Getting to know NodeMCU and its DEVKIT board

Set up and get started programming for this open source IoT development environment

Michael Yuan

August 07, 2017
(First published June 12, 2017)

This article introduces the open source IoT development board called NodeMCU. One of its most unique features is that it has built-in support for wifi connectivity, and hence makes IoT application development much easier. While Arduino boards might provide greater flexibility, NodeMCU boards provide a more consistent and accessible experience for IoT developers.

Development boards, such as Arduino and Raspberry Pi, are common choices when prototyping new IoT devices. Those development boards are essentially mini-computers that can connect to and be programmed by a standard PC or Mac. After it has been programmed, the development boards can then connect to and control sensors in the field. (You can read more about IoT development hardware in this developerWorks article.)

Because the "I" in IoT stands for internet, the development boards need a way to connect to the internet. In the field, the best way to connect to the internet is by using wireless networks. However, Arduino and Raspberry Pi do not have built-in support for wireless networks. Developers will have to add a wifi or cellular module to the board and write code to access the wireless module.

In this article, I will introduce an open source IoT development board called NodeMCU. One of its most unique features is that it has built-in support for wifi connectivity, and hence makes IoT application development much easier.

What is NodeMCU?

The NodeMCU (Node MicroController Unit) is an open source software and hardware development environment that is built around a very inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains all crucial elements of the modern computer: CPU, RAM, networking (wifi), and even a modern operating system and SDK. When purchased at bulk, the ESP8266 chip costs only $2 USD a piece. That makes it an excellent choice for IoT projects of all kinds.

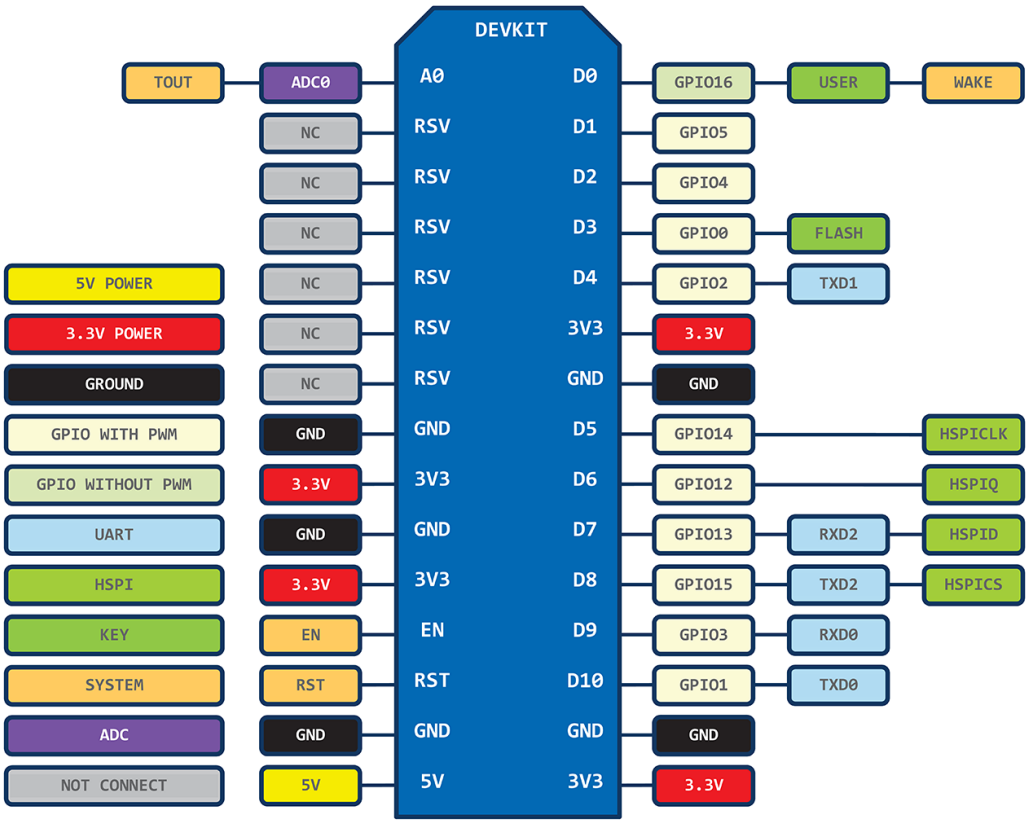
However, as a chip, the ESP8266 is also hard to access and use. You have to solder wires, with the appropriate analog voltage, to its PINs for the simplest tasks such as powering it on or sending a keystroke to the "computer" on the chip. And, you have to program it in low-level machine instructions that can be interpreted by the chip hardware. While this level of integration is not a problem when the ESP8266 is used as an embedded controller chip in mass-produced electronics, it is a huge burden for hobbyists, hackers, or students who want to experiment with it in their own IoT projects.

Borrowing a page from the successful playbooks of Arduino or a Raspberry Pi, the NodeMCU project aims to simplify ESP8266 development. It has two key components.

