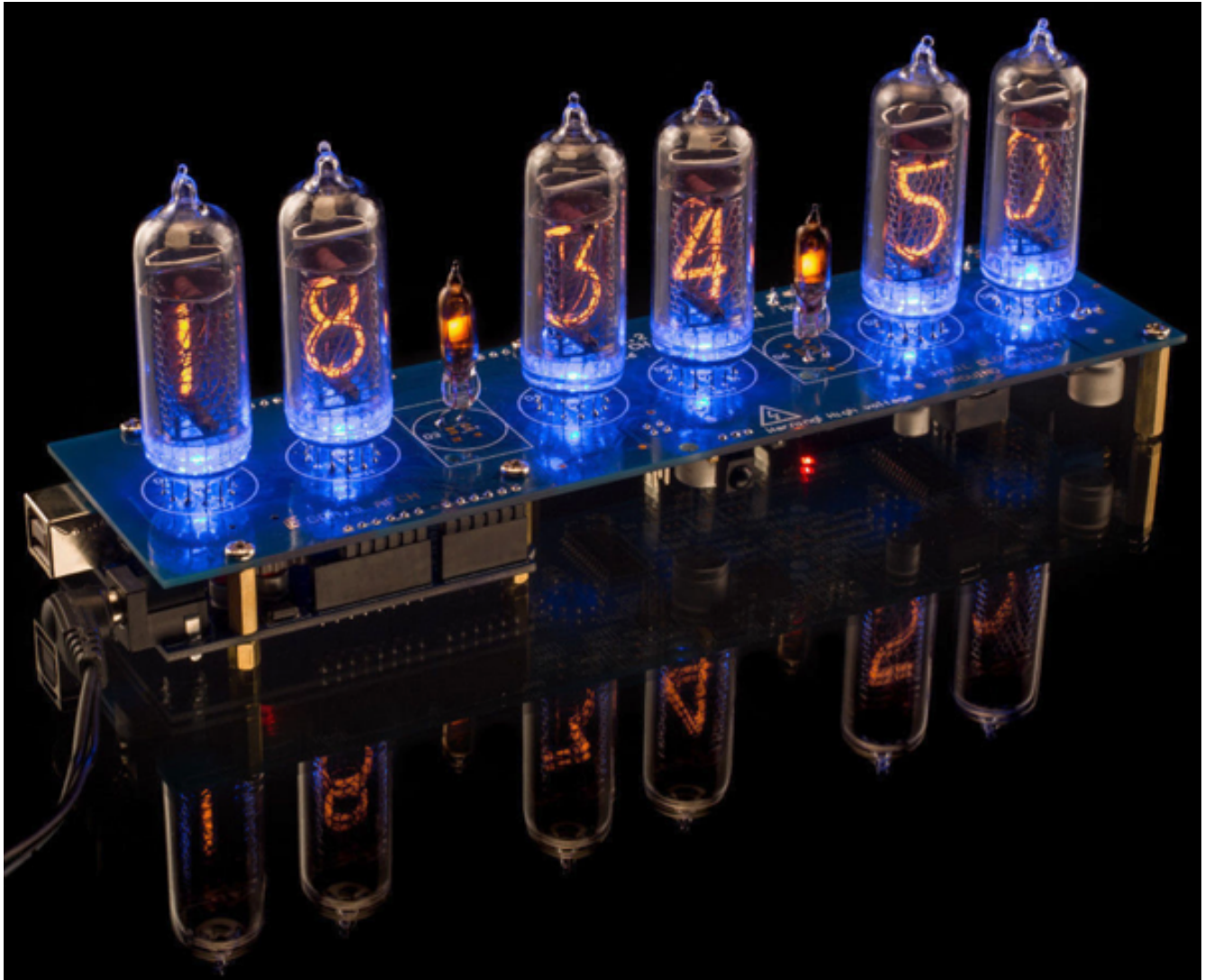


Figure Two

Shield tested with Arduino Uno plugged on



Chapter Five - Nixie Tube Clock

Figure Three

Arduino Uno like form factor for the NodeMCU Amica module
The connectors correspond to the layout on an Arduino Uno module.

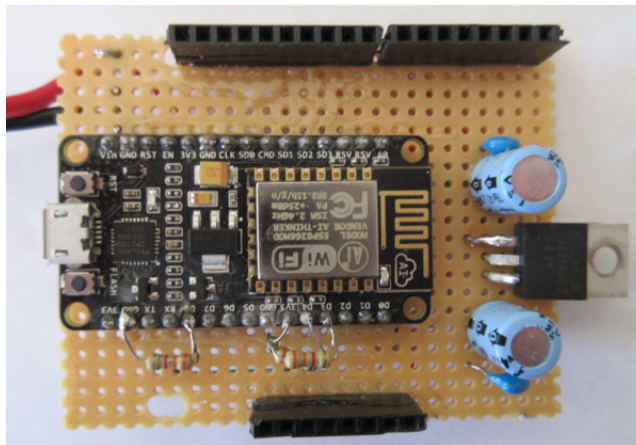
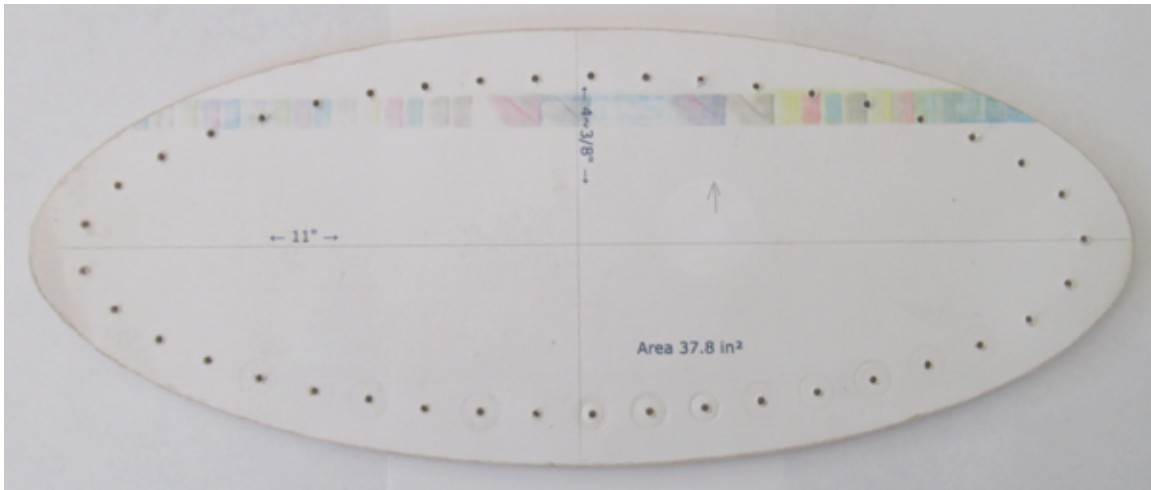


Figure Four

The Ellipse Template

Ellipse dimensions are $4 \frac{3}{8}'' \times 11''$



Chapter Five - Nixie Tube Clock

Figure Five

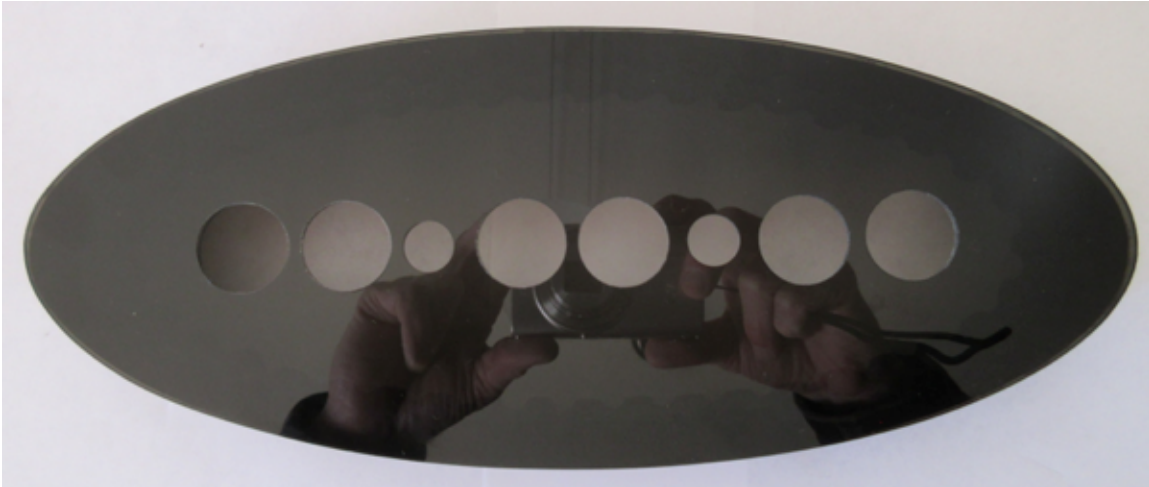
Ellipse Chassis Taking Shape



Figure Six

Template used again to machine top and bottom of chassis

Large holes in the top are for the Nixie tubes and smaller holes are for the neon bulbs



Chapter Five - Nixie Tube Clock

Figure Seven

Test fitting pieces together

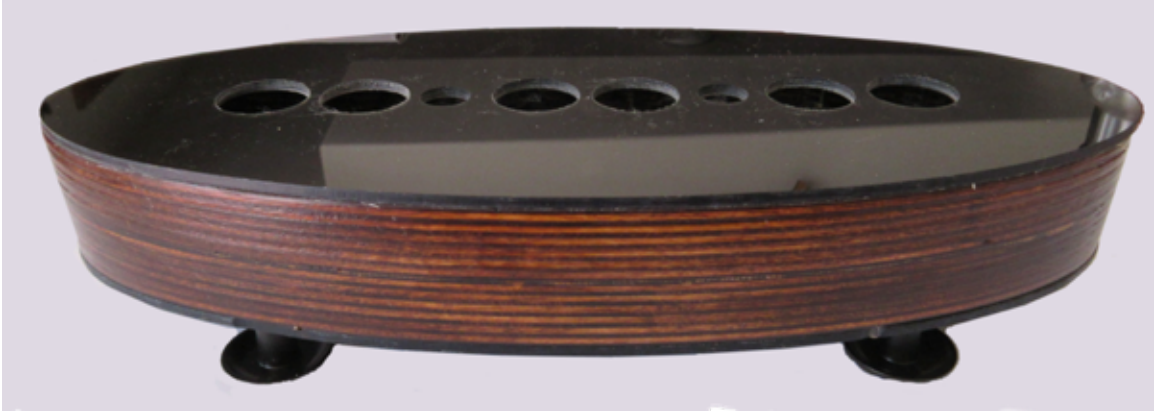


Figure Eight

The finished clock in two views



Chapter Five - Nixie Tube Clock

Figure Nine

Rear view of the clock showing the power connector

Notice no buttons or switches for time/date setting



Figure Ten

Clock in operation

Time is 5:58:14



Chapter Six - RSS News Reader

Introduction

I admit it; I am a self proclaimed news nut. I don't know, this might actually be an undiagnosed illness since I look at the news on the web constantly during the day and then watch the news in the evenings on TV. There is always something going on in our world (for better or worse) that I don't seem to want to miss. So when I was thinking about other applications for the amazing ESP8266 I thought why don't I write an RSS reader so that I can monitor the headlines from many different news sources from around the world. Then if something catches my attention, I can open up my laptop and read the full story.

For those of you who aren't familiar with RSS, according to Webopedia:

“RSS is the acronym used to describe the de facto standard for the syndication of Web content. RSS is an XML-based format and while it can be used in different ways for content distribution, its most widespread usage is in distributing news headlines on the Web”.

An RSS XML document, also called a feed or channel includes summarized headline text along with metadata, like publication date and author information.

The RSS acronym itself has had numerous definitions over time. Originally it meant **R**DF Site **S**ummary, later RSS was defined to mean **R**ich Site **S**ummary. Currently RSS is understood to mean **R**eally **S**imple **S**yndication. Regardless of the true definition, in this article I will use the term RSS to mean an Internet feed of headlines from one or more news/information sources.

If you have been reading Nuts and Volts regularly you probably have noticed that I have written quite a few article about putting the NodeMCU Amica module with embedded ESP8266-12 chip to work. These articles include:

1. “Meet the ESP8266: A Tiny, WiFi Enabled, Arduino Compatible Micro Controller” in the October 2015 issue.
2. "Thinking of You" article in the November 2015 issue.
3. “ESP8266 NTP Clock” article in the June 2016 issue.
4. “ESP8266 Weather Clock article in the November 2016 issue.

All of these projects were developed within the Arduino integrated development environment (IDE) using the ESP8266 as a relatively high performance micro-controller with a built in WiFi interface. That is to say no other micro-controller was used to control the ESP8266 as a peripheral like in so many other ESP8266 projects I see on the Internet. Hosting applications directly on the ESP8266 itself both drives project costs down and increases reliability at the same time as a result of fewer parts and

Chapter Six - RSS News Reader

less software involved.

In this article I am going to describe an RSS News Reader application I built on the same hardware as my previous two articles. That is, a NodeMCU Amica module, an Adafruit 1.8" TFT LCD display, a pushbutton switch, USB cable, USB power supply and some wire. With this hardware you can upload the NTP Clock software and have a auto setting time and date clock; upload the Weather Clock software and get current and forecasted weather conditions for your location along with the NTP clock functionality; or the software I provide with this article to have an RSS News Reader all without changing a single wire or component.

As supplied, the software for this RSS News Reader has hardcoded RSS feeds for NPR, CNN, AP, L.A. Times, BBC, Reuters and USAToday and it is a simple matter to add your own feeds and/or delete or reorder any that I have supplied.

Once the RSS News Reader is configured it will connect to your local WiFi network and then make a request for headlines from NPR (which happens to be at the top of the feed list but more on that later) and will continuously cycle through the display of headlines by horizontally scrolling them across the LCD. After all headlines have been displayed, a new request to NPR will be made and the process repeats.

If, however, you press and hold the feed advance pushbutton until the last headline has scrolled off of the screen you will advance to the next feed on the feed list and its headlines will then be displayed. When a headline has an associated publication date, that will be displayed on the LCD as well.

To summarize, the RSS News Reader will continuously display headlines from the selected news feed until the feed is changed using the feed advance pushbutton or it is powered down. Since the RSS News Reader makes a new request every time all of the headlines have been displayed, the headlines you see will be as up to date as the news source itself.

Hardware

As mentioned, the RSS News Reader uses the same hardware as used in my two previous ESP8266 articles. To save you from going back and (re)reading previous articles, the hardware information is repeated here starting with the parts list.

Part	Source
NodeMCU LUA Amica R2 Module	Electrodragon.com
1.8” TFT SPI LCD Display (blacktab)	Adafruit.com - Product ID: 358
Pushbutton Switch SPST	Radio Shark or anywhere else
USB Cable - USB A to USB Micro B	Radio Shack or anywhere else
USB Power Supply capable of at least 1 amp @ 5 volts	Radio Shack or anywhere else
Hook up wire and breadboard	Radio Shack or anywhere else

Figure One shows a Fritzing connection diagram/schematic for the RSS News Reader. Figure Two shows the design wired up on a breadboard. The RSS News Reader is powered via a USB cable and a USB power supply module although it could be powered by plugging it in to your computer.

The wire by wire connections are shown below because they might not be clear from the Fritzing diagram.

NodeMCU Amica Pin	Adafruit 1.8” LCD Display Connection	News Feed Advance Pushbutton SPST Switch
D1 (GPIO 5)		SW1
D3 (GPIO 0)	LITE	
D4 (GPIO 2)	D/C	
D5	SCK	
D7	MOSI	
D8 (GPIO 15)	TFT_CS	
3V3	VCC	
GND	Gnd	SW2

The GPIO designations are shown above as that is how these digital I/O lines are referred to in the Arduino code.

The Adafruit LCD display also has a micro SD memory card interface which can be used with the ESP8266 but it was not needed for this project.

Software

As mentioned, the software for the ESP8266 RSS News Reader was developed using the Arduino IDE. See my previous articles and/or the *Resources* section for how to set-up the Arduino IDE on your computer for targeting ESP8266 type devices. Make sure to select “*NodeMCU 1.0 (ESP-12E Module)*” as the board type in the tools menu.

Chapter Six - RSS News Reader

The ESP8266 RSS News Reader software should be available in the code directory associated with this document. The file is called *ESP8266_RSSNewsReader.ino*. To use this software, copy/move the ESP8266_RSSNewsReader directory from the code directory into your Arduino directory.

While the hardware is about as simple as it gets, the software is somewhat complicated and is made up of the following files:

File	Description
ESP8266_RSSNewsReader.ino	Main program. Initializes the hardware and software, logs into the local WiFi network, initializes the RSS reader callbacks and prepares the <i>loop()</i> function for accessing and retrieving the selected RSS feed.
ESP8266_ST7735.cpp	LCD driver code specific to the Adafruit 1.8" (blacktab) display utilizing the hardware SPI interface of the ESP8266.
ESP8266_ST7735.h	Header file for the LCD driver code above
RSSReader.cpp	Implementation of the RSSReader class. The RSSReader has a lot of functionality including implementing a crude parser for extracting information from RSS XML documents, has code for parsing URLs to extract the host and the path components and the code that requests the RSS documents from news sources across the Internet. It also manages the feed advance pushbutton.
RSSReader.h	Header file for the RSSReader code above
TGFunctions.h	Misc functions for formatting text and graphical data for display on the LCD.
TextScroller.cpp	Class for horizontally scrolling text strings on the LCD display.
TextScroller.h	Header file for the TextScroller code above
Icons.h	Data for the WiFi icon used on the login screen in xbm format.

In addition to the files listed above, a modified version of the Adafruit_GFX library is required. This library provides text and graphics functions for the LCD display when connected to an ESP8266 device. Whereas the stock library is available at:

<https://github.com/adafruit/Adafruit-GFX-Library>

the modified version of this library I used to develop the RSS News Reader is included in the code directory associated with this document. Remember libraries must be installed in the *arduino/libraries* directory on your development computer and the Arduino IDE must be restarted to recognize them.

RSS News Reader Software Operation

Before you can use the RSS News Reader you must supply login information for your WiFi network. You do this by opening up the main sketch/program file *ESP8266_RSSNewsReader.ino* and finding the user configuration section shown below:

```
// *****  
// Start of user configuration items  
// *****  
  
// Set your WiFi login credentials  
const char * WIFI_SSID = "xxxxxxx";  
const char * WIFI_PASS = "xxxxxxxxxx";  
  
// *****  
// End of user configuration items  
// *****
```

This information allows the ESP8266 to login to your WiFi network as required to access the RSS feeds across the Internet. The login screen shown in Figure Three is displayed on the LCD while the login process is occurring. It will be replaced with the RSS News Reader Screen shown in Figure Four when the login process completes successfully. Check your WiFi login credentials, WIFI_SSID and WIFI_PASS, if the login screen doesn't go away.

You'll also notice towards the top of the sketch this array of character strings.

// Array of feed URLs

```
const char *rssFeedURLs [] = {  
    "www.npr.org/rss/rss.php?id=1001",  
    "http://rss.cnn.com/rss/cnn_topstories.rss",  
    "http://feeds.bbc.co.uk/news/rss.xml",  
    "http://hosted.ap.org/lineups/SCIENCEHEADS-rss_2.0.xml?SITE=OHLIM&SECTION=HOME",  
    "http://www.latimes.com/rss2.0.xml",  
    "http://rss.cnn.com/rss/cnn_tech.rss",  
    "http://feeds.reuters.com/reuters/topNews",  
    "rssfeeds.usatoday.com/usatoday-NewsTopStories",  
};
```

This is the RSS feed list with the NPR feed as the first entry. That is why the first feed displayed by default is NPR. As you use the feed advance pushbutton, you move down the list one RSS feed at a time and when you increment past the last entry you wrap around to the first again.

Chapter Six - RSS News Reader

You can easily delete feeds from this list or rearrange their order to suit your preferences. You can even add feeds to this list by googling the news provider you are interested in and looking for the URL they publish for their RSS feed(s). Once you have that, insert it into the feeds list, recompile the code and upload it to the NodeMCU Amica module and you will be all set.

Most of the remaining code in the *ESP8266_RSSNewsReader.ino* sketch file is concerned with declaring instances of the lcd driver, the text scroller and the RSS Reader classes and initializing them. The *loop()* function in the sketch continually calls the *read* function of the RSSReader class passing the URL of the RSS feed to display.

The code for the *RSSReader* is the most complex in this application. The complexity is a result of having to request the headline information from a news source and then to extract the information of interest from the XML returned from the source. For those not familiar with XML:

XML is a software and hardware-independent tool for storing and transporting data in human readable format.

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to be self descriptive
- XML is a W3C Recommendation

More information on XML can be found in the links in the *Resource* section.

Usually in a RSS News Reader application hosted on a personal computer, a full blown XML parser would be employed to extract the headlines and other pertinent data. Unfortunately, I couldn't find a XML parser that could fit in the memory available on the ESP8266 so I had to use a different approach for data extraction.

Without going into too much detail, the portions of the RSS XML that I wanted to retrieve resemble the following.

```
<title>This is some headline from a news feed</title>
<description>This optional component of the news story provides more detail than the title</description>
<pubDate>Fri, 06 Nov 2015</pubDate>
```

All of the components have the same format. A start tag like `<title>` followed by the actual content followed by an end tag `</title>`. Whereas the title portion of a news headline is required in the RSS XML, the description and the pubDates are not. Note, this application doesn't currently use the description information from the XML even though the code supports retrieving it.

To extract data from the XML I wrote a Finite State Machine or FSM that is feed every character returned from the news source over the Internet. The FSM ignores all characters until it sees an opening `<`. Then it starts to listen for the tags of interest which in this case are title, description and pubDate. If

it finds one of these it starts collecting the text starting after the > in the start tag until it see < which is the start of the end tag. Once the text is collected a callback (more on callbacks shortly) is made to the main program passing this text. After the callback completes, the FSM goes back to listening for an opening < character and the process repeats. The complete XML text is processed this way. This is not as elegant or as easy of a solution to parsing XML as using a real XML parser but it gets the job done.

The *RSSReader* class has three functions that allow application code to register interest in the data extracted from the RSS XML. They are:

```
void setTitleCallback(pt2Function titleCallback);
void setDescCallback(pt2Function descCallback);
void setPubDateCallback(pt2Function dateCallback);
```

where the argument to these functions is a function pointer defined as follows:

```
// A function pointer for callback

typedef void (*pt2Function)(char *);
```

A *pt2Function* is a pointer to a function that takes a single argument which is a pointer to a char string and returns nothing or void. Because the code in the main sketch is interested in both the title and pubDate information from the RSS XML it defines the *titleCallback* and the *pubDateCallback* functions shown below:

```
// Callback called every time a title tag is found in the RSS XML
void titleCallback(char *titleStr) {

    char buffer[TITLE_BUFFER_SIZE];

    // Make buffer empty
    buffer[0] = '\0';

    // First add a leading space char to make reading easier
    strcpy(buffer, " ");

    // Then add the title string
    strcat(buffer, titleStr);

    // Then add some trailing spaces
    strcat(buffer, "   ");

    // Scroll the composite text
    textScroller.scrollText(SCROLL_Y, buffer);
}

// Callback called every time a pubDate tag is found in the RSS XML
void pubDateCallback(char *dateStr) {

    drawCenteredText(DATE_Y, 1, dateStr, COLOR_RED, COLOR_BLACK);
}
```

Later in the *setup()* part of the code the *RSSReader* is made aware of these functions via the following:

```
// Setup callbacks for title and pubDate tags in RSS XML
reader.setTitleCallback(&titleCallback);
reader.setPubDateCallback(&pubDateCallback);
```

At runtime the following sequence of events occur:

Chapter Six - RSS News Reader

1. A request is made to a news source for its RSS information
2. The returned XML is parsed by the RSSReader code and when a title element is found, the *titleCallback* function is called and when a pubDate element is found, the *pubDateCallback* function is called.
3. The titleCallback formats the title data and sends it to the text scroller for horizontal scrolling across the LCD display.
4. The pubDateCallback simply displays the dateString passed in centered on the LCD display.

Using callbacks is a way to keep the code organized and structured and to prevent the code from becoming one big hard to maintain mess.

Conclusions

Now by using a single circuit you can have a auto setting clock, a weather clock or a RSS news reader. If you are a self proclaimed news nut like myself why don't you build one of these devices for your desk so you are never very far away from the latest headlines.

Resources

The following resources may be of interest:

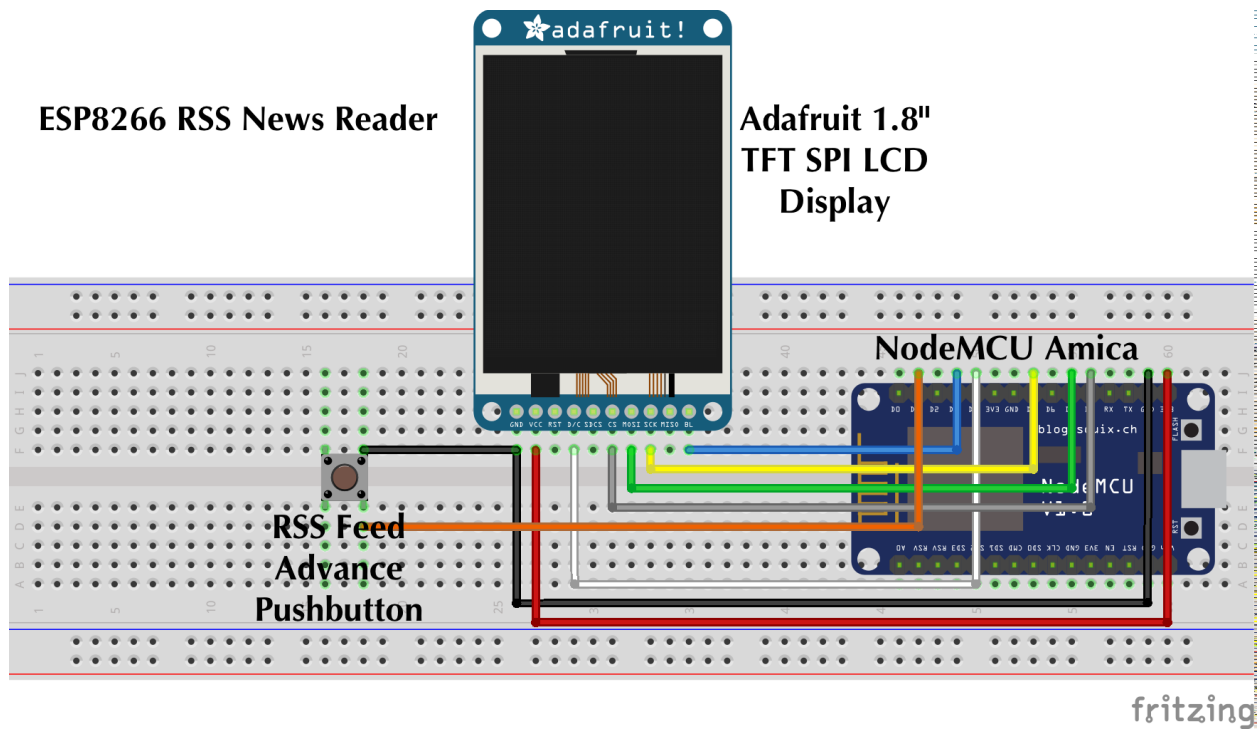
Information about XML can be found at: <http://www.w3.org/XML/>.