1. An open source ESP8266 firmware that is built on top of the chip manufacturer's proprietary SDK. The firmware provides a simple programming environment based on eLua (embedded Lua), which is a very simple and fast scripting language with an established developer community. For new comers, the Lua scripting language is easy to learn.
2. A DEVKIT board that incorporates the ESP8266 chip on a standard circuit board. The board has a built-in USB port that is already wired up with the chip, a hardware reset button, wifi antenna, LED lights, and standard-sized GPIO (General Purpose Input Output) pins that can plug into a bread board. Figure 1 shows the DEVKIT board, and Figure 2 shows the schema of its pins.

Figure 1. The NodeMCU DEVKIT board

Figure 2. The NodeMCU pin schema



The NodeMCU DEVKIT board that comes preloaded with the firmware can be purchased for $8 USD a piece, which makes it a very economical device for prototyping and even for production use.

But, what about Arduino, you ask? The Arduino project creates an open source hardware design and software SDK for a versatile IoT controller. Similar to NodeMCU, the Arduino hardware is a microcontroller board with a ready USB connector, LED lights, and standard data pins. It also defines standard interfaces to interact with sensors or other boards. But unlike NodeMCU, the Arduino board can have different types of CPU chips (typically an ARM or Intel x86 chip) with memory chips, and a variety of programming environments. In fact, there is an Arduino reference design for the ESP8266 chip as well. However, the flexibility of Arduino also means significant variations across different vendors. For example, most Arduino boards do not have wifi capabilities and some even have a serial data port instead of a USB port. I feel that NodeMCU provides a more consistent and accessible experience for IoT developers.

1. Getting Started with NodeMCU

While this article assumes that the user uses a late model Mac OS X computer, if you use a Linux machine, the instructions are essentially the same. For Windows users, you'll need to install Python and pip first.

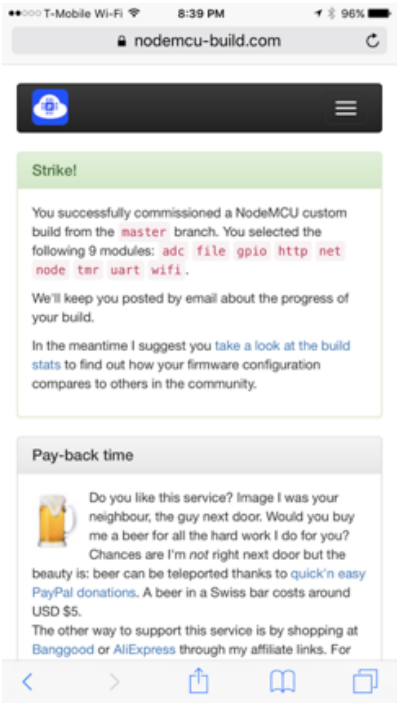
1a. Install a USB-to-serial-port driver

Download and install a USB-to-serial-port driver application on your computer. I used one from the Silicon Labs website. You need this driver application for your computer to communicate with the NodeMCU board by using a standard USB cable.

1b. Create your custom NodeMCU firmware

Create your own custom NodeMCU firmware. Go to the NodeMCU Custom Builds web site, and choose the options that you want.

Figure 3. Configure and create your own NodeMCU firmware



Besides the standard (default) choices, below are some interesting options that you might want to choose:

- **ADC:** Support for measuring analog input (voltage level) on the NodeMCU board's A0 pin.
- **HTTP:** Support for writing code to handle HTTP requests.
- **SSL / TLS:** Support for HTTPS secure connections.
- **MQTT:** Support for the MQTT protocol to send data to other devices or servers using a publish/subscribe model over TCP/IP.
- **Websocket:** A convenience library to access websocket-based web services.
- **DHT:** A convenience library to read data from DHT family of environmental sensors.
- **Enduser setup:** Support a "capture portal" to let the user enter her own wifi password, without having to hardcode wifi credentials in application code.

After the firmware is created, the system will email you a link to download your firmware binary file. Choose the build with float number support unless you know that your application will deal with integer numbers only.

1c. Flash the firmware to the NodeMCU device

Install the `esptool` Python library, and use it to flash the firmware that you just downloaded to the NodeMCU device.

Using Python and pip, run the following command to install `esptool` and all its dependencies.

```
sudo pip install esptool
```

Connect the NodeMCU device to the computer using a USB cable. The blue light on the NodeMCU will flash briefly upon connection. Then, run the following command to flash the firmware that you just downloaded (the `*.bin` file) to the NodeMCU.

```
esptool.py --port=/dev/cu.SLAB_USBtoUART write_flash -fm=dio -fs=32m 0x00000 nodemcu-master-10-modules-2017-01-28-02-40-34-float.bin
```

On a Mac, the port is `/dev/cu.SLAB_USEtoUART` as described above; on Windows, the port might be `COM8`; on Linux, the port might be `/dev/ttyUSB0`.