Information about programming the ESP8266 in the Arduino environment can be found at: github.com/esp8266/Arduino and in my four articles mentioned previously.

Information about the NodeMCU Amica can be found at: www.electrodragon.com/product/nodemcu-lua-amica-r2-esp8266-wifi-board/.

Information about the Adafruit 1.8" TFT SPI LCD display can be found at: <http://www.adafruit.com/products/358>.

Figure One
Fritzing Connection Diagram / Schematic



Chapter Six - RSS News Reader

Figure Two
RSS News Reader Breadboard

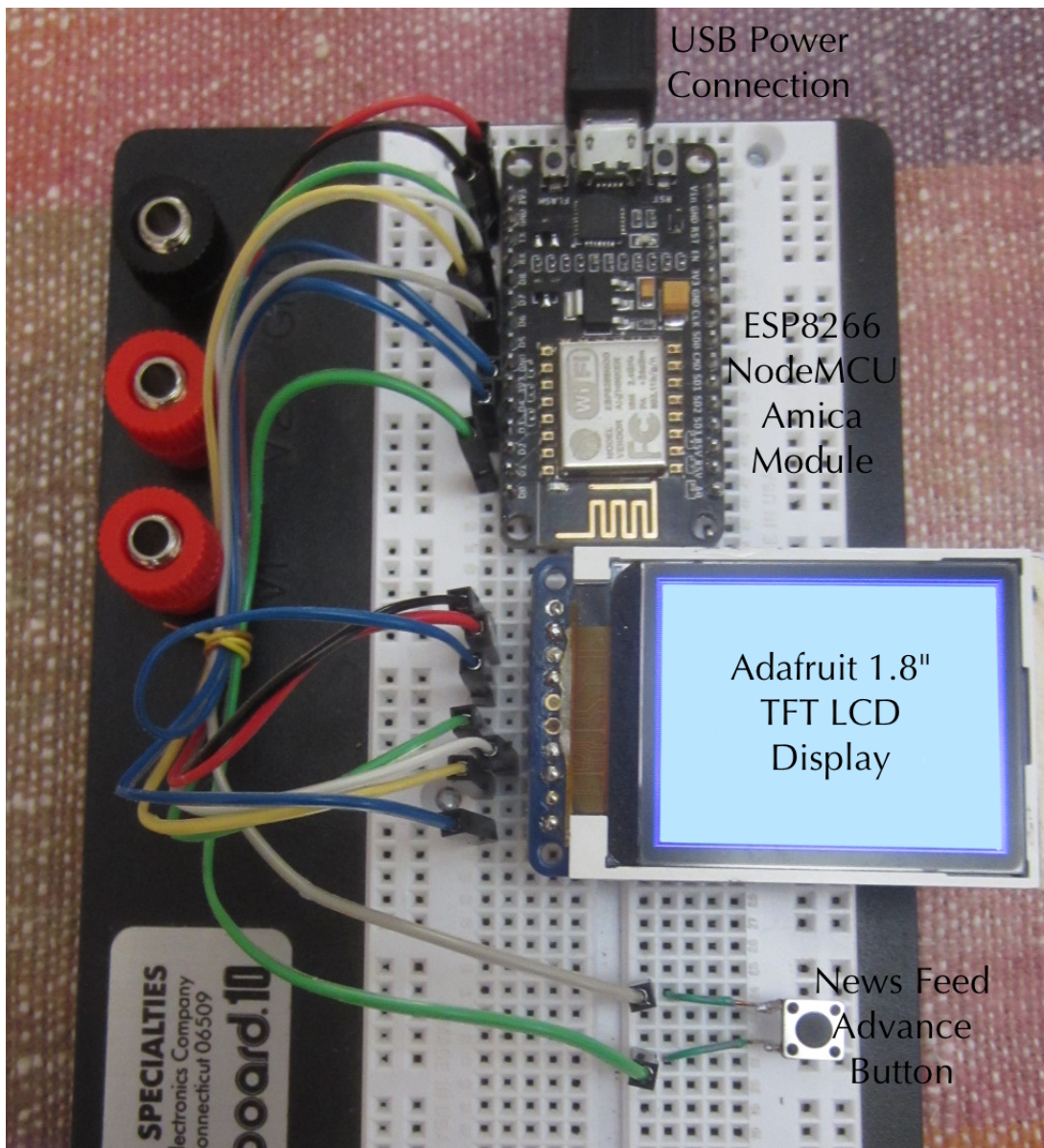
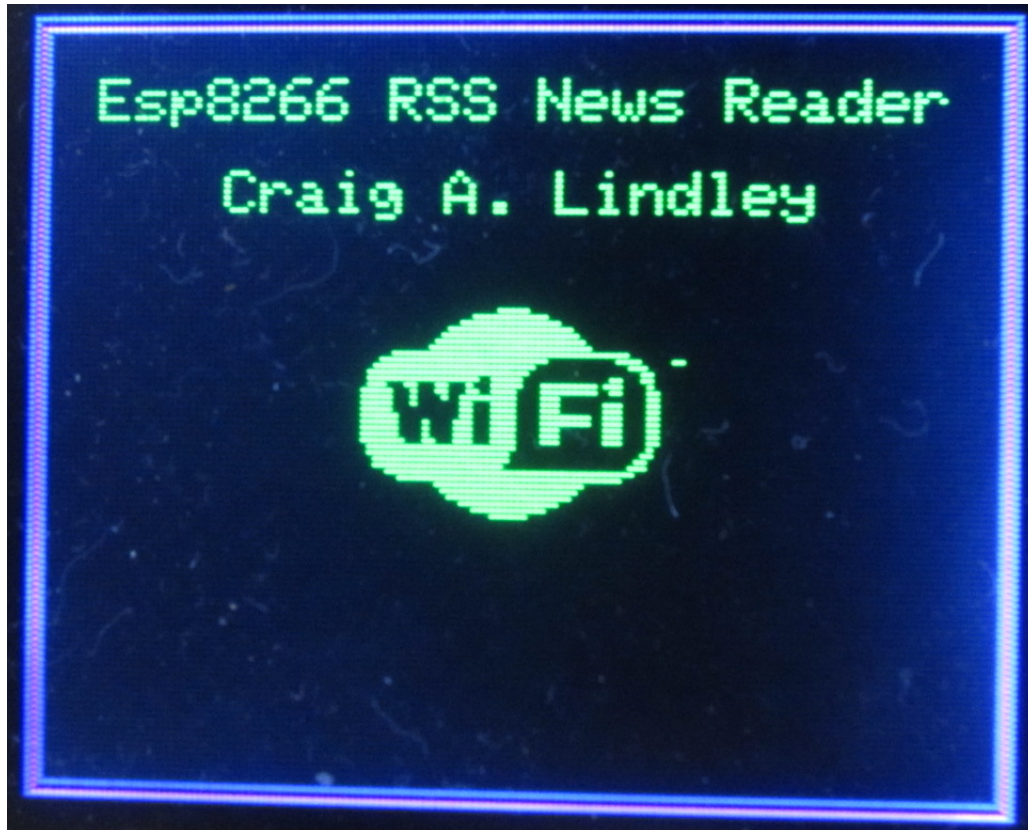


Figure Three
WiFi Login Screen

This screen will be displayed during the WiFi login process



Chapter Six - RSS News Reader

Figure Four
RSS News Reader Screen
The large font is the scrolling news headline



Chapter Seven - NeoPixel LED Tree

Introduction

Just when I thought I was done building “LED blinky” things (my house is filled with them) I see something on the Internet that catches my eye and off I go again building something new. To my friends this is now a standing joke and they, in fact, call me “Mr. Blinky”. Oh well I could be called worse.

In this case many things caught my attention and the result is the NeoPixel LED tree described in this article. The NeoPixel LED tree:

- Is beautifully made from laser cut 3 mm Baltic birch plywood.
- Is powered by a NodeMCU Amica ESP8266 32 bit WiFi enabled processor
- Has 93 individually addressable NeoPixel LEDs allowing for a large number of display patterns
- Is remote controllable from any browser on your WiFi network
- Has 50 display patterns built in including some with a Christmas theme.
- Has an *Auto* mode which randomly selects display patterns so you get a taste of every display pattern the tree has available.

I cannot take credit for much of this design because I basically merged things I found on the Internet together, and with my knowledge of the ESP8266, the result was this project. Specifically:

- I came across the mailable laser cut Christmas tree card project on hackaday.io
- I found a rather inexpensive set of NeoPixel LED rings (93 LEDs in all) for sale on Amazon
- I found software that designs laser cut flex boxes. A flex box is used for the base of the tree.
- I found a NeoPixel special effects library for the ESP8266 that provided most of the display patterns and an example program from the same library that was the basis of the tree remote control program I wrote.
- I used the ESP8266 WiFiManager library to allow the tree to connect to any WiFi network without having to hardcode WiFi credentials into the program. This is very handy when you move the tree between WiFi networks or locations.
- I used the ESP8266 ArduinoOTA (OTA stands for Over The Air) library that allows the tree's software to be updated wirelessly without having to physically connect a USB cable. This helps prevent damage to the tree because updates are hands off.

Links to all of the stuff I found on the Internet are provided in the *Resources* section.

Laser Cutting Tree Pieces

When I first came across the laser cut Christmas tree card project I quickly fell in love with it. Each piece of the tree (see Figure One) is like a snow flake made out of wood. Also, the light color of the

Chapter Seven - NeoPixel LED Tree

Baltic birch contrasted nicely with the brown burn marks caused by the laser cutter.

Which brings me to the laser cutter. Where was I going to find one to use or was I going to have to pay some company to cut the tree out for me? Luckily it was brought to my attention that many of the public libraries in the Pikes Peak region where I live have Maker Spaces with laser cutters (and a lot of other equipment) that are available to the public to use under the supervision of library staff. This was fantastic in that not only was using the laser cutter free to use (you must supply your own materials) but there were knowledgeable and friendly staff on hand to help me get going. In preparation for using the laser cutter I was given a quick talk on safety around the laser cutter and shown around a display that showed the many different materials the laser cutter can be used on including: stone, brick, wood, paper, plastics, fabric, etc. I may try to engrave some stones later for Christmas presents. With the prerequisites out of the way, my library card was stamped allowing me to use the maker space equipment whenever the space is open. When I then saw the Epilog Zing laser cutter (see Figure Two) in action I immediately gained an appreciation for the capabilities and the precision possible with this device.

The library's Epilog laser cutter is driven by a program called Corel Draw which I thought was long extinct but I was mistaken. Current versions of Corel Draw have many features that make it suitable for driving a laser cutter. I have been successful in using design files in both svg and ps (postscript) formats with Corel Draw. Using Corel Draw with the laser cutter is pretty easy once you have done it a few times. I had some initial issues but got them worked out with the help of the library staff. The Epilog laser cutter comes with a handy chart that tells you how to set the power level and the speed of laser for the materials you are using. The 30 watt laser easily cuts through the thin birch plywood used for the tree and is amazing to watch.

The parts required for the tree are as follows:

Quantity	Description	Size	Source
2	Baltic birch for tree. Half of the tree is laser cut from each piece	6" x 8"	Woodcraft, Home Depot, Lowe's or any wood supply store
?	Baltic birch for the flex box used for the base of the tree	6" x 10"	Same as above
1	Round Aluminum Tube 6 mm outside diameter	About 5 3/4" long	Ebay, hobby shops

You may have noticed the question mark for quantity in the table above. The reason for this is the laser cut flex box base is very fragile due to flexible corners the box's have so I ended up having to cut five of these boxes before I got one that didn't break during assembly. The secret is to completely soak a paper towel with water and then wrap the three pieces that make up box together and place them in a microwave for about a minute. Set the wrapped wood aside for a couple of minutes after removal from the microwave to allow the plywood to thoroughly steam before attempting assembly. Next very carefully stretch and hold the long piece that makes up the box sides a couple of millimeters to widen

the wooden corners. Then proceeding very slowly, wrap the side piece around the bottom piece locking the tabs in place as you go. Once you get the wrapping done, place the top piece onto the box and sit the box aside and allow it to dry completely before proceeding. Once the box is dry it will retain its shape so you can take the top or bottom off to work on the electronics that will be placed inside.

I believe the small size of the flex box base is partially responsible for its fragility. Larger flex boxes probably wouldn't be quite as fragile because there would be more material in the flexible corners. I have included the postscript design file for these flex boxes in the zip file available from the Nuts and Volts website associated with this article. The general design file called *flexbox_v.1.3.ps* can be opened in any text editor and the boxlength, boxwidth and boxheight variables can be changed to produce boxes of any size. I have also include *treebase.svg* which is the design for the tree base used here. This file was created by copying *flexbox_v.1.3.ps* to a different file, opening it up in a text editor and then setting the user defined parameters as follows:

```
% Define box parameters -- This is the size of the box you want
/boxlength 3.0 inch def          % long dimension of flat side of the box
/boxwidth 1.5 inch def           % dimension across the hinge
/boxheight 2.5 inch def          % short dimension across flat side of the box
/cornerradius 0.5 inch def       % radius of the hinge
/materialthickness 0.125 inch def % thickness of the material which will become the box
```

I then used a free online service to convert the modified postscript file to an svg file because I thought that was necessary for use with Corel Draw. I have since discovered Corel Draw can work just as well with the original postscript file so no conversions are necessary. The design for the base is shown in Figure Three.

Laser cutting the tree was much easier and successful on the first attempt. I cut two copies incase I damaged any of the pieces. The design for the tree is shown in Figure Four. Once cut the tree pieces are fragile and need to be handled with care. I should note that not all of the pieces from the tree design are used in the NeoPixel LCD tree. Only sections 1, 3, 4, 6 and 8 and the star are used. The other laser cut pieces could be used for Christmas tree decorations or perhaps even ear rings.

The Electronics

The electronics used in the tree couldn't be much simpler nor easier to put together. The required parts are shown below:

Item	Description	Source
NeoPixel ring set	Mokungit 93 Leds WS2812B WS2812 5050 RGB LED Ring Lamp Light with Integrated Drivers. Includes rings of 32, 24, 16, 12 and 8 LEDs along with a single stand alone LED	Amazon
NodeMCU Amica module	32 bit uC with WiFi interface	Electrodragon.com

Chapter Seven - NeoPixel LED Tree

Item	Description	Source
USB power supply	Capable of 5 VDC @ 2.5 amps minimum	Amazon, RadioShack
Capacitor	1000 uF @ 35 volts	RadioShack
Capacitor	.1 uF	RadioShack
Capacitor	.01 uF	RadioShack

The NeoPixel ring set is shown in Figure Five. The schematic for the tree electronics is shown in Figure Six.

Building The Tree

Before beginning assembly of the tree the tree sections are glued to the NeoPixel rings as shown in Figure Seven. I glued the single NeoPixel to the tree's star as well. I should note that the center hole of each tree section is slightly smaller than the 6 mm support tube. I used a reamer to carefully enlarge each hole to allow them to slide down the tube into position.

The tree is built from the base up. I started by fitting parts X, Y and Z together which make up the tree's support structure. I used a small file to slightly taper the tabs so they would fit into the round disk X. With that done I glued the round disk to the top of the flex box and then glued parts Y and Z to the disk. I inserted the support tube into this assembly to keep the parts correctly aligned while the glue dried. Once dried I drilled a small hole in what will be the back of the tree (Figure Eight) for the three wires that connect the NodeMCU module in the base to the large NeoPixel ring which makes up the first tier of the tree. Within the base I glued the NodeMCU module in place and then wired up the electronics as shown in Figure Nine. With that done I reassembled the flex box base so it could support the tree as it was being built.

The tiers of the tree are separated using laser cut spacers “a” and “b” where the “a” spacers have a slightly larger outside diameter than the “b” spacers. I used “a” spacers towards the bottom of the tree and “b” spacers towards the top. I used five spacers between the base and the first tier, seven between the first and second tiers, seven between the second and third tiers, 6 between the third and fourth tier, five between the fourth and fifth tiers and five as well between the fifth tier and the star. I covered the spacers and wires between the tiers of the tree with heat shrink tubing as it was important to hide the wires as much as possible. Figure Ten shows this in detail. The spacers had to be reamed as well so they would fit over the support tube.

Wiring the NeoPixel rings neatly was somewhat difficult to do. Four wires must be soldered onto the little pads of the NeoPixel rings. Each ring has a data input pad (DI), a 5V pad, a GND pad and a data out (DO) pad. The DO pad of a lower ring must connect to the DI pad of the upper ring and the 5V and GND connections are paralleled all the way up the tree. The DO of the 8 NeoPixel ring is wired to the DI pad of the single NeoPixel LED glued to the tree's star. There is no connection from the single NeoPixel LED's DO pad.

Care must be taken when soldering these connections as it would be very easy to lift these pads if too much heat were applied. Figure Eleven shows the completely assembled tree in operation.

Tree Software

The software for the NeoPixel LED tree was developed using the Arduino IDE. I used version 1.8.0 for MacOS but you should be able to use the Arduino IDE on Windows as well. See my previous articles or the *Resources* section for how to set-up the Arduino IDE on your computer for targeting ESP8266 type devices.

To use the OTA feature of the Arduino IDE you must allow the IDE to accept remote connections. How this is accomplished varies depending upon the computer platform you are running on but usually involves a change to your computer's firewall settings. Google “Enabling ArduinoOTA <your platform>” to get the instruction appropriate for your computer.

The NeoPixel LED tree software should be available on the Nuts and Volts website in association with this article. The file is called: *Lindley_ESP8266NeoPixelTree.zip*. To use this software, unzip the file and copy/move the *ESP8266NeoPixelTree* directory into your computer's Arduino directory. The zip file also contains the versions of the libraries I used during program development. It is important to use these versions as different versions may not function correctly. These library zip files should be unzipped and then moved into your arduino/libraries directory for use. It is best to install new libraries before the Arduino IDE is started.

Whereas the electronic for this NeoPixel LED tree borders on the trivial, the software/firmware for the tree is a bit more involved. The files which make up the program are described below:

File	Description
ESP8266NeoPixelTree.ino	The main program file that drives the NeoPixel LED tree.
index.html.h	Miscellaneous HTML snippets that support the web UI
main.js.h	Javascript functions used in the web UI

In addition to the files above, the following Arduino libraries are required:

Library	Function	Source
ArduinoOTA	Allows the NeoPixel LED tree code to be remotely updated after the initial upload.	Provided with the esp8266/Arduino platform code
WiFiManager	Allows the NeoPixel LED tree to change WiFi credentials without having to make a code change.	https://github.com/tzapu/WiFiManager
WS2812FX	A NeoPixel LED special effects library for the	A modified version of this

Library	Function	Source
	ESP8266.	library is included in the article download from Nuts and Volts.

Once you have the code and libraries in place, you must connect the NodeMCU device in your tree to your computer using a USB cable. Next select the *NodeMCU 1.0* board type from the Arduino IDE and the port that is used to connect to your NodeMCU device. Once that is completed, you can compile the code and if error free you can upload the code to your tree. Once this initial upload is successful you can disconnect the USB cable and do any further updates remotely.

Remotely Controlling The Tree

Before you can remotely control the NeoPixel LED tree, you must first provide WiFi credentials so the tree can connect to your local WiFi network. If the ESP8266 has not been connected to the WiFi network previously it will create a wireless access point called, *NeoPixelTreeAP*, that you must connect to with your computer. Once that is done, with a browser go to 192.168.4.1 and you will be presented with a page for assigning credentials. When you click on the SSID of your network and then specify the password, the ESP8266 should take down the access point and attempt to establish a connection to your WiFi network. You can monitor the status of this process if you have the Arduino IDE's serial monitor open. Once you establish a connection to the WiFi network the ESP8266 will remember the credentials going forward. You should never have to go through this process again unless you move the NeoPixel LED tree to another network or location.

With WiFi setup complete, change your computer back to your normal WiFi network and navigate your browser to 192.168.0.3. If all is well you should see the web page (see Figures Twelve and Thirteen) you will use to control the NeoPixel LED tree. In the middle of the page is a long list of display pattern buttons that you can click on. If a display mode allows its color to be configured you can click on the hue strip on the left to set a color and you can click on the middle strip to set the color's saturation. At the bottom of the page you can change the overall brightness of the tree and the speed at which the display patterns run. The *Auto* mode selects a new display pattern randomly every 30 seconds. It also selects a random color, random brightness and random speed the display pattern will run with. The *Off* mode makes the tree appear to be off but it is still listening for display pattern changes from the UI.

Final Thoughts

Did the world need yet another blinky thing? Probably not, but when has that ever stopped me from building anything. The NeoPixel LED tree is a beautiful addition to my collection of blinky things and would make a great gift for someone who likes such things. Show a child how to control the tree via a browser and watch them sit for a long time clicking the screen and seeing how the tree reacts. It initially surprised me how fast the tree reacts to changes made in the browser; it is virtually instantaneous. Even without the LEDs lit up the laser cut tree is kind of nice to look at as the wood has nice burned edges.

In its current form the NeoPixel LED tree is for decoration/fun only. There may be, however, some

serious uses for such a device. The tree could be used as a night light in a young ones room. With changes to the software the tree could be used for the “Thinking Of You” device I wrote about in the November 2015 issue of Nuts and Volts. It could also be repurposed as a display showing whether the stock market is up or down. It might also be programmed to inform you when it is time to standup and take a break or to go home from work. The possible uses are almost endless.

This was a fun though rather challenging project to build mainly as a result of never having used a laser cutter before and the problems I had with the flex box base. I learned a lot about using a laser cutter that I am sure I can apply to future projects. I also learned that I would like to own a laser cutter but that is another issue all together.

Resources

Information about the laser cut Christmas tree can be found at: <https://hackaday.io/project/9222-laser-cut-mail-able-christmas-tree>

Information about laser cut flex boxes can be found at: <http://www.thingiverse.com/thing:17240/#files>

Information about Corel Draw can be found at: www.coreldraw.com

Information about programming the ESP8266 in the Arduino environment can be found at: github.com/esp8266/Arduino.

Information about the NodeMCU Amica can be found at: www.electrodragon.com/product/nodemcu-lua-amica-r2-esp8266-wifi-board/.

Information about the WiFiManager can be found at: <https://github.com/tzapu/WiFiManager>

Information about ArduinoOTA can be found at: http://esp8266.github.io/Arduino/versions/2.0.0/doc/ota_updates/ota_updates.html

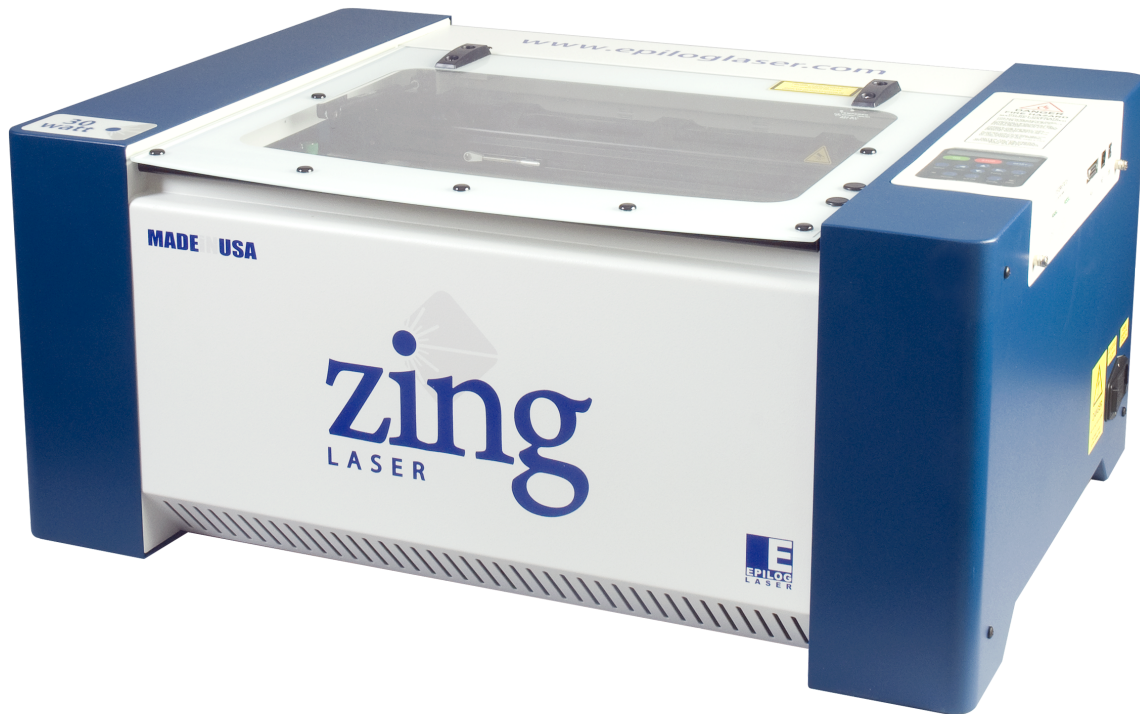
Information about the NeoPixel WS2812FX special effects library is available at: <https://github.com/kitesurfer1404/WS2812FX#ws2812fx---more-blinken-for-your-leds>. Note this project uses the modified version of this library available from the Nuts and Volts website.

Chapter Seven - NeoPixel LED Tree

Figure One
Laser Cut Tree Parts Resemble Wooden Snow Flakes
As noted in the text not all pieces are used



Figure Two
Epilog Zing Laser Cutter
Available for public use at my local library and maybe yours too



Chapter Seven - NeoPixel LED Tree

Figure Three
Tree Base Design

Base is 3" wide x 2.5" deep x 1.5" tall and is laser cut from 3 mm Baltic Birch

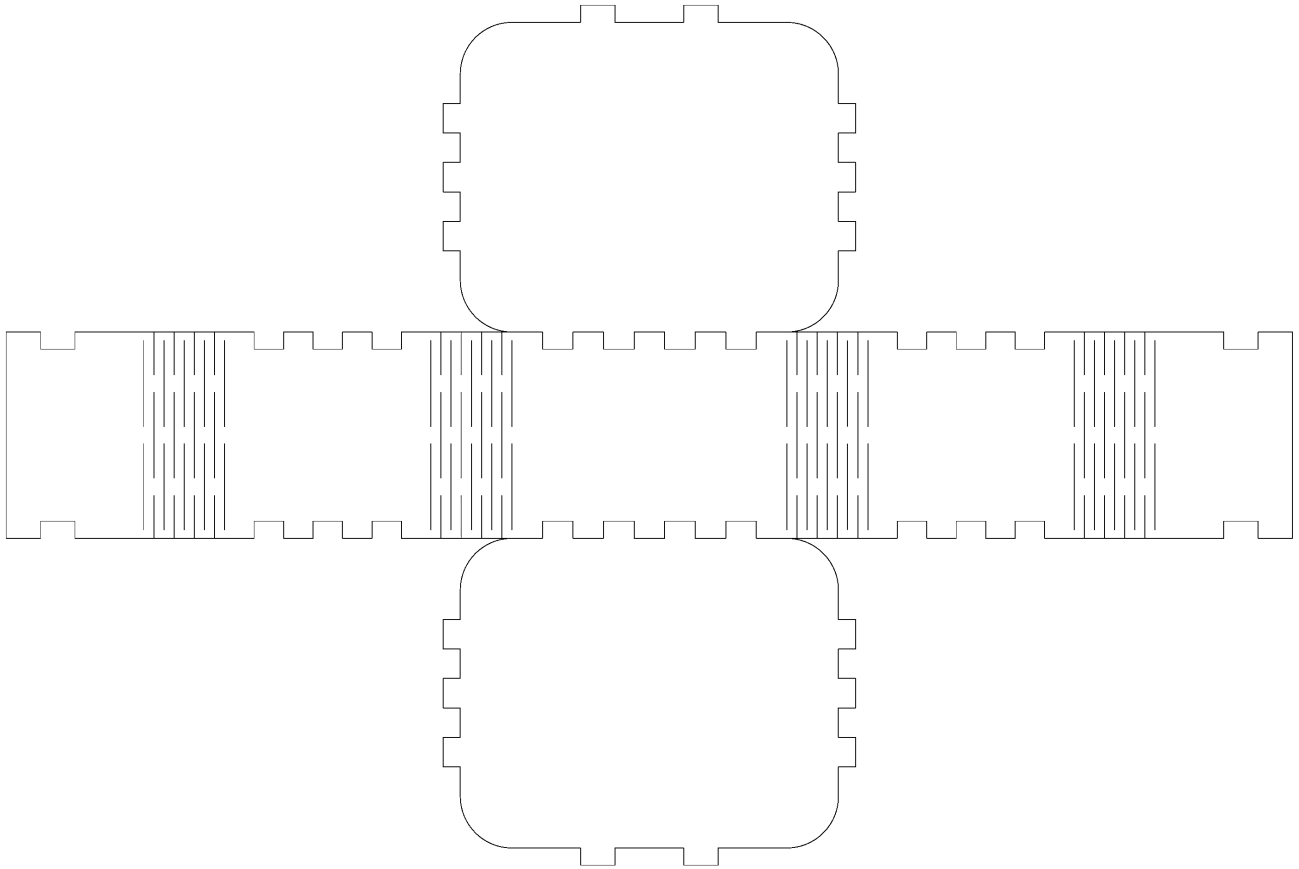
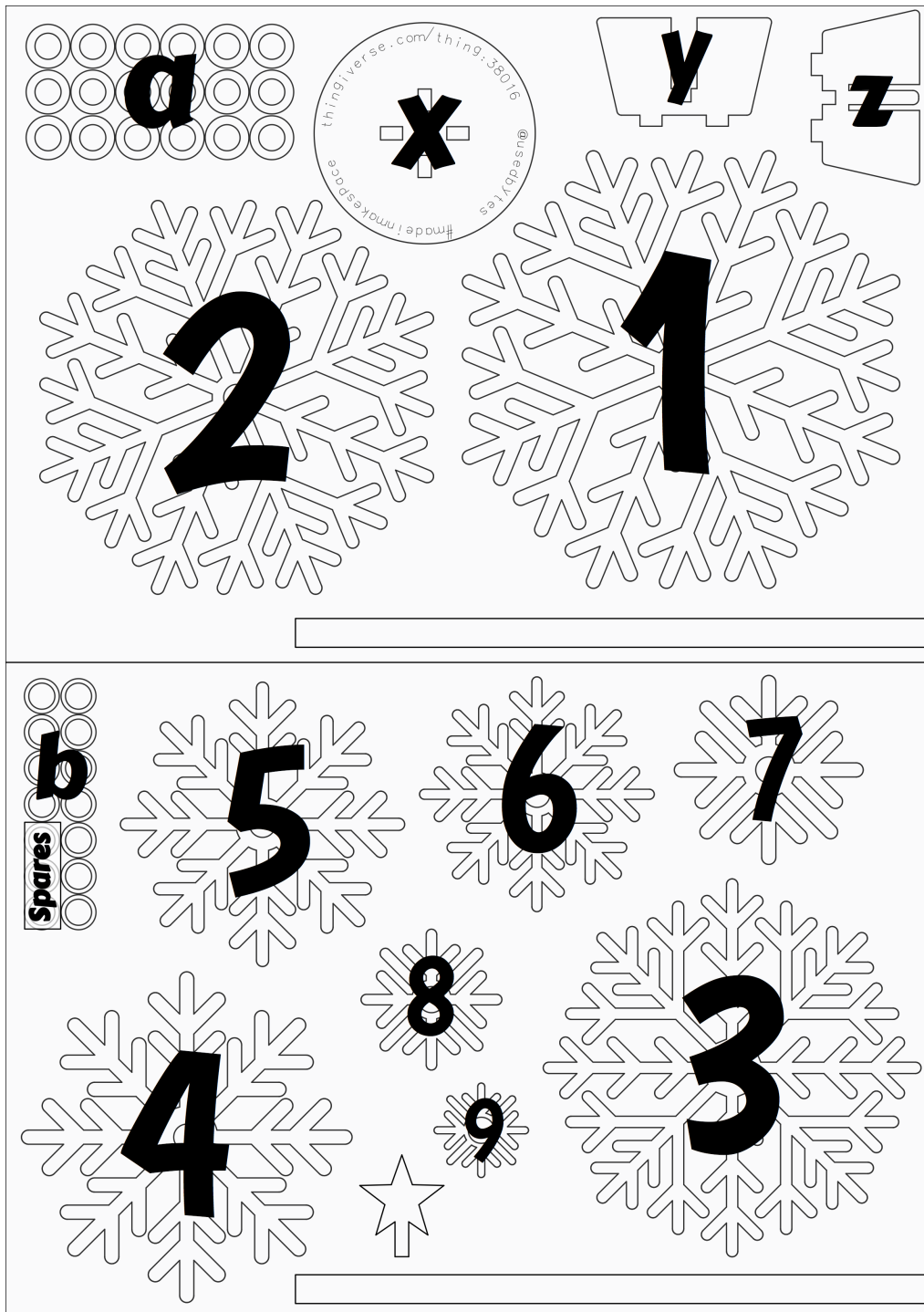


Figure Four
Tree Design from Thingiverse.com with items labeled



Chapter Seven - NeoPixel LED Tree

Figure Five
NeoPixel Ring Set from Amazon

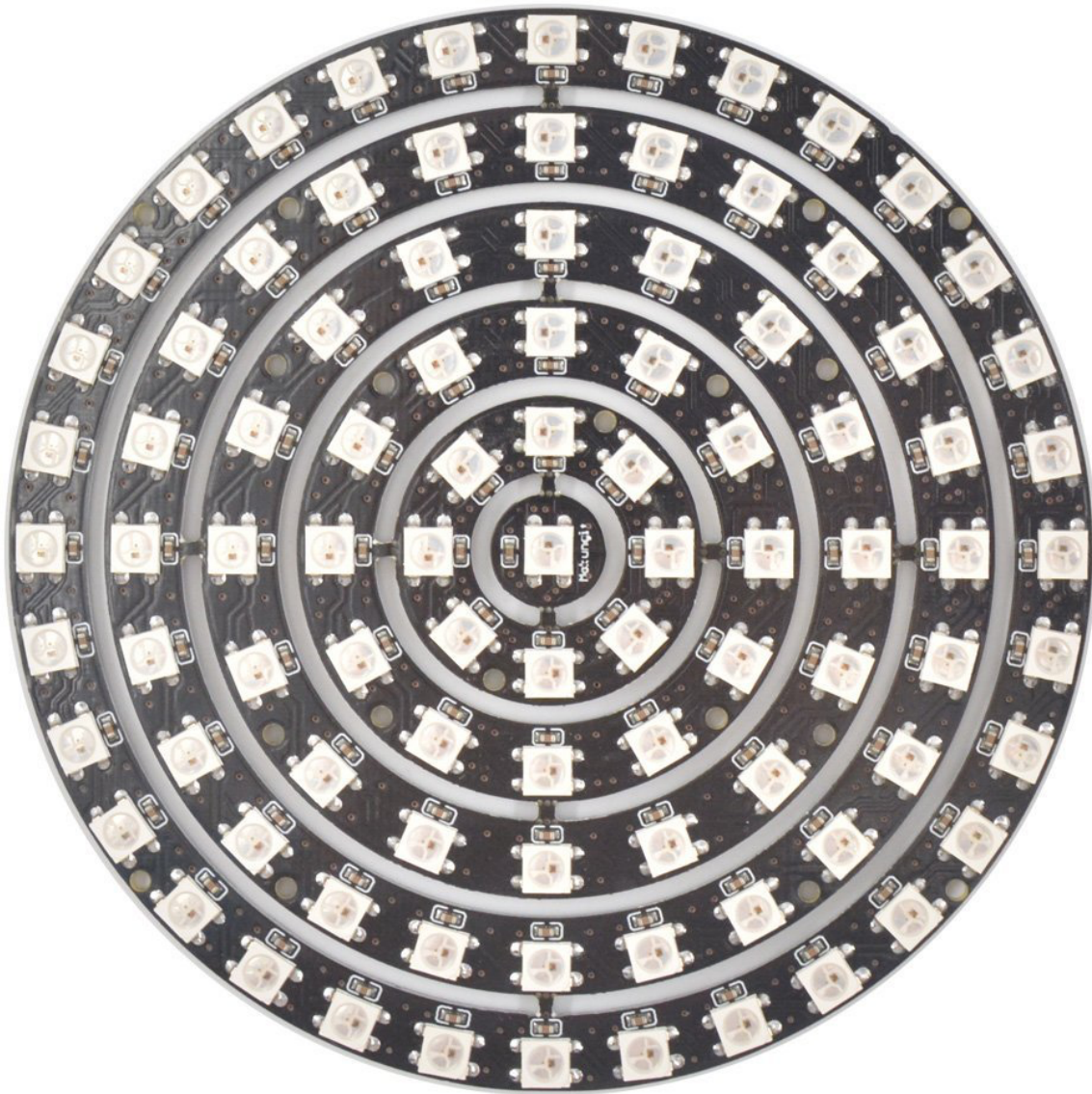
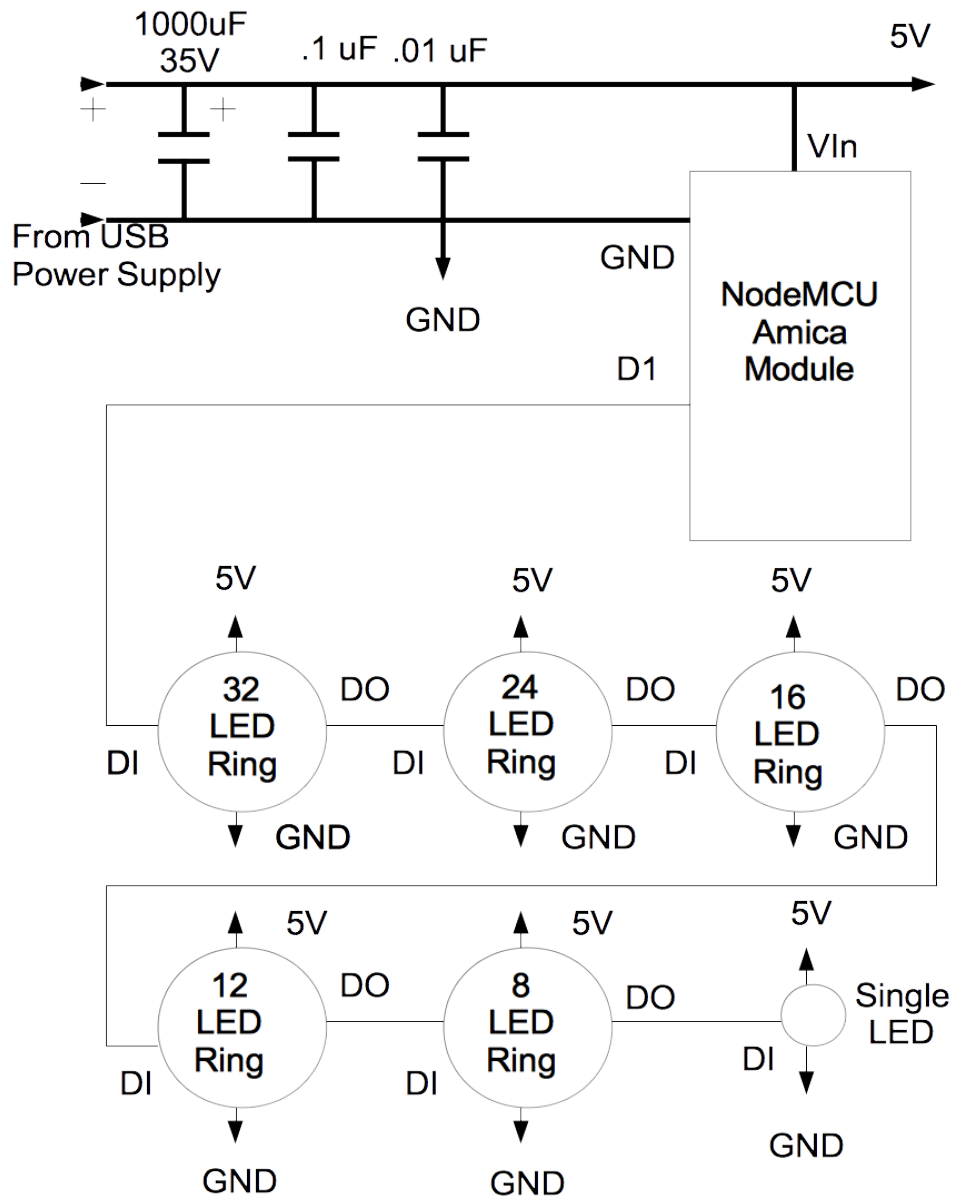


Figure Six
Schematic Diagram



Chapter Seven - NeoPixel LED Tree

Figure Seven

LED rings attached to tree segments with Goop glue
NodeMCU Amica device glued to base top as well
Base itself doesn't require glue to hold together

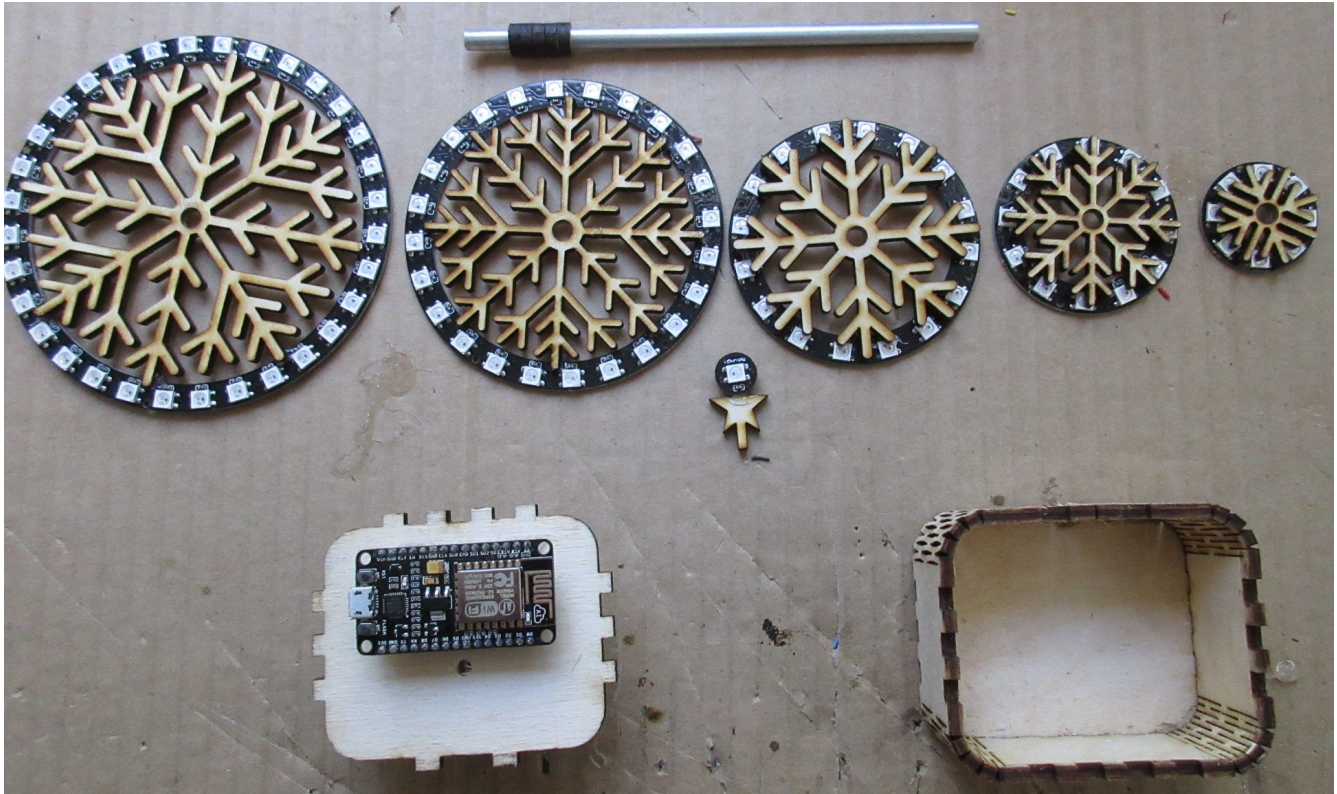


Figure Eight
Tree Base Detail

Note the hole into the base for the wires. Heat shrink tubing is used to hide the wiring.
USB power supply wire extruding thru hole in back of base

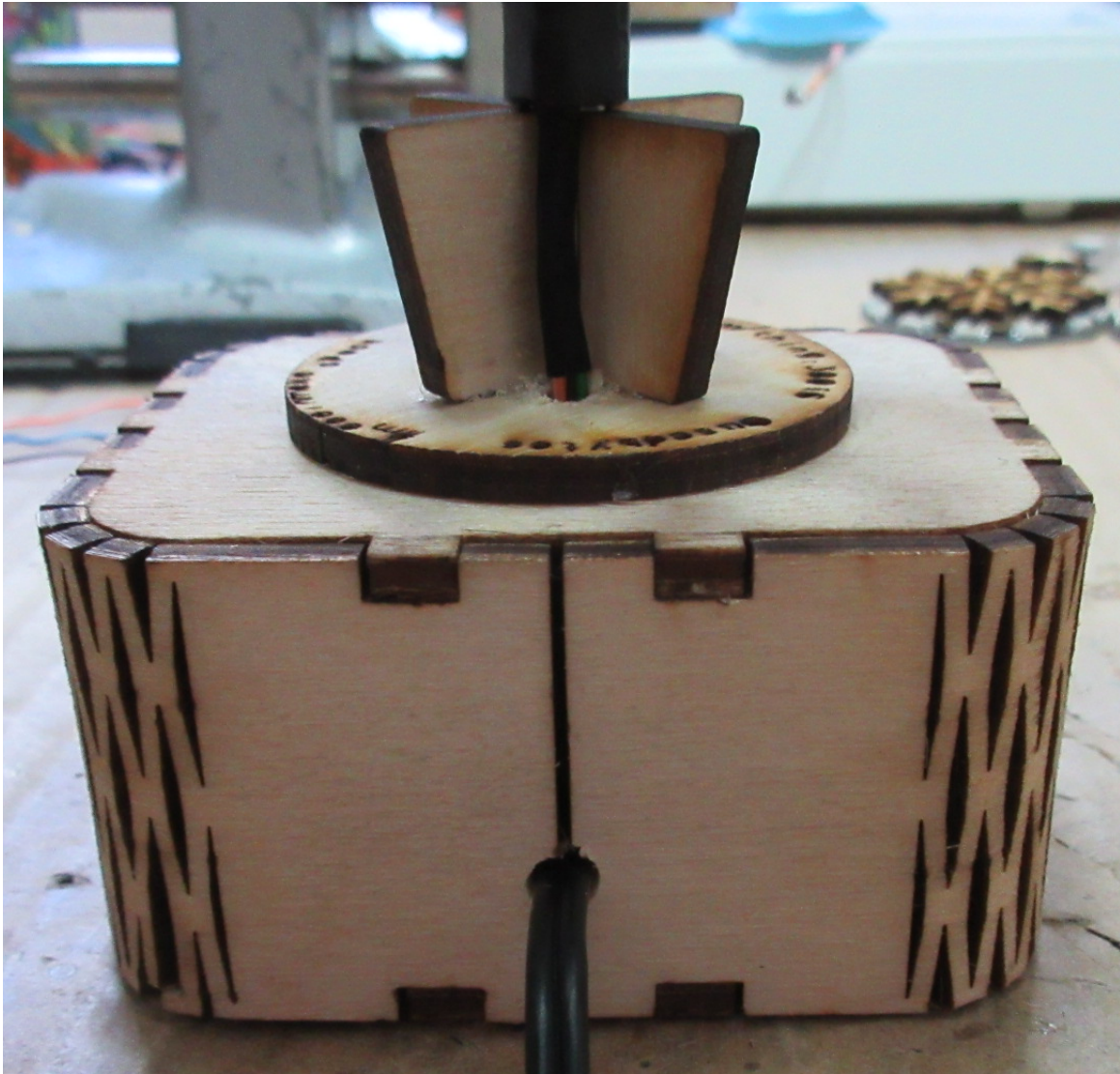


Figure Nine
Base Electronics
Black wires from USB power supply

Black wires from USB power supply

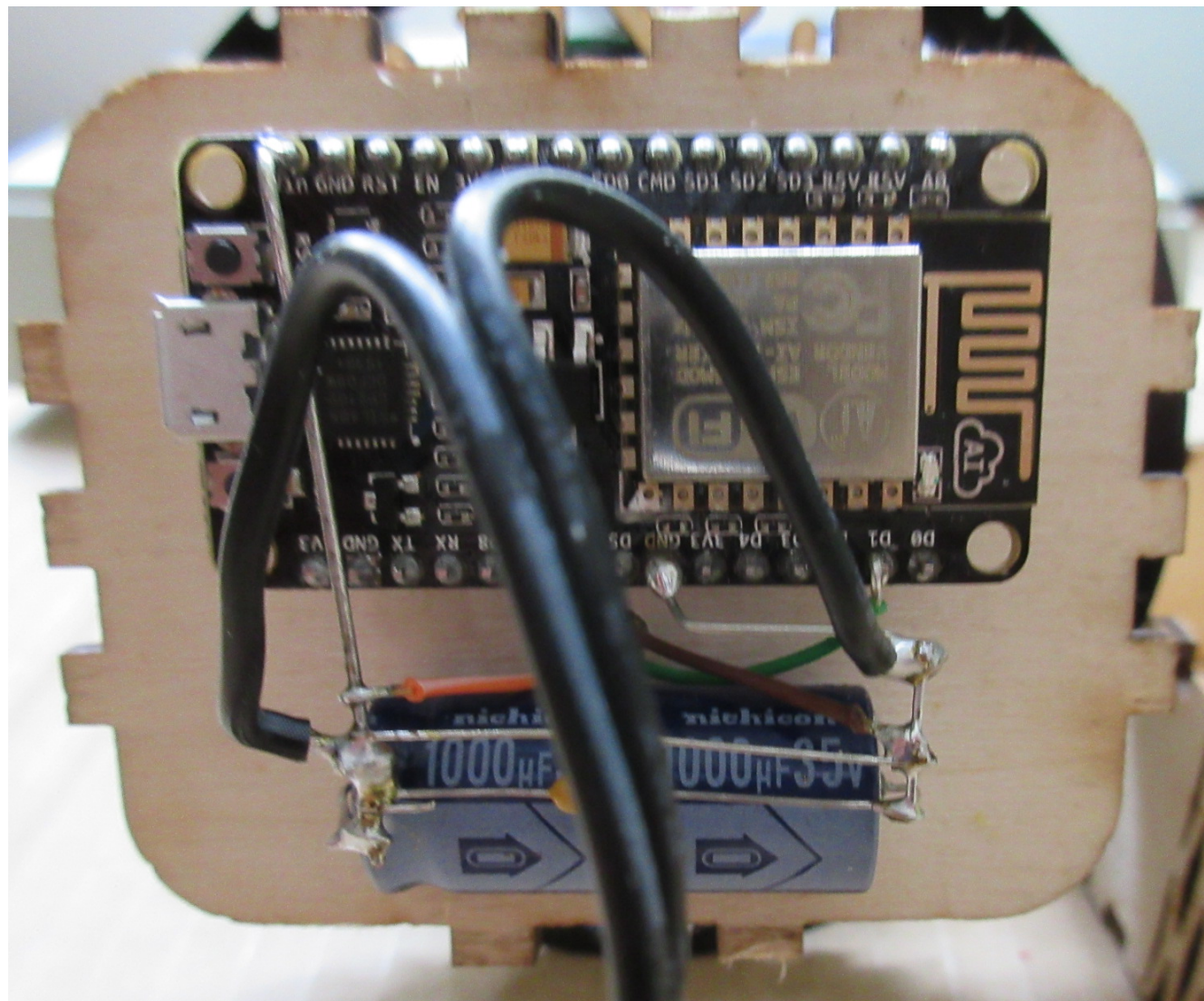
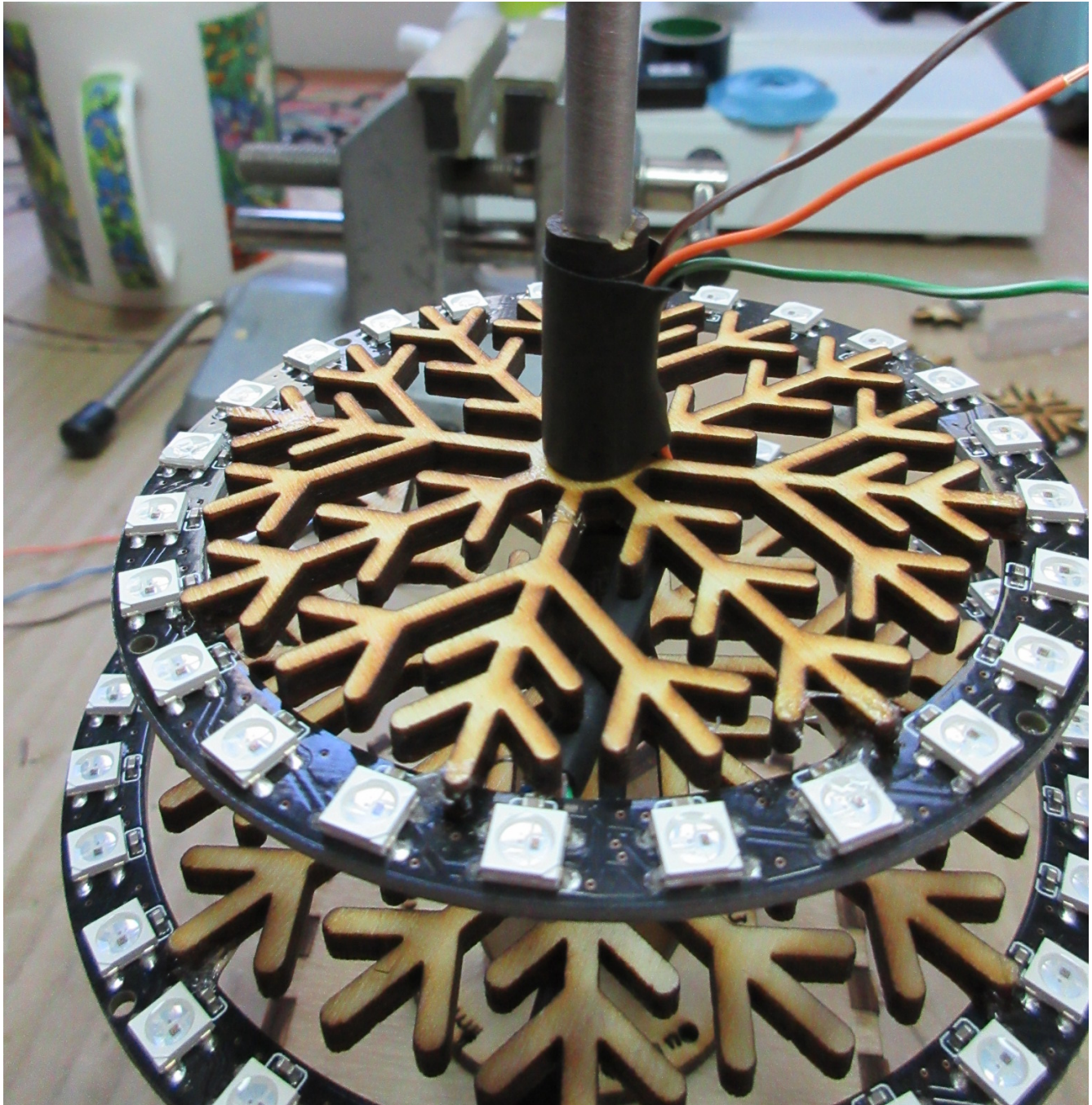