1d. Run some Lua code

Connect to your NodeMCU, and run some Lua code. Fortunately, when you installed `esptool`, the system also installed `miniterm` as a dependency. So, go ahead and run the following command.

```
miniterm.py /dev/cu.SLAB_USBtoUART
```

Now, press the RESET button on the NodeMCU board, and then press the RETURN key on your computer keyboard a few times. You will first see some random characters, and then finally you'll see an interactive command prompt.

```
--- Miniterm on /dev/cu.SLAB_USBtoUART  9600,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
###@######G##p##########]#bvFD##########
>
>
```

On the command prompt, you can now run Lua scripts. Try this command:

```
print ("Hello World")
```

Now, you can experiment with it. A good place to start is simple code snippets from the "Examples" section of the NodeMCU website.

As an alternative to `miniterm`, you can use the ESPlorer GUI tool. However, you'll need to have Java installed on your computer.

1e. Update the init.lua script

Whenever the NodeMCU board powers up or resets, it executes the `init.lua` script. This script is the starting point of your application. Below is an example `init.lua` script, which flashes the on-board red LED light every second.

```
-- D0 is the LED on the NodeMCU board
lpin = 0
-- timer to run every 1000ms
tmr.alarm(1, 1000, tmr.ALARM_AUTO, function()
    -- Turn on the LED
    gpio.write(lpin, gpio.LOW)
    -- Keep it on for 100ms
    tmr.delay(100000)
    -- Turn off the LED
    gpio.write(lpin, gpio.HIGH)
end)
```

Use the `luatool` program to install the `init.lua` script onto the device. You can download `luatool` from this GitHub repo. Then, run the following command from the same folder as the `init.lua` file.

```
python luatool.py --port /dev/cu.SLAB_USBtoUART --src init.lua --dest init.lua --verbose
```

From my experience, you will need to run `miniterm` first and see the Lua command line prompt. Then, use `CTRL-]` to exit `miniterm,` and then run `luatool.py`. The reason is that `luatool` is less tolerant to noise in the serial port. So, we should run `miniterm` first to clear up the channel.

2. NodeMCU programming basics

Thus far, you have seen how to load and run Lua applications on NodeMCU. In this section, let's review some basic techniques for running NodeMCU apps.

2a. Getting on wifi

A key feature of the NodeMCU is its out-of-the-box wifi capabilities. In the `init.lua` script, you can connect the NodeMCU device to any wifi network with a few lines of code.

```
-- setup Wifi
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","password")
```

Notice that you will need to know the network name and password in order to join. A common technique is to loop through a list of known network name and password pairs if the device might be placed in several different environments.

2b. Connecting to the internet

The NodeMCU SDK contains an HTTP module for making HTTP requests over the internet. When you build your NodeMCU firmware, you will need to select the HTTP options to include this module. The code snippet below shows how to make an HTTP GET request, and execute some code upon completion of this request. In the callback function, the `code` argument is the HTTP return value (for example, 200 indicates success and 404 indicates that the URL is not accessible), and the `data` argument is the content in the HTTP response message.

```
geturl = "http://www.ibm.com/"
    http.get(geturl, nil, function(code, data)
        -- Turn off the red LED on NodeMCU board
end)
```

2c. Accessing the GPIO pins

The General Purpose Input Output (GPIO) pins are digital pins on the NodeMCU DEVKIT board. Each pin can have only two states: a low voltage state and a high-voltage state, representing 0 and 1 respectively. From the NodeMCU Lua application, you can read the state from each pin, and then set the state.

```
-- Read the state of GPIO PIN #5. The val value is 0 or 1
gpio.mode(5, gpio.INPUT)
val = gpio.read(5)


-- Set the state of GPIO PIN #5 to be HIGH
gpio.mode(5, gpio.OUTPUT)
gpio.write(5,gpio.HIGH)
```

2d. Reading analog signals

While the GPIO pins are digital pins, some IoT sensors send in data as analog signals. That is, the voltage of the input wire represents the data. For example, the actual voltage level from a temperature sensor might indicate the temperature reading. On the NodeMCU DEVKIT board, the A0 pin can function as an ADC (Analog to Digital Convertor) pin. When an input wire is connected to the A0, its voltage level between 0 to 3.3V will be converted to an integer number between 0 and 1024. The code snippet below shows how to read an analog value from A0 pin.

```
-- Read the voltage level on A0. The v value is from 0 to 1024
adc.force_init_mode(adc.INIT_ADC)
v = adc.read(0)
```

The ADC pin can convert voltage from 0 to 3.3V only. If your sensor produces an analog voltage outside of this range (such as from 0 to 5V), you will need to add a resistor between the input wire and the A0 pin. It is easy to add this resistor when you wire the NodeMCU DEVKIT board on to a breadboard.

Conclusion

In this article, I introduced you to the NodeMCU DEVKIT. It is a powerful, easy to use, and yet very low-cost solution for IoT application development. For IoT beginners, I feel the NodeMCU is one of the best options to go from prototyping all the way to production.

In my next article, I use NodeMCU as an example to show you how to develop a complete IoT sensor solution with back-end MQTT services.

Related topic

- Here's another developerWorks tutorial that uses a NodeMCU board in its IoT prototype.