Figure Ten
Assembly Detail Close Up
Heat shrink tubing used to hide spacers and wiring between sections

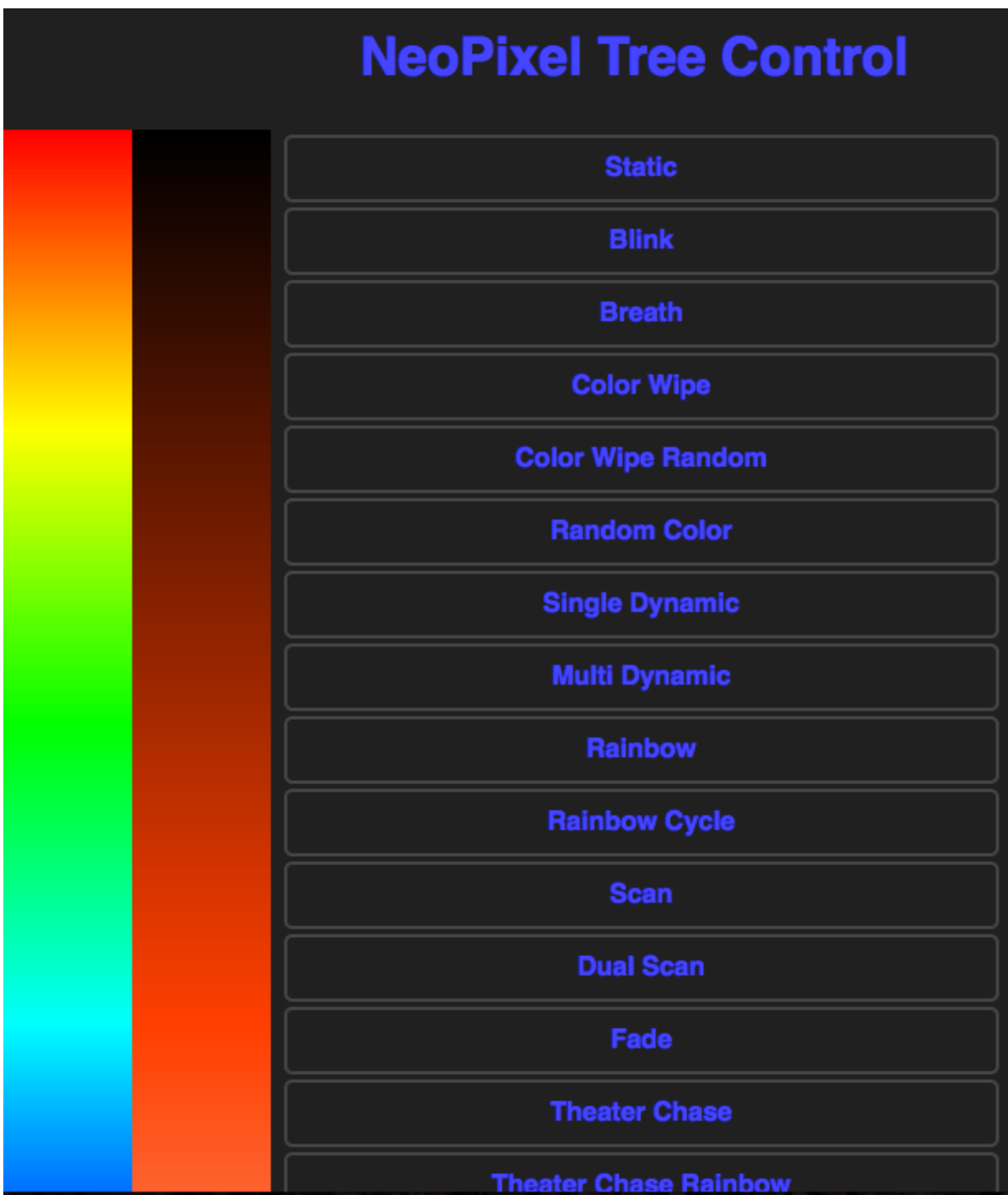


Chapter Seven - NeoPixel LED Tree

Figure Eleven
Finished and Operational NeoPixel LED Tree

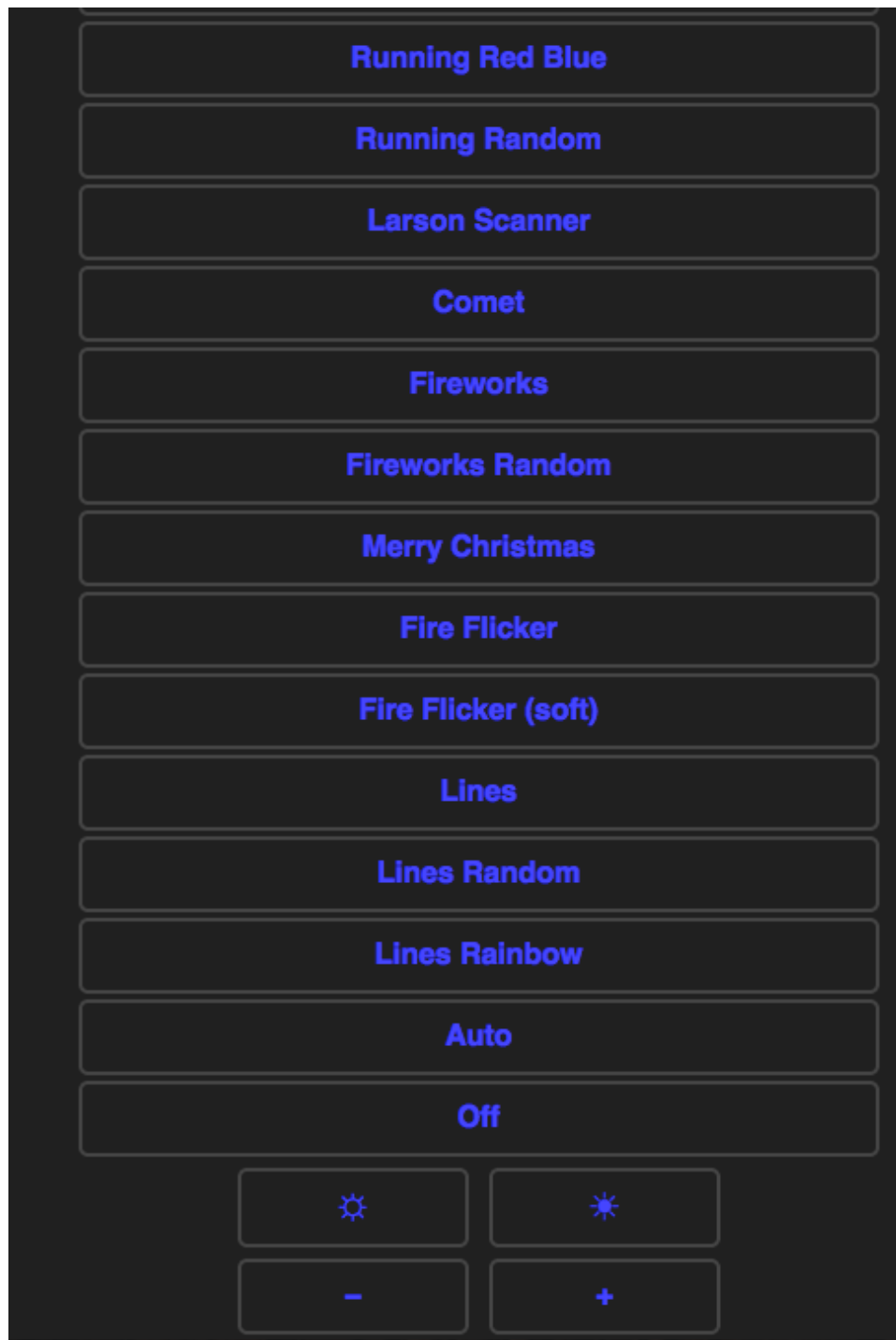


Figure Twelve
Top Portion of Web UI for controlling NeoPixel LED tree



Chapter Seven - NeoPixel LED Tree

Figure Thirteen
Bottom Portion of Web UI for controlling NeoPixel LED tree



Chapter Eight - Thinking Of You

Introduction

My family and friends are increasingly spread out across the country or even across the globe and keeping in touch can be a problem. This is especially true for someone like myself who doesn't really like to talk on the phone much. In addition, there are problems with people's hectic life styles and conflicting schedules. When you think of your friend or family member you may be too busy to call them that instant or if you do they may be too busy talk. Thus the spontaneity of the thought can and usually does fade away.

I was thinking about these issues in relation to my sisters who are located across the US. I wanted something that would virtually instantaneously let my sisters know I was thinking of them but that was totally non intrusive. I wanted them to know without having to disrupt what they are doing with a phone call or text message. And vice versa, when the thought struck them they could let me know they were thinking of me as well.

With these thoughts in mind I designed what I call a **Thinking of You** or ToY device. These are small Internet connected devices meant to be used in a home, apartment, work place or anywhere else with a fixed Internet connection. They should be placed on a desk at the office or in the living room at home where they can easily be seen and interacted with. Each ToY device has a single push button switch and an RGB LED for user interaction.

ToY devices were designed to be used in groups. For example my family's group is made up of three ToY devices: one for myself and one for each of my two sisters.

Each ToY device is programmed with WiFi info for each group member and assigned a colorful lighting pattern for the RGB LED that identifies the member within the group. When these devices are powered up (they are meant to be on all of the time) they connect to the local WiFi network and wait. When someone in the group presses their button (indicating they are thinking of others in the group) his/her pattern is displayed on each device no matter where they are in the world. When other members of the group press their buttons in response, their patterns are appended to the pattern being displayed so all members of the group know who responded. The ToY devices then continuously displays the concatenated patterns for 30 minutes and then extinguish themselves until someone starts the process again.

The ToY devices are built using an inexpensive module called a NodeMCU Amica that incorporate a ESP8266-12 WiFi module with embedded application processor. I highly recommend you read my previous article "Meet the ESP8266: A Tiny, WiFi Enabled, Arduino Compatible Micro Controller" from the October 2015 issue of Nuts and Volts for background information on ESP8266 devices.

ToYs are really cool, easy to build, little devices that I plan on giving my sisters for Christmas. It is a good thing my sisters don't read Nuts and Volts or the cat would now be out of the bag. Maybe you should consider building these for your family as well.

As a side note, I had some friends who live across town help me test this concept. I provided them with a ToY device and I had one as well. As we are both early risers it always made me smile that in the early morning one of us would press the button on our device and the other would reciprocate and our ToY devices would happily begin pulsating with our assigned patterns letting each other know we were awake.

Hardware

The hardware which makes up a ToY device is really quite simple and can be built by anyone with a little soldering experience in an hour or two. The parts required for each ToY unit are shown in Figure One and the schematic in Figure Two.

The NodeMCU Amica module, shown in Figure Three, is what makes this simple build possible. This module consists of a ESP8266-12 along with the support circuitry required to make this module an excellent choice for Internet of Things (IoT) projects and WiFi development.

The ESP8266 is a whole family of WiFi modules which vary in the number of available I/O pins, the amount of onboard memory, the types of interfaces available and in how the RF antenna is attached/implemented. The ESP8266 module on the NodeMCU Amica has its RF antenna etched directly onto the circuit board.

Information on the whole family of ESP8266 devices is available at:
www.esp8266.com/wiki/doku.php?id=esp8266-module-family.

The following attributes of the ESP8266 family were extracted from the data sheet available at:
nurdspace.nl/File:ESP8266_Specifications_English.pdf.

- 802.11 b / g / n
- Wi-Fi Direct (P2P), soft-AP
- Built-in TCP / IP protocol stack
- 802.11b mode + 19.5dBm output power
- Built-in temperature sensor
- Supports antenna diversity
- Off leakage current is less than 10uA
- Built-in low-power 32-bit CPU which can double as an application processor
- SDIO 2.0, SPI, UART , ADC
- Standby power consumption of less than 1.0mW (DTIM3)

In other words, the ESP8266 family of modules features low power consumption, high RF power output and is capable of supporting all of the current 802.11 standards required for WiFi connectivity. In addition, it supports many industry standard hardware interfaces and can function as the application processor in many designs as it does in this one. The ESP8266 is a 3.3 VDC part.

The NodeMCU module comes from the factory programmed with the LUA programming environment

but since the ToY code was written in the Arduino environment we will overwrite the factory programming for this application.

The ToY device uses one digital input pin for monitoring the request push button switch and three digital output pins for driving a common cathode RGB LED. Current limiting for the LED is implemented using 1K ohm resistors for each of the three colored LEDs which make up the RGB LED.

To read the switch and to drive the RGB LED we must establish a mapping between the general purpose I/O pins Arduino expects and the physical pins on the NodeMCU module. Figure Four which shows the NodeMCU pinout helps with this task.

The mapping I have chosen for this design is shown below.

Function	Configuration	NodeMCU Module Pin	Arduino GPIO
Request switch	Digital input with pull-up	D1	GPIO 5
Red LED drive	Digital output for PWM	D4	GPIO 2
Green LED drive	Digital output for PWM	RX	GPIO 3
Blue LED drive	Digital output for PWM	D2	GPIO 4

The ToY software configures these four I/O lines for the functions shown above.

Power for the ToY module is supplied by an USB power supply/charger capable of at least 1 amp @ 5 volts DC. A weak power supply will cause the hardware to operate erratically if at all so make sure the power supply you use is up to the task.

Configuration

ToY devices must be configured before they will function as a group. For each member of the group the following information is required.

1. A name to be associated with the ToY group member. This is only used in debugging messages available through the Serial Monitor.
2. The SSID or network name of the WiFi network the device will be associated with.
3. The password of the WiFi network the device will be associated with.
4. A Teleduino key
5. The selection of a LED lighting pattern for the device.

Each ToY device in a group must have a unique key and a unique LED pattern. Also each device must exist on a different WiFi network. With the current software, only one ToY device can exist on a WiFi network.

ToY devices use the Teleduino service I described in my previous article to coordinate their activity.

Chapter Eight - Thinking Of You

Each member of a ToY group must be assigned a Teleduino key (a string of 32 hex characters that uniquely identifies a specific ToY device) which is available for free from:

www.teleduino.org/tools/request-key. If you have five members of your group you will need five keys.

After all of the information about ToY group members is gathered it must be edited into the *devices* array within the software. (in the ThinkingOfYou.ino file). An example is shown below:

```
DEVICE devices[] = {  
    "Sister1", "S1_SSID", "S1_PWD", "S1_KEY", redHeartPattern,  
    "Sister2", "S2_SSID", "S2_PWD", "S2_KEY", greenBlueSweepPattern,  
    "Craig",   "MY_SSID", "MY_PWD", "MY_KEY", rainbowPattern,  
};
```

There will be one line of data in this array for each group member.

Once the *devices* array is filled in, the Thinking of You software must be compiled and uploaded into each ToY device. The software installed in each member of a ToY group must be identical.

Software

Although the ToY hardware is quite simple the software is anything but. Luckily you don't really need to know how the software works in detail to be able to use the ToY devices. If you are interested in learning how the software works read on.

The software was developed using the Arduino IDE. See my previous article for how to set-up the Arduino IDE on your computer for targeting ESP8266 type devices. Make sure to select “NodeMCU 1.0 (ESP-12E Module)” as the board type in the tools menu.

The Thinking of You software should be available on the Nuts and Volts website in association with this article. The file is called: *Lindley_ThinkingOfYou.zip*. To use this software unzip it and move the ThinkingOfYou directory into your Arduino directory.

The Thinking of You software is made up of the following files:

Filename	Purpose
ThinkingOfYou.ino	The main program file. This code initializes the ToY device and then runs a rather complex state machine which: polls the request switch for activity, queues requests from ToY group members for pattern display and controls the length of time LED patterns are displayed.
LEDControl.ino	Code for controlling the RGB LED using PWM (Pulse Width Modulation) and code for color creation and conversion.
Patterns.ino	Defines the colorful lighting patterns that can be associated with a ToY group member. Each pattern is written as a state machine. You could

Filename	Purpose
	easily add your own patterns once you understand how this software works.
TeleduinoClient.h	The interface specification for the TeleduinoClient class. It defines a class constructor and four public class methods.
TeleduinoClient.cpp	A rather complex state machine that establishes a TCP/IP connection to the Teleduino server and interprets the data it receives back.
TeleduinoRequest.ino	Code for issuing an HTTP GET request to the Teleduino server
Types.h	Miscellaneous data type definitions

Once you have the Arduino IDE setup correctly on your computer, have downloaded the code from the Nuts and Volts website and have edited the *devices* array with information about your ToY group members you need to compile the code and upload it to each ToY device in your group. There shouldn't be any warnings or errors during the compilation and/or upload processes.

When you power up your ToY device you should see the RGB LED change color in sequence from green to a whitish blue color and then go off. This sequence indicates your ToY device is working correctly and that it has established a local WiFi connection. If you don't see this LED color sequence bring up the Arduino Serial Monitor and look for error messages which hopefully will lead you to the problem and subsequent solution.

Note: there is a small blue LED on the NodeMCU module that indicates the module is powered up. It is normal for this LED to be on all of the time.

How Things Work

When a ToY device powers up the network name (SSID) and password of each of the groups members (from the *devices* array) are passed to the ESP8266 and it in turn tries to connect to each sequentially. Once the local WiFi connection is established, a persistent connection to the Teleduino server is made and the local ToY device's key is passed to identify itself. The Teleduino server first verifies the key is valid and then begins a message exchange with the ToY device to verify it is up and running. This exchange, managed by the TeleduinoClient code, repeats approximately every 5 seconds to verify the ToY device is still alive and well. This process will continue as long as the ToY device is powered up.

An HTTP GET request targeting each group member is sent to the Teleduino server when any one of the ToY group members presses its button. Passed in this request is the index into the *devices* array of the device making the request.

The TeleduinoClient code in each group member will then receive an event from the Teleduino server which indicates which ToY device made the request. A lookup in the *devices* array then retrieves the function pointer for the LED lighting pattern corresponding to the ToY device that made the request and this pointer is added to the LED pattern display stack. Code running in the background sees that the display stack is no longer empty and starts the LED lighting pattern on each ToY device.

If and when a group member responds its display pattern is appended to the display stack and the patterns display sequentially.

A background function monitors how long the patterns have been displayed and after 30 minutes of no switch activity clears the display stack and turns off the RGB LED.

Packaging the ToY Device

I wanted the ToY devices to be as small as I could easily make them and as unobtrusive looking as possible. I definitely didn't want them to look like a bunch of wires and parts kludged together because they will need to be placed in a conspicuous place to be seen and interacted with.

Fortunately I had just visited a Container store and remembered these cool little, vividly colored, plastic boxes I thought would work perfectly. They come in all sizes but I picked one with dimensions of 1 5/8" x 1 5/8" x 2 7/8" which would easily fit the NodeMCU module with room to spare. I decided to buy two of these boxes for each device: a clear one and a colored one. I would use the top of the clear box and the bottom of the colored box together. This would allow the light from the RGB LED to radiate from the top of the box but would help hide the electronics within the bottom.

With that decided I used steel wool to frost the clear top to diffuse the LED's light. I then drilled a hole in the middle of the top for the miniature push button switch. I used a small file to create a square hole for the switch and then used a couple of drops of super glue to hold the switch in place.

I then cut the small connector off the end of a USB cable as I needed to feed the cable through a hole in the bottom portion of the box. I also decided the electronics was still to visible so I cut four pieces of black frame matting to fit the inside of the bottom of the box. This not only hides the electronics but formed a support for mounting the NodeMCU module as well.

Figure Five shows two units being prepped for assembly. Here you can see the frosted tops with attached switches, the USB cables fed through the sides of the colored boxes, the pieces of frame matting and how each NodeMCU module was attached to a piece of the matting with drops of super glue in each corner.

All of the electronic components are soldered directly to the NodeMCU module except for the switch which is wired to the module. This can be seen in Figure Six. I made the resistor and LED wires as short as possible to make them mechanically rigid. I made sure the LED was placed off to the side of the USB connector on the NodeMCU module so my modified USB cable could be easily attached.

As mentioned, I had to cut the end off of the USB cable so that it could pass through a small hole in the side of the plastic box. It would be possible to strip the wires from both ends of the cable and reattach the micro USB connector to the cable but I decided to use a new connector instead. This would allow the cable/connector to have a lower profile so it wouldn't extend up into the clear plastic box top.

With the cable completed, I attached the cable to the NodeMCU unit and slid it into the colored box

bottom. I then inserted the other three pieces of matting material and held them in place while I put a drop of super glue in the corners to hold them together. Placing the clear plastic box top onto the bottom completed the construction.

Conclusions

Keeping in touch with family and friends is very important in the hectic world we live in. That, however, is not the only use for a ToY device. It could be used, for example, to alert your friends across town that it is time for beer o'clock or be used to monitor your children's continued presences at some else's house or to inform your morning car pool members you are on your way and who knows what else. I am sure many uses for these devices will present themselves if you give it some thought.

For me, I think this will make a nice Christmas present for myself and my sisters. Then, with a touch of a button we can let each other know that we are thinking of them.

I would like to thank my friends Dave and Barbara Resch for helping me test the Thinking of You concept and the actual devices.

Resources

Information about Teleduino is available at: www.teleduino.org

Information about programming the ESP8266 in the Arduino environment can be found at: github.com/esp8266/Arduino.

Information about the NodeMCU Amica can be found at: www.electrodragon.com/product/nodemcu-lua-amica-r2-esp8266-wifi-board/.

Chapter Eight - Thinking Of You

Figure One

Thinking of You Device Parts List (per unit)

Quantity	Description	Source
1	NodeMCU Amica R2 Module	electrodragon.com
1	RGB LED common cathode	amazon.com
3	1 K ohm ¼ watt resistor	anywhere
1	USB power adapter/charger 1 amp @ 5 VDC minimum	ebay.com
1	Micro USB Cable - A to Right Angle Micro B	amazon.com
1	Micro USB Type A Male 5 Pin connector (optional)	amazon.com
1	PCB Momentary Tactile Push button switch	amazon.com
1	Amac Box Clear #60300 you can also buy two boxes: one colored and one clear and put the clear top on the colored body	containerstore.com
	Frame matt material	Hobby Lobby/Michaels
	Super Glue	Hobby Lobby/Michaels
	Hookup wire	anywhere

Chapter Eight - Thinking Of You

Figure Two
Thinking of You Device Schematic

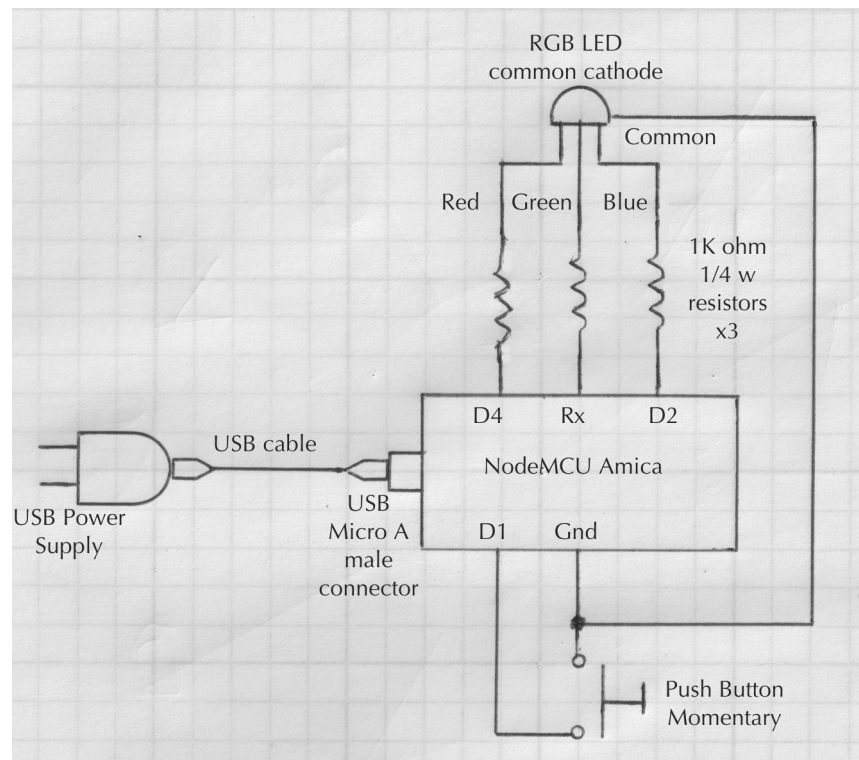
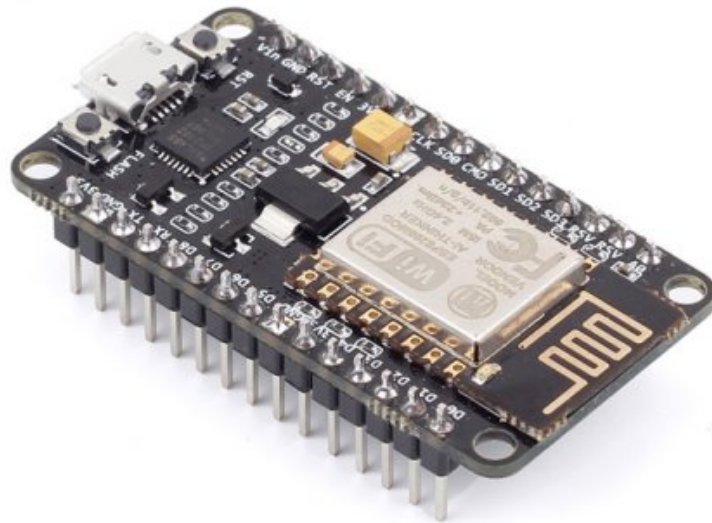
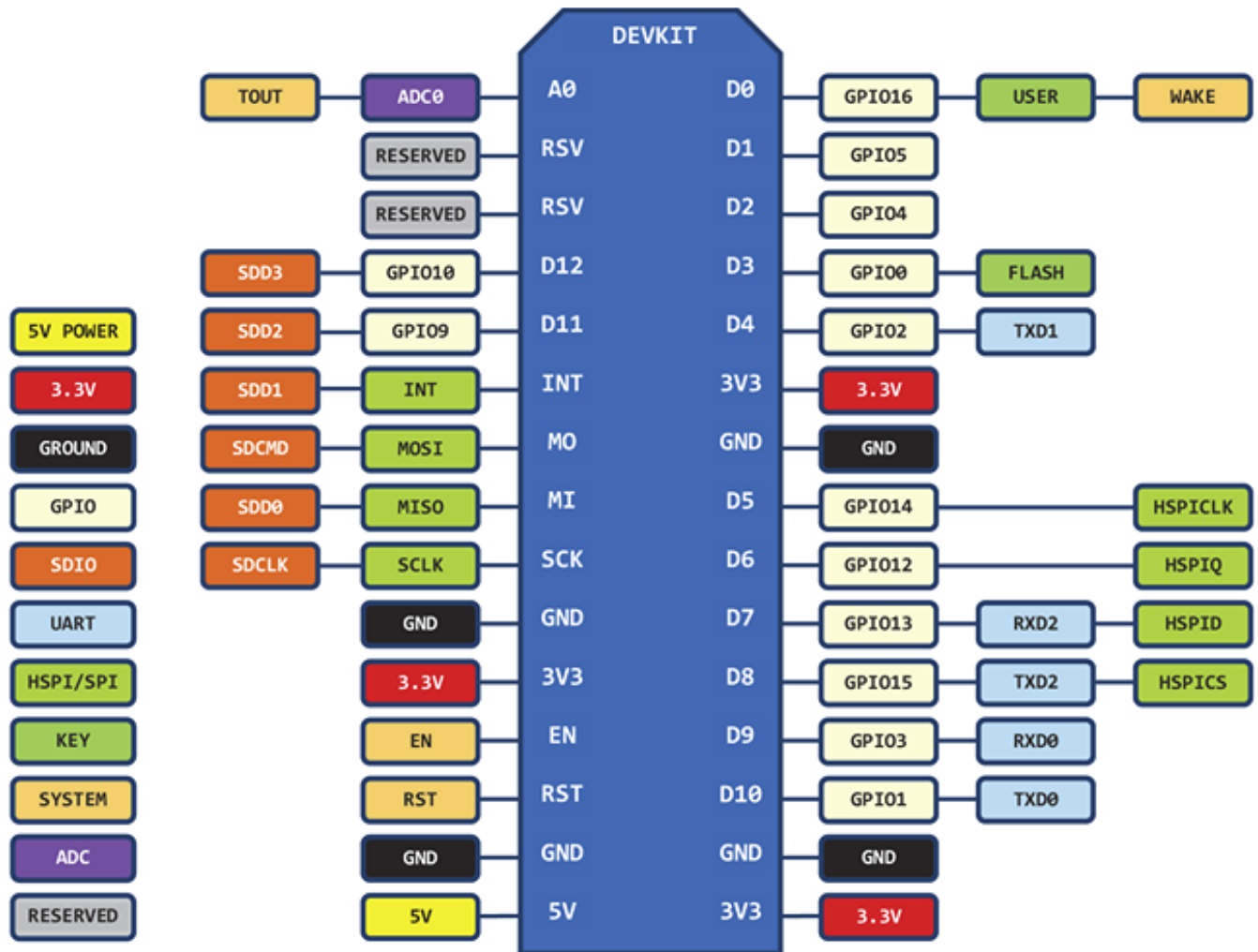


Figure Three
The \$10 NodeMCU Amica R2 Module
containing the ESP8266-12



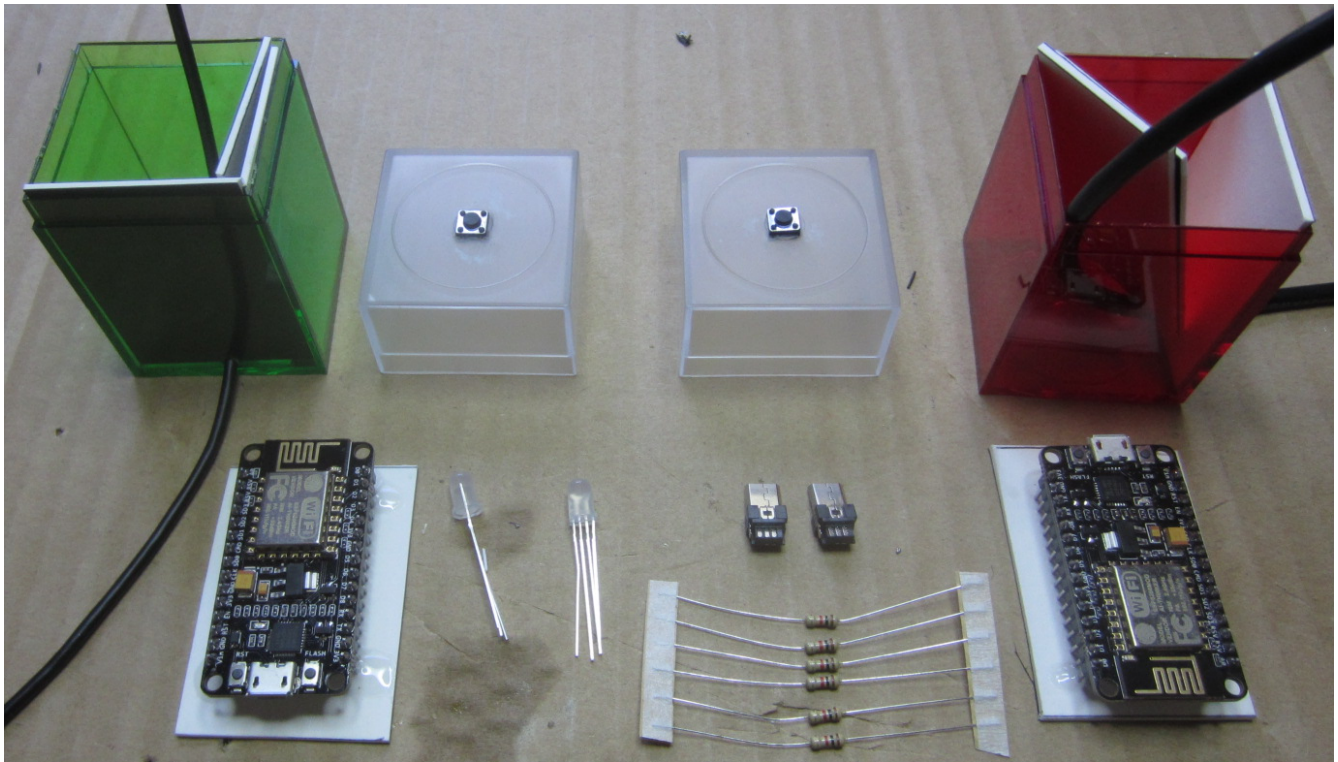
Chapter Eight - Thinking Of You

Figure Four
NodeMCU Amica Module
Pinout



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Figure Five
Preparation for Construction



Chapter Eight - Thinking Of You

Figure Six
Assembly Complete

Note: the NodeMCU module is super glued to the frame matt material

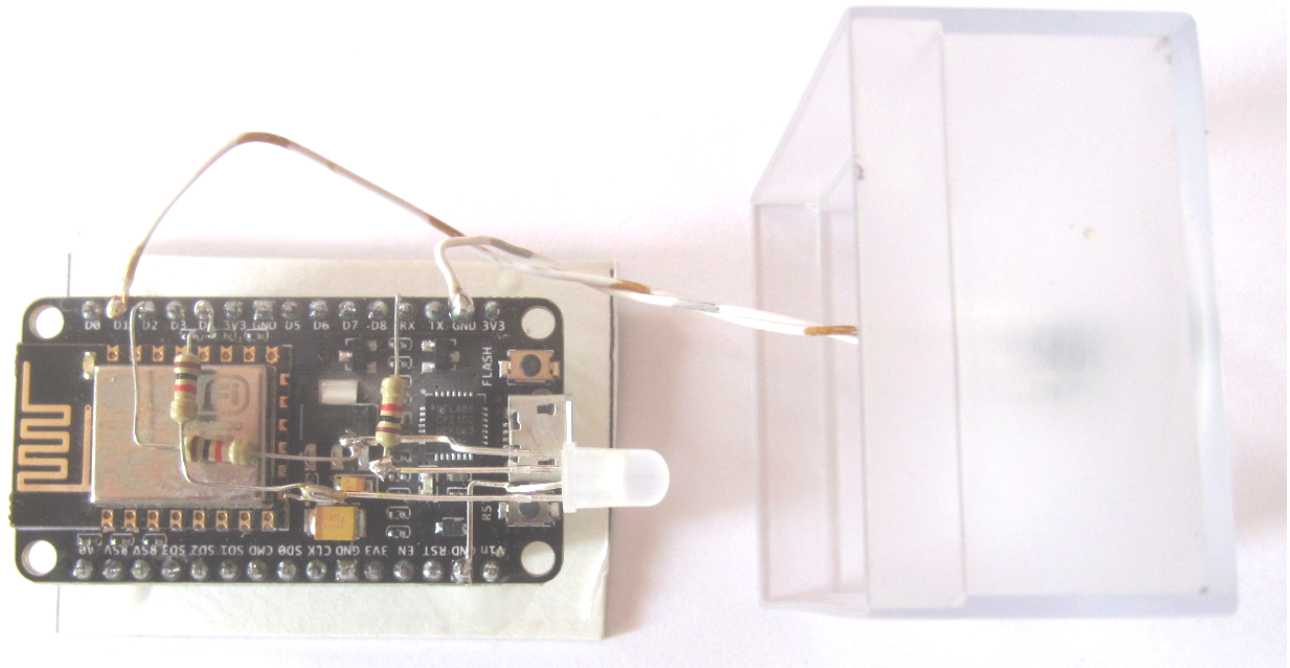
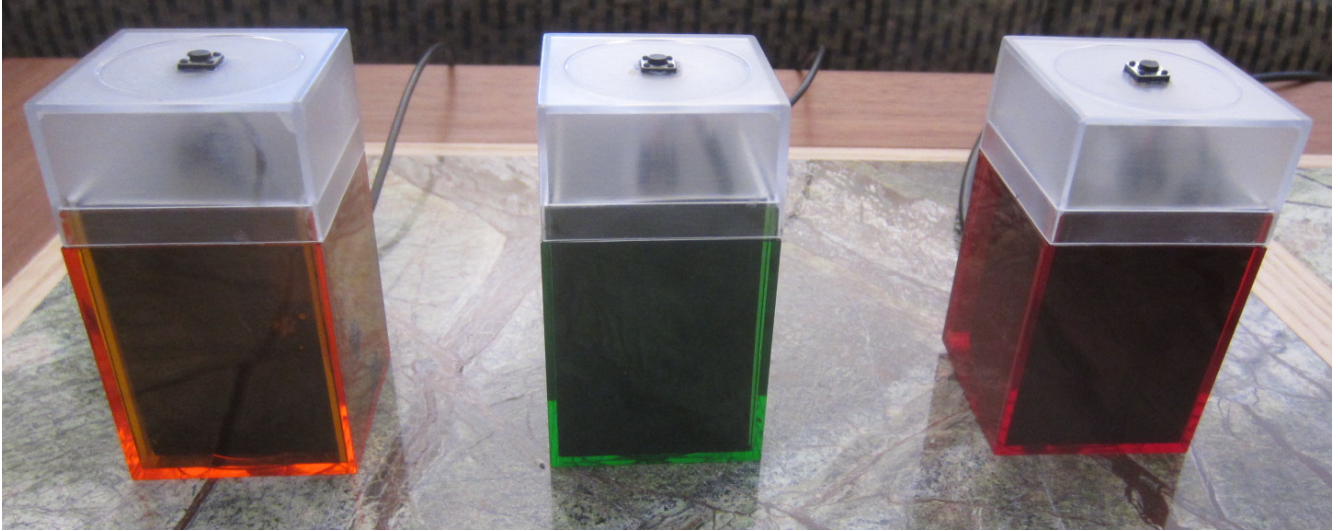


Figure Seven
ToY device is smaller than a coffee cup



Chapter Eight - Thinking Of You

Figure Eight
My Family's Group of Three
Thinking of You Devices



Chapter Nine - WiFi Robot and Robot Controller

Introduction

Let me state up front that I know next to nothing about robots or the current state of personal robot technology. Most of the projects I design and build have to do with making LEDs blink colorfully or making music on computers. Many of my recent projects have been based upon the ESP8266 micro controller / wifi module because of its capabilities and its inexpensive price. Specifically, my recent projects have all used the NodeMCU Amica device shown in Figure One, which contains a ESP8266-12 module. Why do I like this module, let me count the ways:

1. It contains a 32 bit micro controller which can be programmed in the Arduino environment allowing anyone with Arduino experience to utilize this device.
2. The micro controller can run at 160 MHz which gives it a lot of power for complex tasks.
3. The micro controller has ~80K of RAM and ~1 MByte of flash for program storage along with some EEPROM as well.
4. It has industry standard i2c and SPI interfaces and 13 general purpose I/O pins most of which support PWM. Figure Two shows the pinout of the NodeMCU Amica module.
5. The module has a built in WiFi interface which supports the 802.11 b / g / n networking standards. This allows the module to talk directly to a home or business WiFi network along with the ability to function as a stand alone wireless network.
6. The ESP8266 has a large and growing user base so there is a lot of information and projects on the Internet and many knowledgeable people who can help if you have problems putting these devices to work.
7. Last, but not least, the NodeMCU Amica module is available for \$6.40 each at electrodragon.com.

Why anyone would still be using a genuine Arduino or many of the other micro controller boards currently on the market is beyond me.

NOTE: See <http://esp8266.github.io/Arduino/versions/2.0.0/doc/installing.html> for instructions on how to install the ESP8266 development software on your computer. Make sure you select NodeMCU 1.0 as your board type and 160 MHz as the CPU frequency if you try to reproduce my results.

In a conversation the other day with Bryan Bergeron he suggested I might apply my knowledge and fondness for the ESP8266 to robotic applications and that got me thinking. I had never built an electronics project that actually moved so this might be a good opportunity to do so. Could a NodeMCU Amica module be the brains of a robot? I decided to find out.

Of course since this would be my first robotics project I needed to keep things simple. I viewed this challenge more as a testbed for the technology than an attempt to build a full featured robot. However after getting my robot to work I realized how extendable the robotics platform I will describe in this article was. It could be used as the basis for any number of robot designs much more powerful than what I present here.

In a nutshell this is what I have done. On the robot front I have coupled the NodeMCU Amica module to a L293D H-bridge motor controller chip connected to an inexpensive two motor robot chassis from makershed.com.

On the robot controller side I have coupled the NodeMCU Amica module to a joystick and an RGB LED. Using the joystick you can wirelessly drive the robot around but at the present time, that is all you can do. The LED provides a visual indication of the status of the wireless communication link between the robot and the robot controller. Limiting as this is, it does prove the NodeMCU Amica module can be the brains of both a robot and its accompanying controller.

Both the robot and the robot controller's electronics are built using point to point wiring on perf board. Figure Three shows the results of my effort.

The Robot Controller

The robot controller can be seen in Figure Four. The robot controller controls the robot via its joystick. While the controller could have been powered with batteries I chose to power it via USB as this was only a prototype. As mentioned the RGB LED displays the controller's status. If the LED blinks red it means the controller could not connect to the robot. If it blinks blue, it means that the link up handshake carried on between the robot and the controller has gotten out of sync. If the LED glows green, the controller and the wireless network are operational.

The position of the joystick controls the speed of the robot's motors. Whenever the joystick is in the center or released position, the robot comes to a stop. The forward position of the joystick is towards the NodeMCU module on the circuit board. Joystick positions above center drives the robot forward while positions behind center drive it backward. If the joystick is moved left of center the robot's left motor is slowed down so the robot turns towards the left. Conversely, if the joystick is moved to the right of center, the right motor slows down making the robot veer to the right. The speed of the robot is controlled by how far the joystick is pushed in any direction. With a little practice the robot can be driven smoothly and is responsive enough to be driven around obstacles.

Text tokens are passed wirelessly between the robot controller and the robot to control its operation. The controller sends out a ping token about once a second that the robot should receive and echo back to confirm link status. The controller expects to receive these ping messages and will go into an error state if they are not received. The ping token is just the string "ping" followed by carriage return (ascii code 13) and line feed (ascii code 10) characters.

Robot motor speed is controlled by two other text tokens "lm:speed" and "rm:speed" passed from the robot controller to the robot. "lm" stands for left motor while "rm" stands for the right motor. Speed is a signed entity that sets the corresponding motor's speed. Speed ranges from zero or stopped through 1023 which corresponds to the maximum speed of the motor. Currently the robot controller software limits maximum motor speed to make driving slower but easier. If speed is positive the motor runs forward; if negative the motor runs backward.

Hardware

The Robot Controller consists of the following parts.

Item	Description	Source
NodeMCU Amica Module	Micro controller with WiFi interface	electrodragon.com
CD4066B	CMOS quad bilateral switch	ebay.com, digikey.com, etc.
RGB LED	Common cathode type	“
3 resistors	1K ohm ¼ watt, 5%	“
Thumb Joystick	COM-09032	sparkfun.com
Misc	Wire, perf board, 14 pin IC socket, solder	Junk box

The Fritzing diagram in Figure Five shows how simple the robot controller's circuit is while Figure Six shows the actual schematic.

While the NodeMCU Amica module is the central component in the robot controller's design it is the interface to the joystick that is the interesting part of the circuit. For those who haven't used a joystick before, the particular joystick used in this design is comprised of two potentiometers and a switch. As the joystick is moved left and right the horizontal potentiometer changes value. As the joystick is moved forward and backward the vertical potentiometer changes values. The joystick's switch closes if the joystick is depressed and opens when the joystick is subsequently released.

In this design the potentiometers are wired between the 3.3 VDC supply (from the NodeMCU device) and ground so the arm of the potentiometer varies between these two values as the joystick is moved. The analog to digital converter (ADC) in the NodeMCU is a 10 bit converter so input voltages in the range 0.0 to 3.3 VDC result in converted values between 0 and 1023.

The problem was the joystick supplies two analog signals while the NodeMCU device has only a single ADC input, A0. I got around this problem by using two sections of an CD4066 CMOS quad bilateral switch. Each of the four analog switches in these devices are controlled by a separate control signal. When the control signal is high, a low resistance path is established between the input and output of the switch i.e. the switch is on. When the control signal is low, the switch turn off. These analog switches are called bilateral because signals can flow in either direction. For example switch section A has two signal connections: A in/out and A out/in and either one can be the input or the output.

I got around the single ADC input issue by turning analog switch A on when reading the horizontal joystick signal and turning it off and analog switch B on when reading the vertical joystick signal, effectively giving the ADC two inputs. Software in the NodeMCU device switches the analog switch control signals back and forth while the joystick's position is being read. Currently the joystick's switch is not being used but it is wired to a GPIO line on the NodeMCU device

in case the software ever wants to make use of it.

Software

While the hardware for the robot controller is relatively simple, the software to make it all work is somewhat complex because there is a lot going on. The software has to establish and maintain the wireless connection between itself and the robot, it has to continually read the position of the joystick and pass motor control tokens over the link and it has to monitor link synchronization between the robot and itself.

While all this activity calls out for some sort of multi-tasking to handle these multiple disparate tasks we don't have that luxury on the NodeMCU. Instead I employed a Finite State Machine (FSM) to manage things. Technically a FSM is:

“A model of a computational system, consisting of a set of states, a set of possible inputs, and a rule to map each state to another state, or to itself, for any of the possible inputs”.

Sounds daunting but don't let the definition scare you, the operation is really quite simple.

The operation of the robot controller can best be understood by examining the code in the file *RobotController.ino* which is available on the website.

The user configuration items are first up. These are the networking parameters which must match that of the robot or the communication between the robot and robot controller won't be possible. The robot creates its own network called an access point (AP) and these parameters make sure the robot controller is talking to the robot's AP IP address at port 8000. *MAX_SPEED* defines the maximum speed of the robot's motors. When set at 900 this slows the robot down to make navigation easier. Setting this to 1023 would make the robot's motors run at full speed.

Next up are the hardware configuration items which makes the software aware of which I/O lines are connected to which peripherals. Following this, are utility functions used throughout the software. The *setup()* function which follows configures the hardware and attempts to logon to the robot's network. If logon fails the RGB LED blinks red to let the user know there is a network problem. If the connection is successful, the LED turns green, the initial state of the state machine is set and the setup function ends.

The *loop()* function contains the FSM which orchestrates everything. The FSM is just a switch statement which is controlled by the *state* variable. The current FSM state is evaluated every time through the loop.

The FSM is made up of the following states:

State	Operation
STATE_INIT	Initializes important FSM variables then transitions to the

	STATE_CHECK_CONNECTION state the next time through the loop.
STATE_CHECK_CONNECTION	Check the status of the network connection between the robot and controller. If the connection is up transitions to the STATE_CHECK_PINGS state. If the connection isn't up a transition is made to the STATE_MAKE_CONNECTION state.
STATE_MAKE_CONNECTION	In this state an attempt is made to contact the robot at its IP address and port. If successful the RGB LED is set to green and a transition is made to the STATE_CHECK_PINGS state. If the robot cannot be contacted, a transition is made to the STATE_ERROR state which causes the LED to blink red.
STATE_CHECK_PINGS	Two things happen in this state. First, a determination is made as to whether a ping synchronization message should be sent to the robot. If the ping timeout has expired, a ping message is send and the timeout is reset. Second, a check is made to make sure the controller has received the echoed ping message from the robot in the allotted time interval. If everything is fine a transition is made to the STATE_CHECK_INCOMING state. If the controller hasn't received a ping message in the allotted time a transition is again made to STATE_ERROR but this time the LED will flash blue.
STATE_CHECK_INCOMING	In this state the controller listens for input from the robot. At the present time the only input of interest from the robot is the echoed ping message. If this is received, the received ping timeout is reset and a transition is made to the STATE_PROCESS_JOYSTICK state.
STATE_PROCESS_JOYSTICK	In this state the joystick's position is read and manipulated into the left and right motor messages previously discussed. New messages will only be sent to the robot when the joystick's position changes. At the end of this process a transition is made back to the STATE_CHECK_CONNECTION state and the FSM process starts over.
STATE_ERROR	This state is entered when there is a problem with the network connection or when the link goes out of sync. The RGB LED blinks with a color indicative of the error and then resets the state to STATE_INIT causing the software to re-start in an attempt to correct the error.

The Robot

Hardware

The Robot is made up of the following parts.

Item	Description	Source
ModeMCU Amica Module	Micro controller and WiFi interface	electrodragon.com
L293D	Motor controller chip	ebay.com, digikey.com, many others
4 - 100uF capacitors	Filter capacitors - 50 VDC or greater	Radioshack
8 – 0.1 uF capacitors	Noise reducing capacitors	Radioshack
Adjustable voltage regulator	Step down shunt regulator	ebay.com
Robot Platform	Including chassis, rear caster, two DC motors with wheels, battery box for 5 AA batteries, power switch, power connector and misc assembly hardware.	makershed.com or many other places
Misc	Wire, perf board, 16 pin IC socket, power connectors, solder, tin for shield, etc.	Junk box

Figure Seven shows a closeup of the robot's electronics and Figure Eight shows its schematic.

Two items of note. First, you'll notice that an SD memory card is visible on the robot's circuit board. This is not used in the basic robot configuration I present in this article but was included to enable future enhancements.

Second, under the perf board on which the robot's electronics were built is a metal shield made out of tin but insulated from the point to point wiring with a piece of cardboard. Initially I had trouble with the NodeMCU rebooting when the motors were turned on and this shield along with the addition of numerous noise filtering capacitors around the motor controller chip cured the problem.

Many of you are probably familiar with the L293D H-bridge motor controller chip used in this design. This chip is used to control the two low current DC motors powering the robot directed by logic signals from the NodeMCU module. In a typical L293D design, a set of inputs, 1A and 2A for one motor and 3A and 4A for the other motor are use to control direction while the enable pins are fed a PWM signal to control motor speed. This arrangement requires six digital lines from the micro controller which I didn't have available (because of the SD memory card interface). So instead I used only two input per

motor and some clever software to control both direction and speed. The following truth table describes how the motors react to their input signals.

1A/3A	2A/4A	Action
0	0	stop
1	1	stop
0	1	forward
1	0	reverse

The stall current for these motors was spec'ed at ~600 mA, well within the drive capabilities of the L293D chip.

The robot platform I purchased came with a battery box for five AA batteries. The 7.5 VDC provided by the batteries was fine for driving the motors but needed to be reduced to 5 VDC to power the NodeMCU module and the L293D chip. This was done using a adjustable step down buck regulator that I purchased on eBay for around \$1.00. After wiring the battery box to the power switch and the regulator I adjusted the output voltage to 5 volts.

Software

The software in the robot is very simple and leaves lots of room for future enhancements. The code is available in the file *Robot.ino* also available on the website. One interesting thing about the software is that it configures the NodeMCU module to create a WiFi access point or AP. In essence the robot creates its own wireless network that the robot controller connects to without the need to use your home WiFi network. This means the robot can be used outside and has a range of about 30 feet.

The other interesting thing about the software is how the motors speed and direction are controlled. This is accomplished with the code below:

```
// Motor control function
void setMotor(int motorID, int dir, int speed) {
  if (motorID == LEFT_MOTOR) {
    if (dir == FORWARD) {
      // We are going forward
      digitalWrite(LM_DIR_PIN, 0);
      analogWrite(LM_SPEED_PIN, speed);
    } else {
      // We are going backward
      digitalWrite(LM_DIR_PIN, 1);
      analogWrite(LM_SPEED_PIN, 1023 - speed);
    }
  } else {
    if (dir == FORWARD) {
      // We are going forward
      digitalWrite(RM_DIR_PIN, 0);
      analogWrite(RM_SPEED_PIN, speed);
    } else {
      // We are going backward
      digitalWrite(RM_DIR_PIN, 1);
      analogWrite(RM_SPEED_PIN, 1023 - speed);
    }
  }
}
```


Chapter Nine - WiFi Robot and Robot Controller

```
    delay(25);  
}
```

The *setMotor* function is passed an identifier for the motor (LEFT_MOTOR or RIGHT_MOTOR), the direction the motor should move in (FORWARD or BACKWARD) and a speed value in the range 0 .. 1023. To go forward the odd control pin on the L293D is set low and a PWM signal is applied to the even control pin. To go in reverse, the odd control pin is set high and a PWM signal is again applied to the even pin but this time the PWM duty cycle must be inverted by subtracting the desired speed from 1023. If you draw out the PWM signals you'll see why this is necessary.

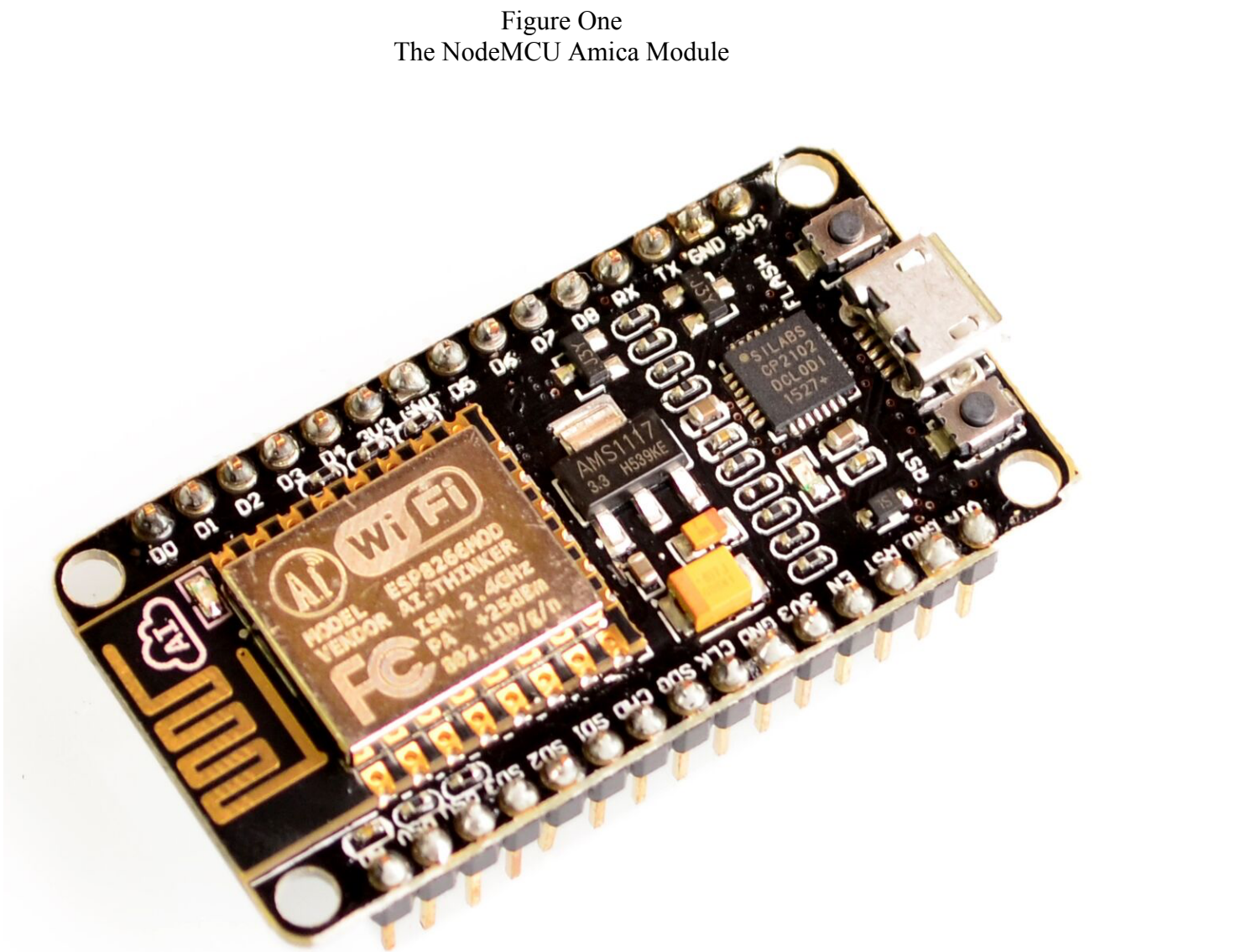
The *setup()* function in the code sets all of the motor control pins to outputs and sets them low to initially stop the motors. Next, access point networking is setup and the server to which the robot controller communicates is brought up.

It is within the *loop()* function where all the action is. Here the code waits for a connection from the robot controller and when established, processes the messages it receives. If a ping is received it is immediately echoed back to the robot controller to verify link integrity. As motor control messages are received they are parsed and the results are sent directly to the motors via the *setMotor* function described above.

Conclusions

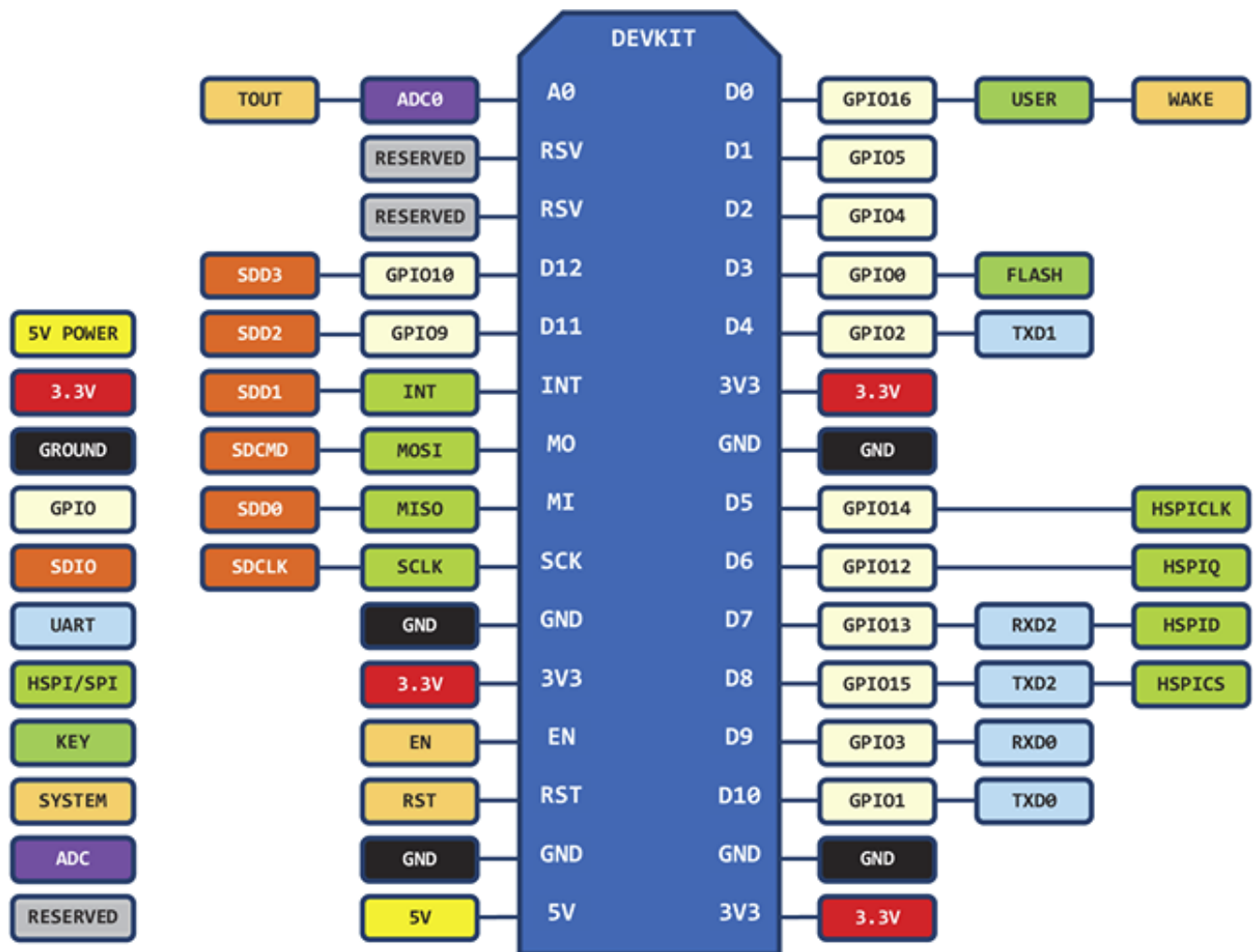
The availability of inexpensive NodeMCU Amica modules with their embedded WiFi and 32 bit micro controller makes this form of robotic remote control possible. The basic robot and robot controller described in this article just scratch the surface of what could be done in real robot applications. The micro controller's on both the robot and the robot controller have many more processor cycles available which could be put to use for object avoidance, line following, laser pointing or you name it. If you do something interesting with what I have presented here, let me know.

Figure One
The NodeMCU Amica Module



Chapter Nine - WiFi Robot and Robot Controller

Figure Two
NodeMCU Amica Pinout Diagram

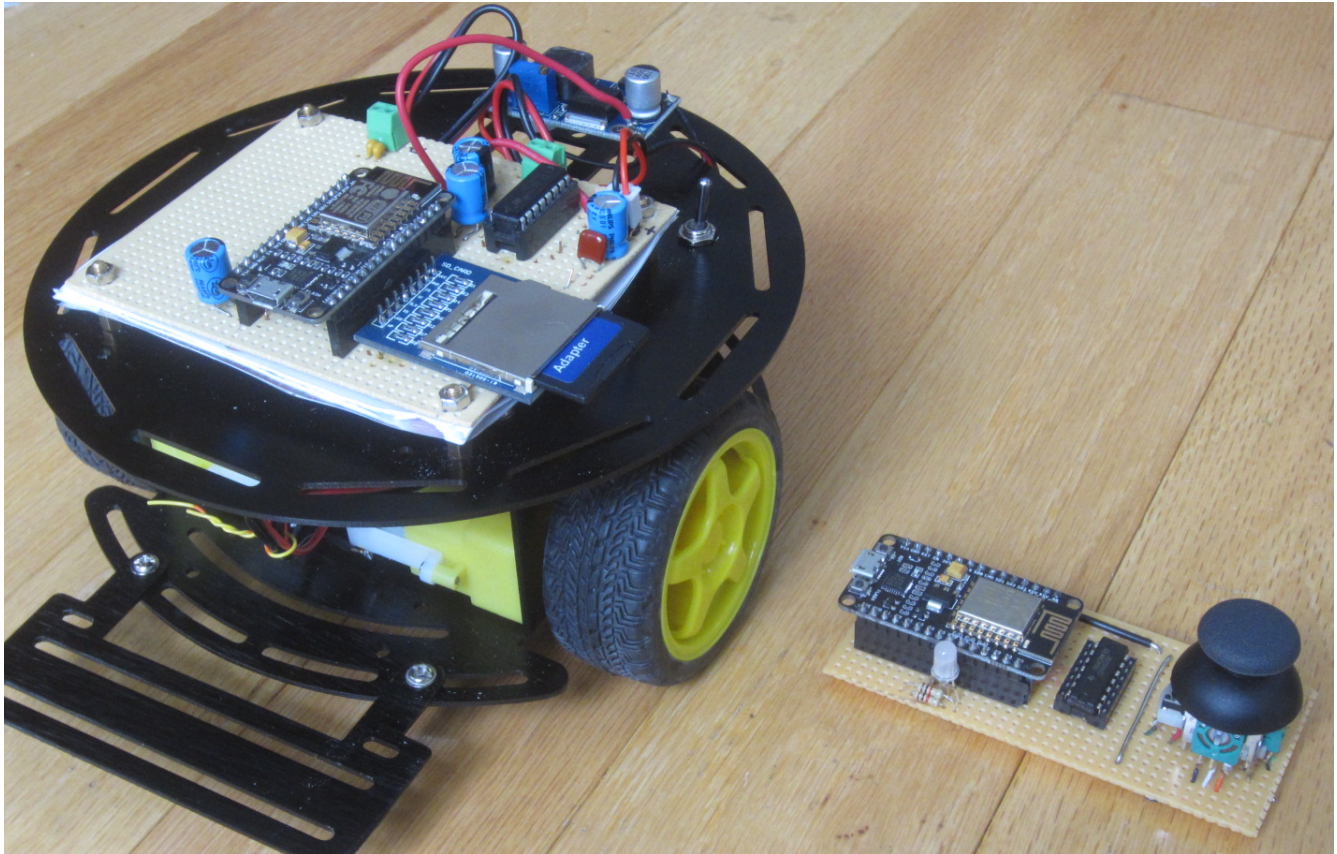


D0(GPI016) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Figure Three

Robot and Robot Controller

The little module on the back of the robot is the adjustable voltage regulator



Chapter Nine - WiFi Robot and Robot Controller

Figure Four
Robot Controller Closeup

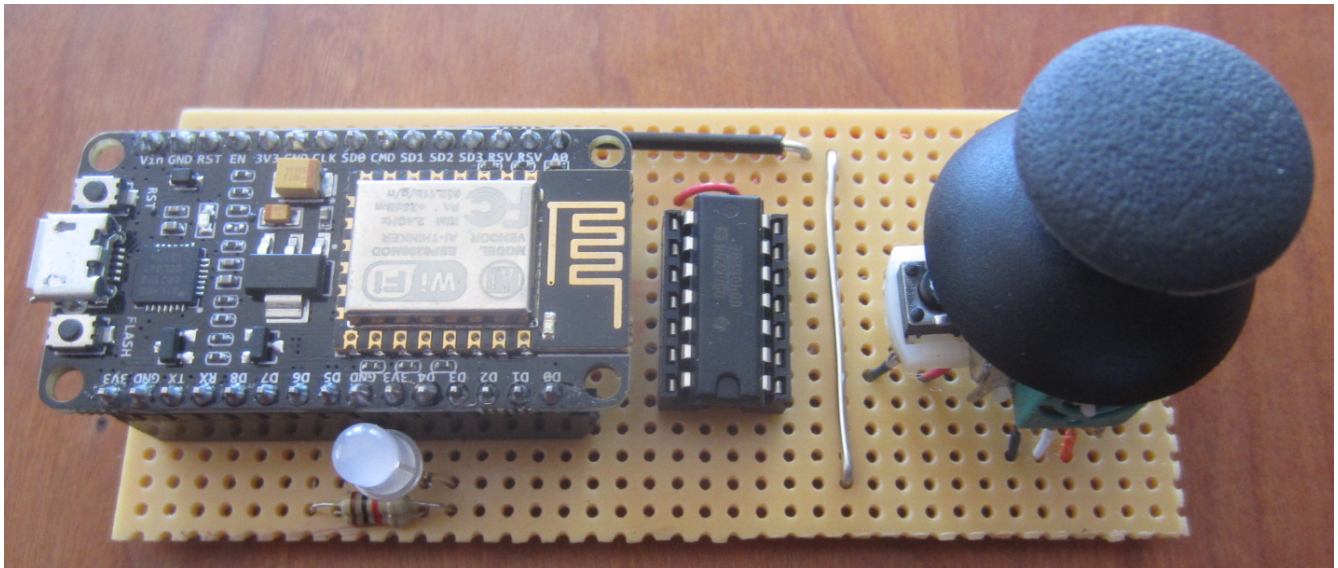
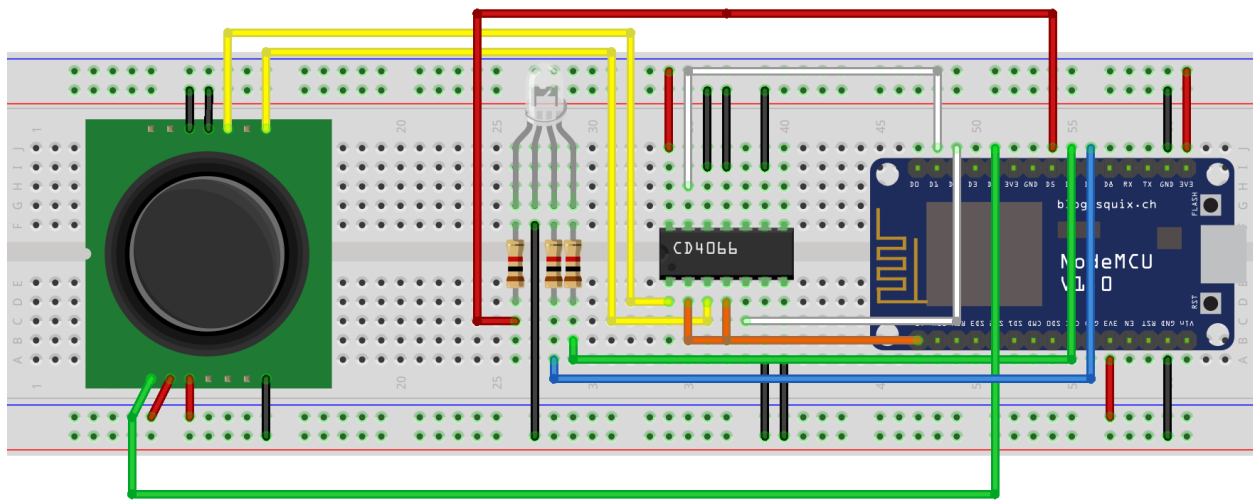


Figure Five
Robot Controller Fritzing diagram shows how simple the circuit is



Chapter Nine - WiFi Robot and Robot Controller

Figure Six
Robot Controller Schematic

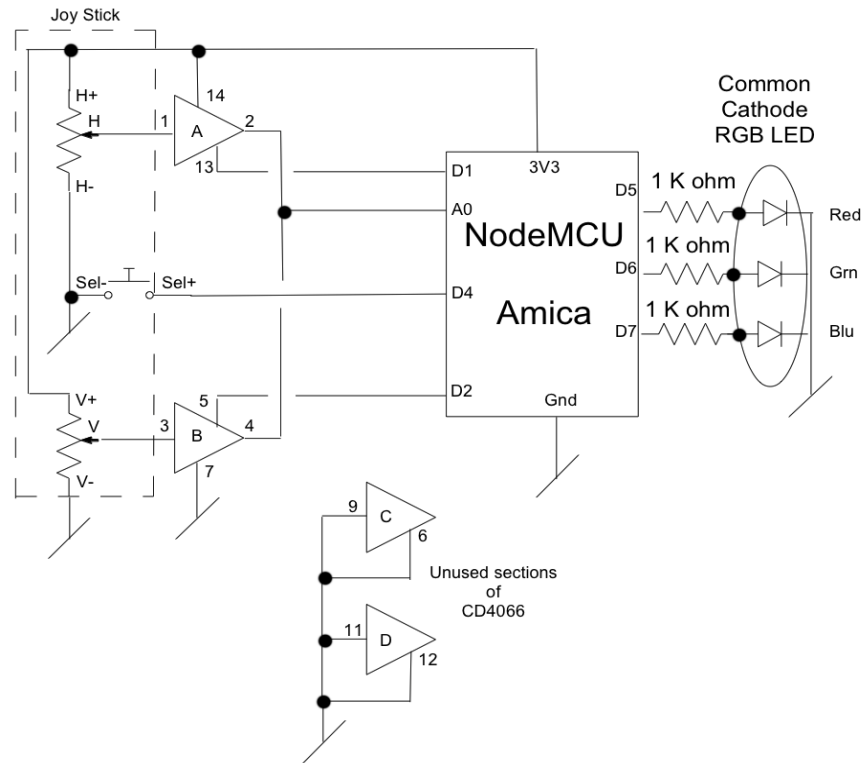


Figure Seven

Robot Electronics Closeup

Note: the SD memory card is not used in this basic robot application

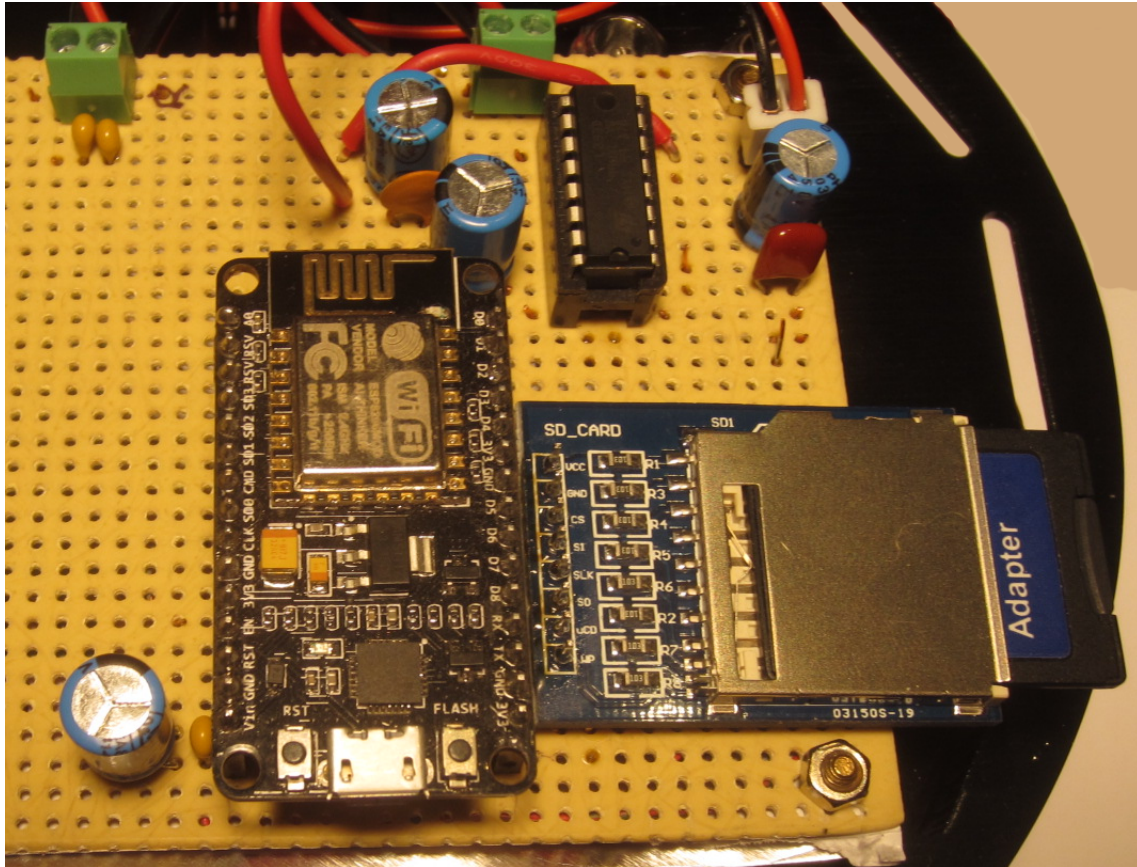


Figure Eight
Robot Schematic

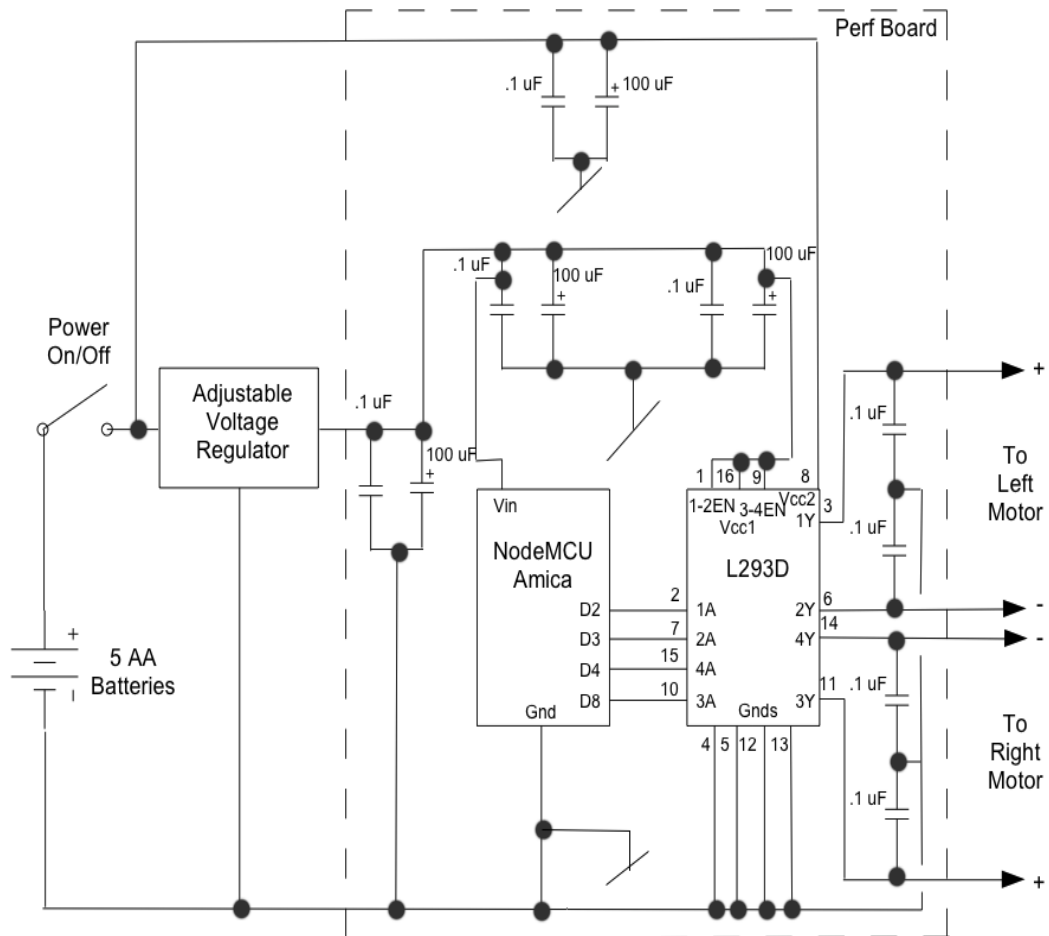
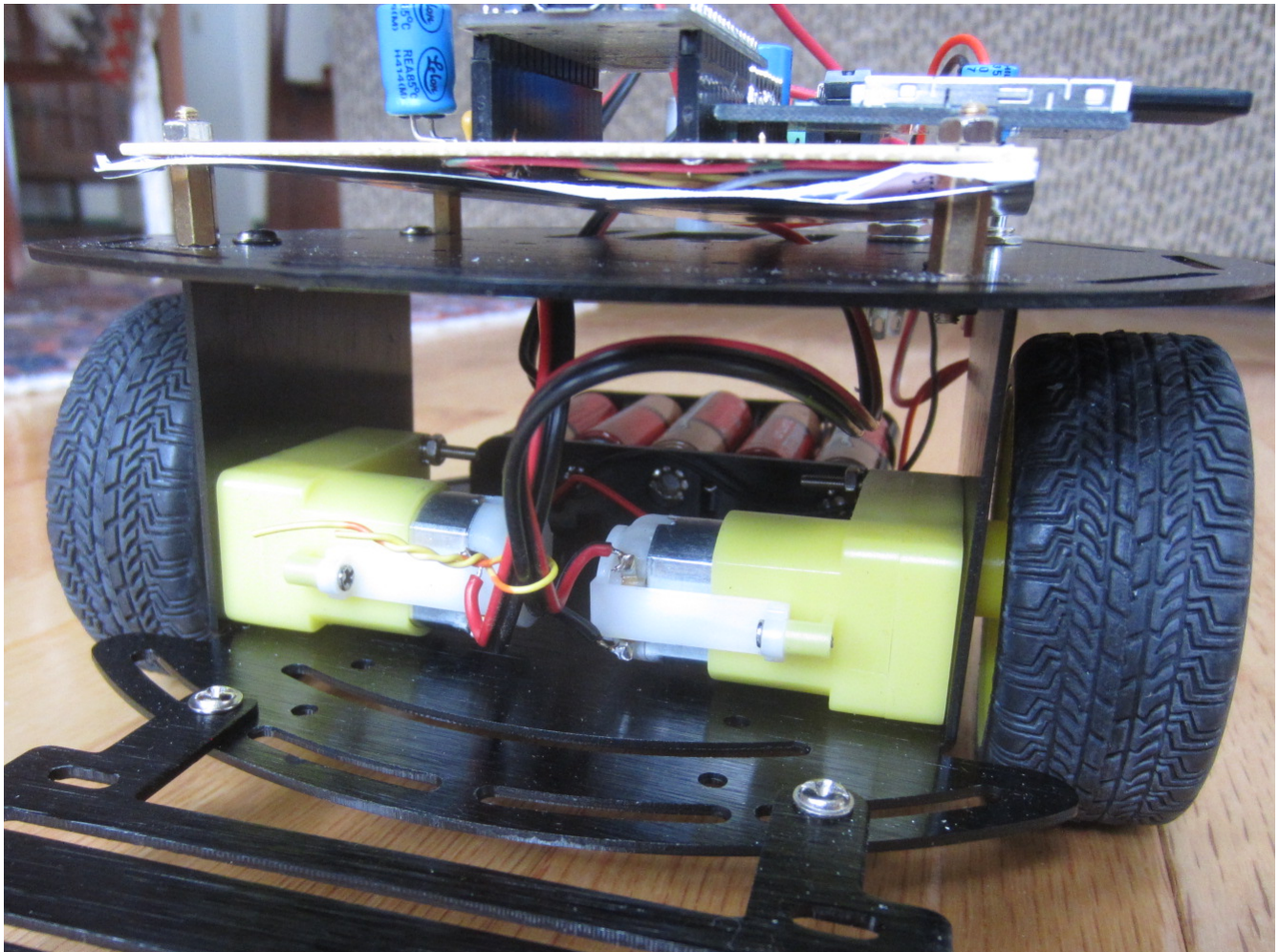


Figure Nine
Robot Thru View



Chapter Ten – NeoPixel LED NTP Clock

Introduction

Did the the world really need yet another digital clock running on a micro controller? I didn't think so until daylight savings time changed again and I had to go around my house and change the time on many of my clocks. Not only that, one of the batteries in a clock had died so I had to replace the battery and then set the time and date again. In another of my clocks the quartz movement seems to gain time slowly so I never really know the time accurately. I started thinking about what a pain this all was so I decided to build a clock that:

- Is extremely simple to build
- Requires no switches for setting time and / or date
- Didn't need a backup battery or batteries of any kind
- Was always accurate
- Dealt with daylight savings time automatically by itself

In other words I wanted to build a clock that needed zero maintenance on my part and that would always display the time and date correctly. I decided to base my clock on the Network Time Protocol (NTP) that all modern personal computers/devices use to synchronize their time keeping activities. The time reported by NTP servers can be traced back to atomic clocks at the National Bureau of Standards or NIST so it is very accurate all of the time. Of course this meant my clock would have to have access to the Internet to request NTP time and that is where the amazing NodeMCU Amica module with a built in ESP8266 processor came in. Not only does this module feature a WiFi interface, it also contains a 32 micro controller which would be the perfect engine for this application. And did I mention it is cheap? around \$4.50 US in single unit quantities. Most realtime clock modules made for the Arduino cost more than this by themselves.

The hardware / software combination I present here implements a digital clock that never needs setting as it gets the current time and date by polling Network Time Protocol (NTP) servers on the Internet. The clock's time is synchronized to NTP time every 5 minutes to maintain its accuracy. Use of a Timezone library means that Daylight Savings Time (DST) is automatically taken into consideration so no time change buttons are necessary. This clock always runs in 12 hour mode.

Clock Operation

Once the clock has been assembled and the code downloaded into it, after a short pause the time will be displayed on the NeoPixels. The background color of the clock is a cyan. Hours are displayed on the small 12 NeoPixel ring using a red pixel for the current hour. Minutes and seconds are displayed on the large 24 pixel ring; minutes in green and seconds in gold. The eight LED strip across the bottom sweeps back and forth every second. The red, green and gold time indicating pixels move as the time changes.

Chapter Ten – NeoPixel LED NTP Clock

To make the clock a bit more fun I created events that occur on 30 minute, 15 minute and 10 minute intervals. During the 30 minute interval, time display is suspended and the components of the current date are displayed sequentially on the large NeoPixel ring. First comes the day of the week display with Sunday being day one. Next the month is displayed with January being the first month, followed by the day and finally the year. After the date is displayed, time display resumes until the next event.

The 15 minute event flashes red, green and blue colors sequentially on the large ring, small ring and NeoPixel strip. This is a colorful diversion from the mundane display of time.

10 minute events occur most often. The 10 minute event causes the NeoPixel LEDs to display a rotating rainbow of colors that will brighten any room.

Hardware

The hardware for this clock is very simple and consists of the following parts:

Part	Source
NodeMCU Amica R2 Module	electrodragon.com
100 ohm ¼ watt 5% resistor	RadioShack
12 LED NeoPixel Ring	adafruit.com
24 LED NeoPixel Ring	adafruit.com
8 LED NeoPixel Strip	adafruit.com
USB cable Micro USB Cable – type A to Micro B	amazon.com, ebay.com, RadioShack
USB Power Supply / Charger Module 5 volts @ at least 1.5 amp	amazon.com, ebay.com
Small picture frame, frame matting material and some super glue	JoAnns, Michaels, Hobby Lobby

The schematic of this clock is shown in Figure One. The NodeMCU Amica module which powers this clock is shown in Figure Two.

This clock is made up of three NeoPixel LED devices (two rings and a strip) which are Adafruit parts. They are mounted on a piece of frame mat material with a couple of drops of super glue. See Figure Three. Connections to these devices are via wires that run thru the back of the mat material.

The larger ring has 24 NeoPixels, the smaller has 12 and the strip has 8.

NeoPixels are individually addressable RGB LEDs with built in PWM and serial data controllers.

Figure Four shows the back of the clock's mat. You can see there isn't much to the clock circuit. The Node MCU module was super glued to the mat so it would remain in place while it was connected. As you can clearly see here only three wires connecting the NodeMCU to the NeoPixel devices. A 100 ohm resistor is placed between the NodeMCU Amica RX output and the input to the large NeoPixel ring to reduce noise on the connection. The output of the large ring connects to the input of the small ring. The output of the small ring drives the input to the NeoPixel strip. The +5V Vin and Gnd connections from the Amica are wired to all three NeoPixel devices.

The clock is powered externally via the USB cable shown and a USB power module of at least 1.5 amp output current which is not shown.

Software

The software is built using the following tools:

Software	Source
Arduino IDE Version 1.6.8 or newer	www.arduino.cc/en/Main/Software
ESP8266-Arduino library Version 2.2.0 or newer	github.com/esp8266/Arduino
NeoPixelBus library in DMA mode	github.com/Makuna/NeoPixelBus
Time library	github.com/PaulStoffregen/Time
Timezone library	github.com/JChristensen/Timezone

NOTE: the libraries I built this code with are included in the code directory accompanying this document.

See <http://esp8266.github.io/Arduino/versions/2.2.0/doc/installing.html> for instructions on how to install the ESP8266-Arduino software required to build the code within the Arduino IDE environment.

To use this software you must first configure it for your location. At the top of the file, *ESP8266_NeoPixelLEDClock.ino*, you will find the user configuration section shown below:

```
// *****
// Start of user configuration items
// *****

// Set your WiFi login credentials
#define WIFI_SSID "??????"
#define WIFI_PASS "?????????"
```

Chapter Ten – NeoPixel LED NTP Clock

```
// This clock is in the Mountain Time Zone
// Change this for your timezone
#define DST_TIMEZONE_OFFSET -6    // Day Light Saving Time offset (-6 is mountain time)
#define ST_TIMEZONE_OFFSET -7    // Standard Time offset (-7 is mountain time)
```

You must first set the `WIFI_SSID` and `WIFI_PASS` to match your local wireless network. The ESP8266 uses this to login to your WiFi network and request the time once every five minutes from NTP servers on the net. The final item of configuration is the specification of your location's time zone offset (in hours) from Coordinated Universal Time (UTC). As shown above I live in the mountain time zone that has a seven hour time difference during non daylight saving time (DST) and six hours when DST is in use. See en.wikipedia.org/wiki/List_of_tz_database_time_zones for a list of timezones around the world.

Make sure you select the NodeMCU 1.0 as your board type in the Arduino IDE before you compile or you will receive plenty of error messages. Once you have modified and saved the file, compile and upload it to the NodeMCU module via a USB cable. If time is not displayed quickly bring up the Arduino Serial monitor and hopefully you will be able to tell what the problem is. If time is displayed, you should be good to go.

If power to your clock is ever lost the clock will set in the new time and date when power is restored and the clock boots up and connects to the Internet. If your WiFi network or modem goes down it may not come back up as fast as your clock but never fear the connection Finite State Machine (FSM) in the code will retry continually until normal clock operation is restored. You never have to set the clock's time or whether or not DST is in effect.

I would like to acknowledge Becky Stern, previously at Adafruit, for giving me the idea of using NeoPixels LED rings to display time.

Conclusions

The finished and operational clock is shown in Figures Five. Mounting the NeoPixel / mat assembly in a small metal frame makes a nice looking DIY desktop clock. This clock has been running at my house for about a year and a half without issue. The pictures of the clock shown in this article are not that great because the NeoPixel LEDs are so bright and the colors so saturated it messes with my camera. Actually the code intentionally reduces the brightness of the NeoPixel LEDs otherwise this clock would be blindingly bright. In person, the colors are just beautiful and pure.

Figure Six shows the clock during a 10 minute event. Every 10 minutes the display of time ceases and the NeoPixels display a rainbow of colors that move across all of the LEDs. Time display returns to normal after completion of the event.

There is also a 15 minute event which flashes colors across all of the NeoPixel LEDs.

The 30 minute event displays the current date using the large ring with each component of the date displayed in a different color. First up, the day of the week is displayed on the LEDs with Sunday being the first day. Next the month is displayed with January being month one, followed by the day of the

month (1..31) and the year minus 2000. This year the year display shows a count of 17. It takes a little practice to read the date from the LEDs but once you are used to it, reading the date is easy.

Chapter Ten – NeoPixel LED NTP Clock

Figure One
NeoPixel LED Clock Schematic

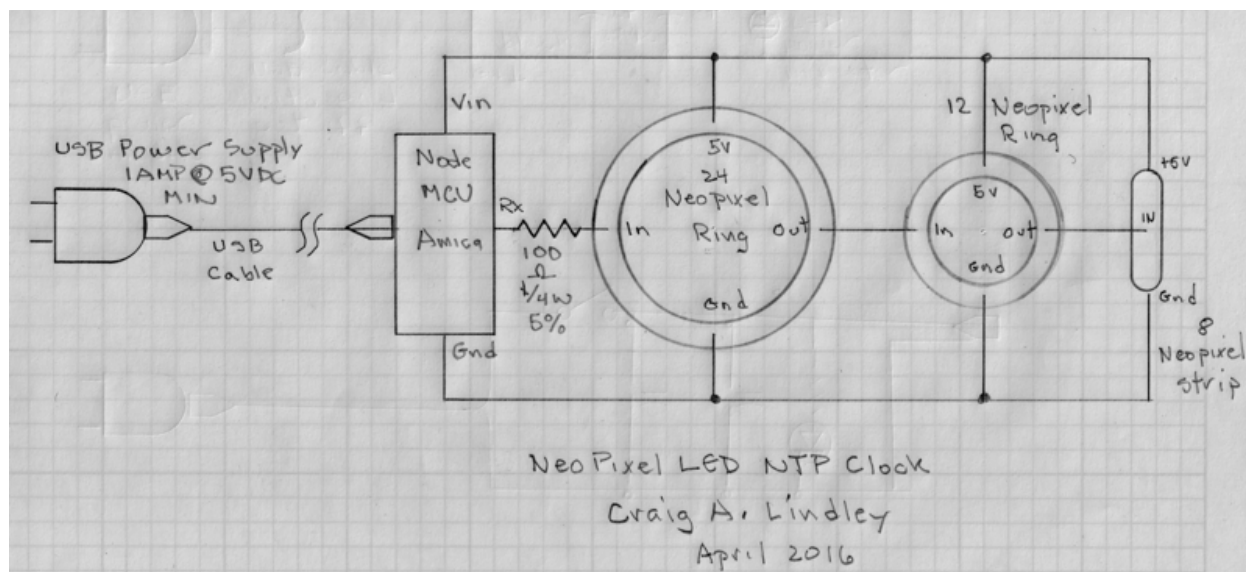
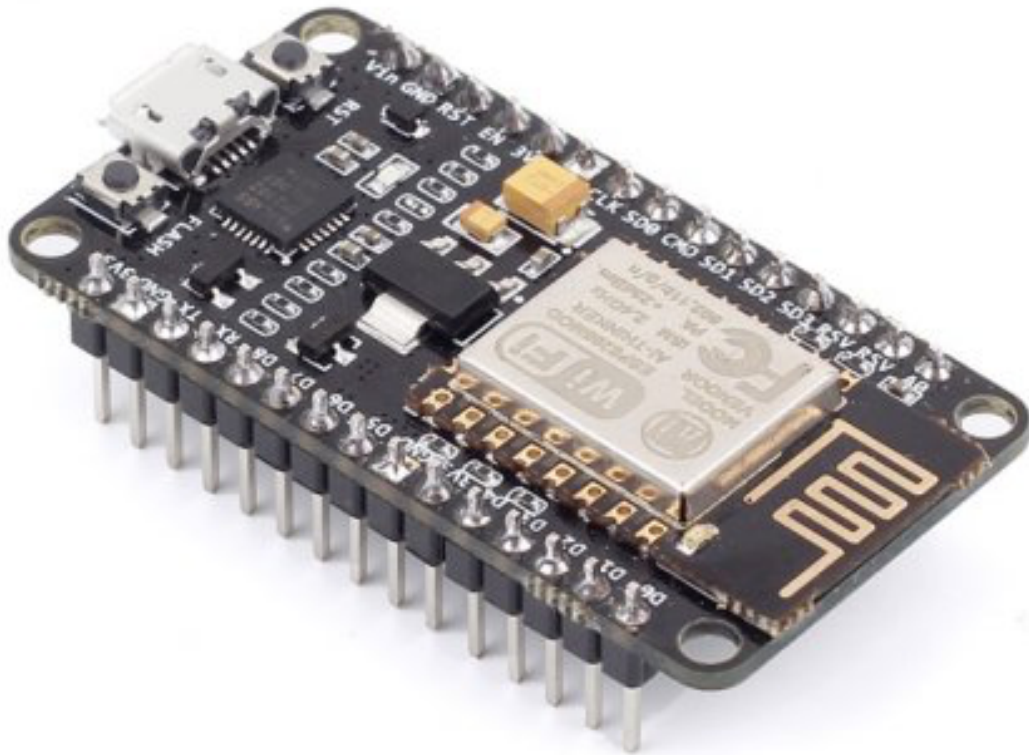


Figure Two
The NodeMCU Amica ESP8266 Module



Chapter Ten – NeoPixel LED NTP Clock

Figure Three
Clock Face with NeoPixel Devices

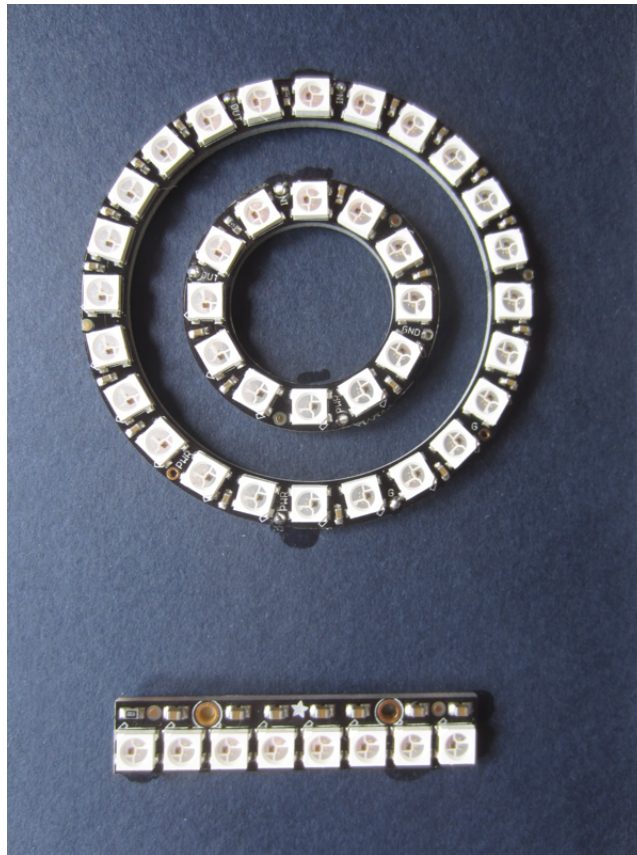


Figure Four
Rear View of Clock
As you can see there is not much to it.

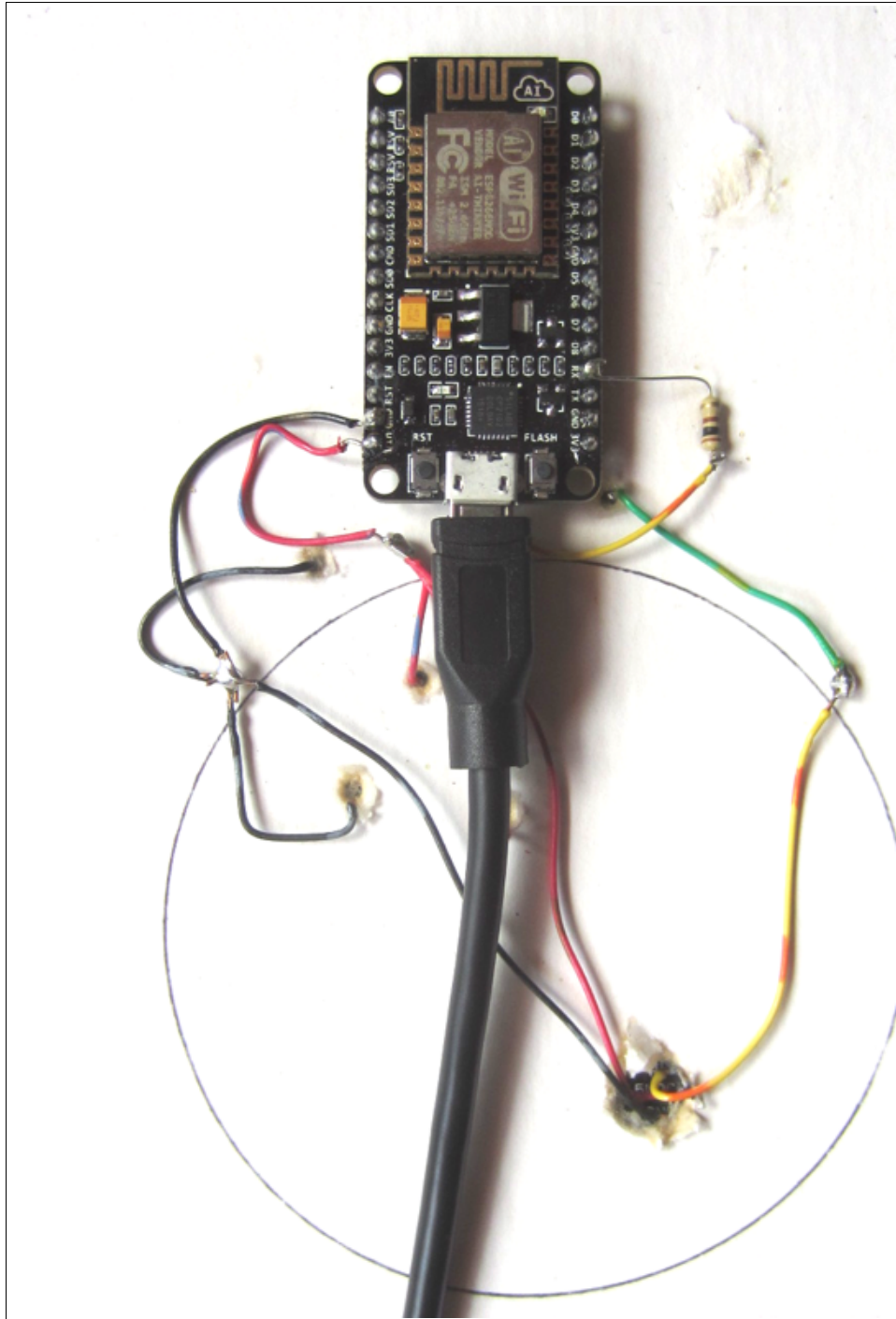


Figure Five
Clock in operation displaying the time



Chapter Ten – NeoPixel LED NTP Clock

Figure Six
Clock displaying the 10 minute event



Snippet #1 ESP8266 & VS1053B Internet Radio

Introduction

Since the ESP8266 has a WiFi interface built in I wondered whether it might be used for an Internet radio application. I didn't think there would be enough memory / performance to have a software based MP3 decoder for decoding the radio stations and have anything left over for additional functionality so I decided to couple it to a VS1053B module instead. The manufacture of the VS1053B chip refers to it as a *Ogg Vorbis/MP3/AAC/WMA/FLAC/MIDI audio codec circuit*.

I had seen these modules around for years but never has a reason to buy one and play with it until now. So I looked around on eBay and found one shipped to my door for \$13.00 US. There are many varieties of these module around and probably any of them would work for this application. The module I bought is shown in Figure One.

In reading the data sheet for the VS1053B I came to an appreciation for all of the functionality built in to this tiny chip. The Internet Radio application discussed here barely scratches the surface of what this chip is capable of. In fact the module I bought has a built in condenser microphone and line inputs for recording audio as well as for audio playback but I have not experimented with that yet. Another surprise was the presents of MIDI capability. This chip contains a midi interface and all of the GM1 and GM2 voices. The MIDI capabilities of this module will be explored in the Snippet #2 later in this document.

So this experimental Internet Radio consists of a NodeMCU Amica module coupled to the VS1053B module with a pushbutton switch for changing the radio station selection. Things don't get much simpler than this.

Hardware

I used a proto board for wiring up the Internet Radio. The operational radio is shown in Figure Two. Jumper wires were used for all connections. The table below shows all of the pertinent connections.

NodeMCU Amica	VS1053B Module	Station Advance PB Switch	Function
D1	DREQ		VS1053B data ready
D2	RST		VS1053B reset pin
D3		PB1	Station advance pushbutton pin
D4	xDCS		VS1053B data chip select
D5	SCK		SPI clock

Snippet #1 ESP8266 & VS1053B Internet Radio

NodeMCU Amica	VS1053B Module	Station Advance PB Switch	Function
D6	MISO		SPI data interface
D7	MOSI		SPI data interface
D8	xCS		VS1053B command chip select
Gnd	Gnd	PB2	Ground connection
Vin	5V		Power connection

The circuit is powered via the USB connection to my computer. I used headphones for listening to the radio stations but a power amplifier could be connected instead if you want to drive speakers. The fidelity is remarkably good when listening to stations that have bit rates of 128 Kbps or greater.

Software

The software for the Internet Radio is comprised of three files as described below:

File	Function
ESP8266_VS1053_InternetRadio.ino	This code creates instances of and initializes the circular buffer, the VS1053 driver and a WiFi client that are needed for the Internet Radio. It also handle connection to the local WiFi network. The radio stations used for testing are also defined here.
CircularBuffer.h	This file implements a circular buffer which buffers the data received from the network and sends it to the VS1053 chip in the 32 byte chunks it requires.
VS1053.h	This is the driver for the VS1053B chip. It manages the low lever SPI interface for interfacing to the chip. There are routines for reading and writing the chip's registers, for verifying a VS1053 is connected to the NodeMCU module, for starting and ending playback, for transferring audio data to the chip and for controlling the playback volume.

For testing purposes I have created the following definitions in *ESP8266_VS1053_InternetRadio.ino*.

```
// Station Descriptor
typedef struct {
    const char *host;
```

Craig A. Lindley's Micro Controller Projects – Volume 1 - ESP8266

```
int port;

const char *request;

} STATION;

// Define some test stations of various bit rates
STATION stations [] = {

    // Beatles radio 128 Kbps
    "64.40.99.2", 8000, "/",

    // Classic rock Florida 160 Kbps
    "us2.internet-radio.com", 8046, "/",

    // Smooth Jazz Planet 192 Kbps
    "airspectrum-ir.cdnstream.com", 8000, "/1258_192",

    // Venice Classic Radio Italia 128 Kbps
    "174.36.206.197", 8000, "/",

    // Nashville FM [24/7 Nonstop Country Music] 192 Kbps
    "46.231.87.20", 8300, "/",

};

#define NUMBER_OF_STATIONS (sizeof(stations) / sizeof(STATION))
```

Three items of information are required to access an Internet Radio station and these are grouped together in the *STATION* descriptor datatype shown in the code above. The items are a domain name or IP address of the radio station's host server, the port number on the server for the connection and a request string used to form the HTTP GET request which starts the streaming of the audio data.

In the code I created an array of *STATION* called *stations* which describes five radio stations of different bit rates and different genres. Stations include one dedicated to Beatles music, a classic rock station, a smooth jazz station, a classical music station and a country station. Something for everyone.

The *selectStation* function manages the connection to an Internet Radio station.

```
void selectStation(void) {

    // Lower volume during change of station
```

Snippet #1 ESP8266 & VS1053B Internet Radio

```
vs1053.setVolume(0);

// Stop playback
vs1053.stopPlayback();

if (client.connected()) {
    // Disconnect from current connection
    client.stop();
}

// Clear the circular buffer
cb.clearBuffer();

// Get parameters of station to play
STATION station = stations[stationIndex];

// Attempt connection to selected host station
if (! client.connect(station.host, station.port)) {
    while (true) {
        Serial.println("Connection failed");
        delay(1000);
    }
}

Serial.printf("Connection succeeded to Internet radio station: %s\n", station.host);

// Send the GET request to the host
client.print(String("GET ") + station.request + " HTTP/1.1\r\n\r\n");

// Start playback
vs1053.startPlayback();

// Raise volume
vs1053.setVolume(DEFAULT_VOLUME);
}
```

This function is called in the program's *loop()* function whenever the station advance pushbutton switch is clicked. The *loop()* functions also reads data from the network and stores it into the circular buffer

and then doles out the data from the circular buffer to the VS1053B in *VS1053_CHUNK_SIZE* (32 byte) chunks.

The *stationIndex* variable is incremented every time the station advance pushbutton is clicked. A check is made to make sure it stays in the range 0 .. 4 so it doesn't overrun the *stations* array.

Conclusions

The components used in this Internet Radio are relatively inexpensive so why not try and build yourself an Internet Radio. The website *internet-radio.com* is a great source of radio station data. To discover the required station parameters (host, port and request) select a station and then click the .pls link to bring up the station's information. For example I chose the *Classic rock Florida* station. Clicking the .pls link yields the following:

```
[playlist]
NumberOfEntries=1
File1=http://us2.internet-radio.com:8046/
```

The host server is then: us2.internet-radio.com

The host port is: 8046

The request string is: /

Use this technique to add to or modify the stations I have coded into the software.

As always, have fun experimenting with your Internet Radio.

Snippet #1 ESP8266 & VS1053B Internet Radio

Figure One

The VS1053B Module

One of many varieties that should work.

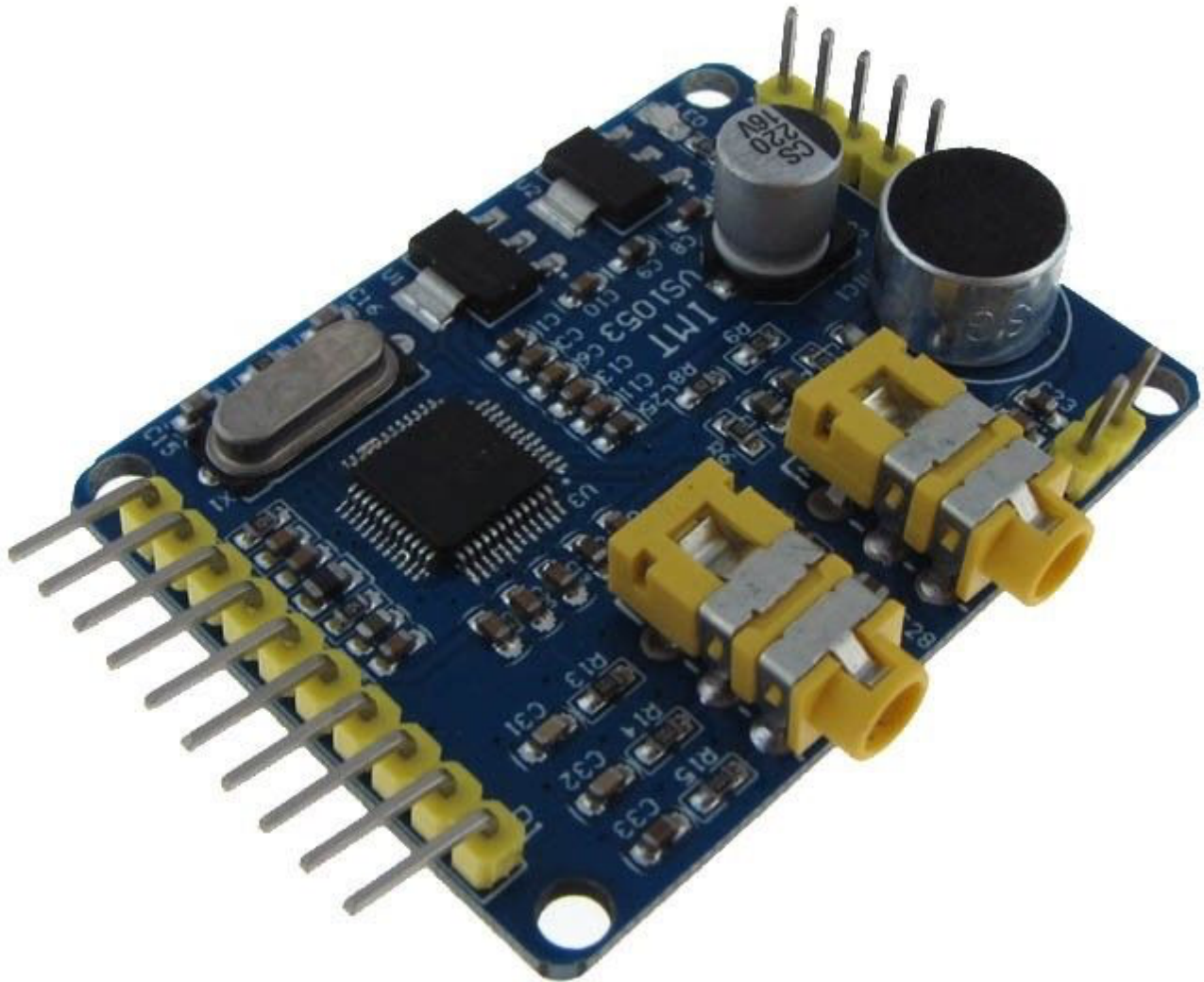
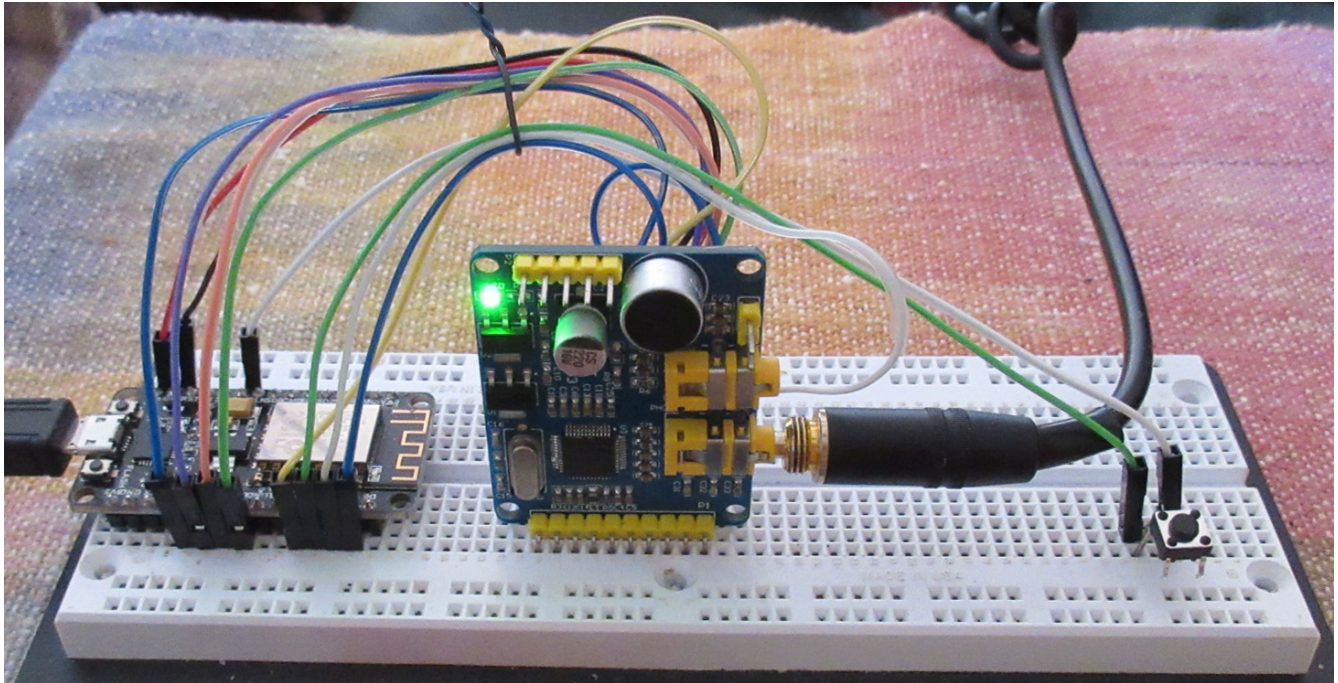


Figure Two

The working breadboard of the Internet Radio

The pushbutton on the right is the station advance switch



Snippet #2 ESP8266 & VS1053B MIDI

Introduction

As mentioned in Snippet #1, the VS1053B device is really quite a marvel in terms of embedded functionality. In that Snippet I used the VS1053B to decode MP3 Internet broadcasts to produce a prototype of an Internet Radio. This worked extremely well so I decided to investigate another of the VS1053B's capabilities, MIDI. It was hard for me to believe that in addition to all of the built in decoders: Ogg Vorbis, MP3, AAC, WMA and FLAC, the VS1053B also had a General MIDI implementation which could support a physical MIDI interface and had two banks of General MIDI instruments along with effects like reverb.

Unfortunately the VS1053B module I bought (see Figure One) did not allow for easily connecting a physical MIDI interface but I found out in my investigation that MIDI could still be supported programmatically which is the technique I pursued here.

My initial MIDI prototype used the MIDI software interfaces I coded up for the VS1053B to just play sequential notes of a specified instrument. While this proved the MIDI interfaces were working, it was kind of boring so my second prototype, which I present here, plays Beethoven's Fur Elise in its entirety. This is a much more satisfying demo in my estimation.

The pushbutton switch used in the Internet Radio snippet to change radio stations is used here to change the instrument the song plays in. Every time this pushbutton is clicked Fur Elise starts over but is played with a different MIDI instrument. There are 127 instruments available plus multiple drum kits.

The code provided with this snippet uses just a small portion of the MIDI functionality provided by the VS1053B module but it should be enough to get you going on projects of your own.

Hardware

I used a proto board for wiring up the MIDI prototype as shown in Figure Two. Jumper wires were used for all connections. The table below shows all of the pertinent connections.

NodeMCU Amica	VS1053B Module	Instrument Advance PB Switch	Function
D1	DREQ		VS1053B data ready
D2	RST		VS1053B reset pin
D3		PB1	Instrument advance pushbutton pin
D4	xDCS		VS1053B data chip select

Snippet #2 ESP8266 & VS1053B MIDI

NodeMCU Amica	VS1053B Module	Instrument Advance PB Switch	Function
D5	SCK		SPI clock
D6	MISO		SPI data interface
D7	MOSI		SPI data interface
D8	xCS		VS1053B command chip select
Gnd	Gnd	PB2	Ground connection
Vin	5V		Power connection

The circuit is powered via the USB connection to my computer; the same one used for programming. I used headphones for listening to the MIDI playback but a power amplifier could be connected instead if you want to drive speakers. Some of the MIDI instruments provided by the VS1053B module sound surprisingly good, others not so much.

I should point out that this is exactly the same hardware configuration used in the Internet Radio Snippet. That is, you can have Internet Radio or MIDI playback by just changing the code loaded into the ESP8266 module all without moving a single wire. Amazing huh?

Software

The software for the MIDI demo is comprised of the four files described below:

File	Function
ESP8266_VS1053_MIDI.ino	This code creates an instance of and initializes the Midi class for operation. All MIDI functions are called via the created instance. The <i>setup()</i> function in the code initializes SPI and then initializes the <i>midiDriver</i> for operation. The <i>loop()</i> function in the sketch reads the pushbutton switch to see if it is active and then increments the <i>tickCount</i> which controls playback timing. See text for details. The <i>loop()</i> function is also responsible for reading data from the network and sending it to the ESP8266.
FurEliseData.h	This is the song data for Fur Elise by Beethoven contained in a rather large array. Each note to be played is represented by a pair of integer values. The first value of the pair is a tick count which determines when the note should sound and the second value of the pair is the MIDI note number to sound. There are 10 notes defined per line in this file and the double zeros at the

File	Function
	end of the array signify the end of the song.
Midi.h	This file defines a MIDI subclass derived from the VS1053 class. Its <i>init()</i> function initialized the VS1053B chip and then loads a plugin that enables the MIDI functionality. Common MIDI software interfaces like <i>noteOn</i> , <i>noteOff</i> , <i>setInstrument</i> , etc. are defined in this file.
VS1053.h	This is the driver for the VS1053B chip. It manages the low lever SPI interface for interfacing to the chip. There are routines for reading and writing the chip's registers, for verifying a VS1053 is connected to the ESP8266 module, for starting and ending playback, for transferring audio data to the chip and for controlling playback volume.

Due to space limitations I cannot go into MIDI in detail here so if you are not familiar with what MIDI is or how it works you might check out: <https://en.wikipedia.org/wiki/MIDI> or just do a search on the Internet for MIDI information. There are hundreds (thousands ?) of websites out there that can help you understand MIDI.

In brief MIDI itself does not contain any sound but is a digital interface that commands MIDI devices to produce sound when they receive MIDI messages. These messages are comprised of up to three bytes of information consisting of a status byte, which indicates the type of the message, followed by up to two data bytes containing parameters. MIDI messages can be *channel messages*, which are sent on only one of the 16 channels and can be heard only by devices receiving on that channel, or *system messages*, which are heard by all devices. There are five types of messages: Channel Voice, Channel Mode, System Common, System Real-Time and System Exclusive.

The VS1053B MIDI implementation requires that each byte of a MIDI message be delimited with a byte of zeros before being sent to the hardware. Don't ask me why. This is accomplished automatically in the *midiWrite* function from the *midi* class and shown below.

```
// Write Midi command via SDI
// a command byte
// b is operand byte
// optional 2nd operand
void midiWrite(byte a, byte b, byte c = 0) {
    // Buffer sized to pad bytes with 0's
    byte buffer[6];
    memset(buffer, 0, 6);
    buffer[1] = a;
```

Snippet #2 ESP8266 & VS1053B MIDI

```
buffer[3] = b;

buffer[5] = c;

sendBytesSDI(buffer, 6);

}
```

As you can see the third parameter, *C*, defaults to zero if it is not specified in the functions which call *midiWrite*. The function *sendBytesSDI* is defined in the VS1053B driver and it sends the six bytes of data to the VS1053B via SPI.

As an example, the *noteOn* MIDI function calls *midiWrite* to send a *noteOn* message. Its code is as follows:

```
// Midi note on message

// chan is channel to play note on

// note is the Midi note to play

// velocity is how loud to play the note

void noteOn(byte chan, byte note, byte velocity) {

    midiWrite(MIDI_NOTE_ON | chan, note, velocity);

}
```

All other MIDI functions are defined in a similar manor.

Playing a song is a lot more complicated than playing a single MIDI note. Playing a song requires timing information in addition to which notes to play. In the MIDI demo provided here, the Fur Elise song data is broken up into timing and note data and then sent to the VS1053B for rendering as music. The following discussion describes how this is accomplished.

The *loop()* function in the *ESP8266_VS1053_MIDI.ino* sketch is called over and over at a very fast rate. Every time through the loop a check is made to see if the *nextTickTime* has expired, and if so the *tickCount* variable is incremented and a new *nextTickTime* value is calculated from the current system millisecond count plus the *MILLISECONDS_PER_TICK* constant. This constant controls the song playback speed and by default is set to 10 milliseconds/tick. Smaller values of this constant speed playback up and larger values slow playback down.

Every time the *tickCount* variable is incremented the *playSongNote()* function is called to determine if it is time to play the next note of the song. As previously mentioned the song data is made up of tick/note pairs. The *playSongNote* function looks to see what the current note's tick count is and compares that to the *tickCount* passed in. If the tick time of the note has not yet occurred nothing happens. If, however, the time has come to play the note, the note's MIDI value is fetched from the song data and a *noteOn* MIDI message with the note's value and *DEFAULT_VELOCITY* is sent to the VS1053B for playback. This process continues on a note by note basis until a tick time of zero is read from the song's data signifying the end of the song.

The *playSongNote* function restarts song playback when the song ends.

Conclusions

The MIDI implementation within the VS1053B may not compare to a Roland Synthesizer but it is more than adequate for experimentation with MIDI. Given the fact that it comes for free inside of a chip you probably purchased for audio decoding and playback is a real bonus. The ESP8266 and VS1053B combination could be used to add musical playback into projects such as music boxes, kiosks, etc. If nothing else it is fun to tease this functionality out of the surprisingly powerful VS1053B chip.

Snippet #2 ESP8266 & VS1053B MIDI

Figure One

The VS1053B Module

One of many varieties that should work.

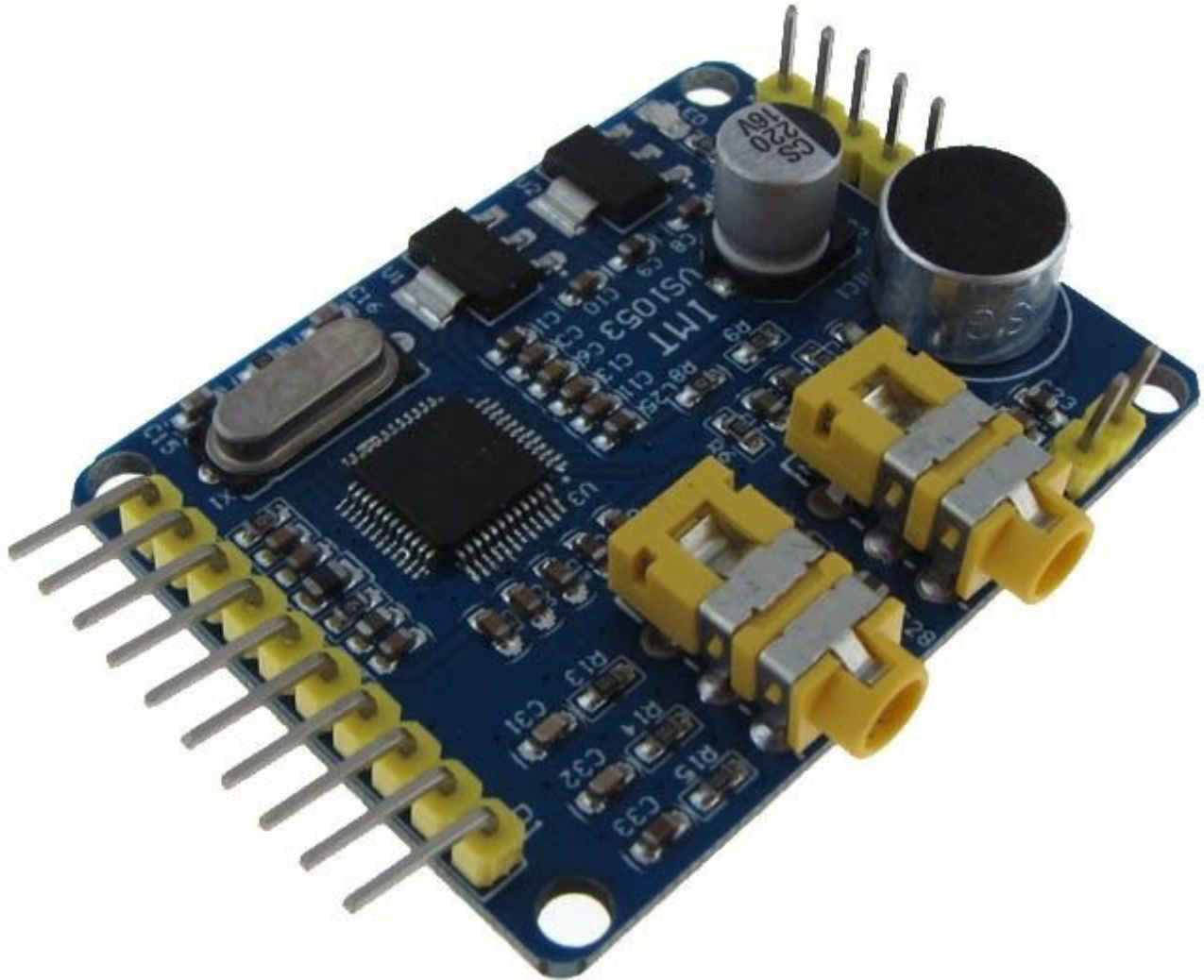
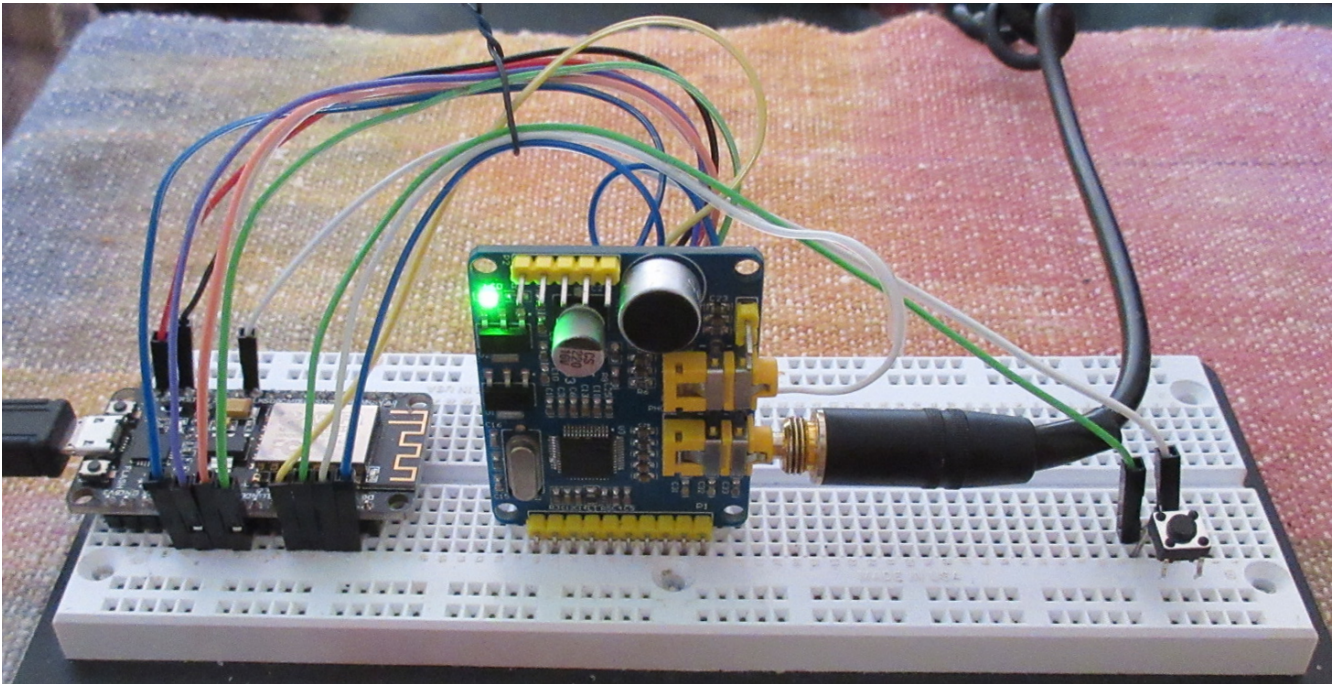


Figure Two

The working breadboard of VS1053B MIDI

Based on the exact same hardware as the Internet Radio described previously

The pushbutton switch on the right is used to change MIDI instruments on the fly



Snippet #2 ESP8266 & VS1053B MIDI