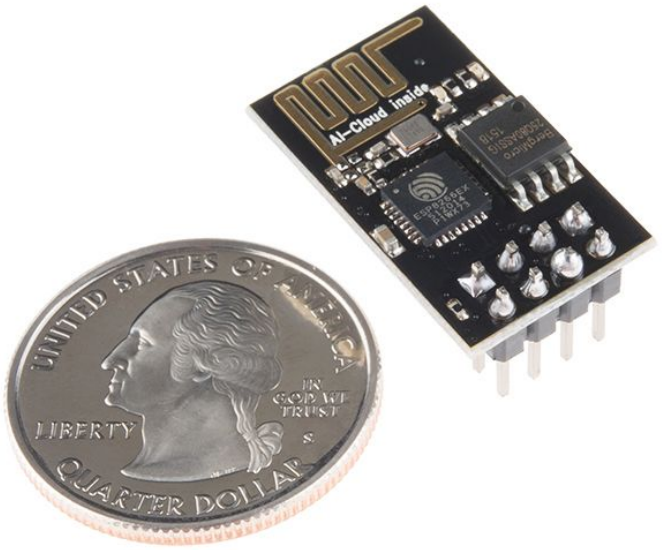
Hackathon 2016
Jason Fowler

Cloud Status IoT
Using $2 esp8266 wifi chip



Introduction

The esp8266 wifi chip is an inexpensive IoT device that can host code including turning it into a web server. For the 2015 Hackathon, I successfully demonstrated turning off and on an LED by clicking a link hosted on the esp8266 chip.

For 2016 Hackathon, my goal is to use the esp8266 to poll a remote web site hosting the status of the Cloud Platform. I will need to be able to demonstrate turning 2 LEDs off and on, remotely from anywhere on the internet, and obtain a simple status from the Cloud Platform to poll for to determine what color the status IoT should display.

Challenges

Here is what I need to do to make this a successful hack:

1) Turn 2 LEDs on and off instead of 1 like last year's hack.
2) Write test code to demonstrate turning LEDs on and off from anywhere on the internet as opposed to within a local network like last year's hack.
3) Determine exactly what Cloud Platform status is, and how to quantify it as either a "0" or a "1" - or in this case, "green" or "red".

Turn 2 LEDs off and on

DEMONSTRATION:

http://www.youtube.com/watch?v=255wpUbPVVU

Remote Control of LEDs from anywhere on Internet

Not so simple. I need web server code hosted somewhere on the internet where any esp8266 chip can poll to obtain Cloud Platform status. Not only that, I have to be able to demonstrate turning the LEDs on and off manually for a test case because the Cloud Platform may be too stable to generate a bad status.

DEMONSTRATION:

https://m.youtube.com/watch?v=GdRhjN-nK2k

How to quantify Cloud Platform status

We in DevOps run a number of monitoring systems (redundancy plus different data types for different "customers"). DataDog is used by both teams to generate overviews of the platform. One status check which can almost guarantee the health of the system in one single number is "Timeouts Per Second". Generally "Timeouts Per Second" (t/s) sit at around 27-29 but when the proverbial "poop hits the fan" that number goes up to 40 or more, based on history of outages, etc.

But how do I get that number onto the web site for the esp8266 chip to poll for? The DataDog API and my old hack-friendly Ruby.

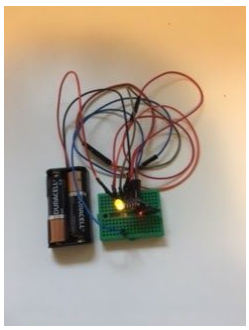
The latest status is 29.42 and the json is:

```
{"status":29.42}
```

Put the ruby file in cron to run every 5 mins:

```
*/5 * * * * /usr/share/nginx/html/status-datadog.rb
```

Now I need to write code to upload to the esp8266 to poll for this json on a remote web site.



Putting it All Together

Once I have an esp8266 capable of parsing json code on the web server, it's time to put it all together. I no longer need the FTDI as I will be running on battery power alone. The esp8266 runs wifi so with battery power it can stand on it's own running the code to get the status.

Components are:

- esp8266 on battery power connected to a green and red led running code to parse the remote web server
- web server running both test php code to demonstrate changing status, and a status json file for the esp8266 to parse
- ruby file run every 5mins on web server to get status from the DataDog API and write it to the status json file

DEMONSTRATION:

Using test admin page to turn status green and red:

http://www.youtube.com/watch?v=Okj9AG_GZrw

Success!

DEMONSTRATION:

Running ruby code to turn status green and red:

http://www.youtube.com/watch?v=hSVeMxlDLjM

Success, again!

Issues Encountered

1) The esp8266 is really flakey for uploading code. I used the Arduino IDE but that is not the issue. The issue is the hardware communication from my laptop (running Linux), to the FTDI FT232 USB to Serial connector, to the esp8266 chip itself. Often I'd get strange errors:

    warning: espcomm_send_command: cant receive slip payload data

    warning: espcomm_send_command(FLASH_DOWNLOAD_DATA) failed

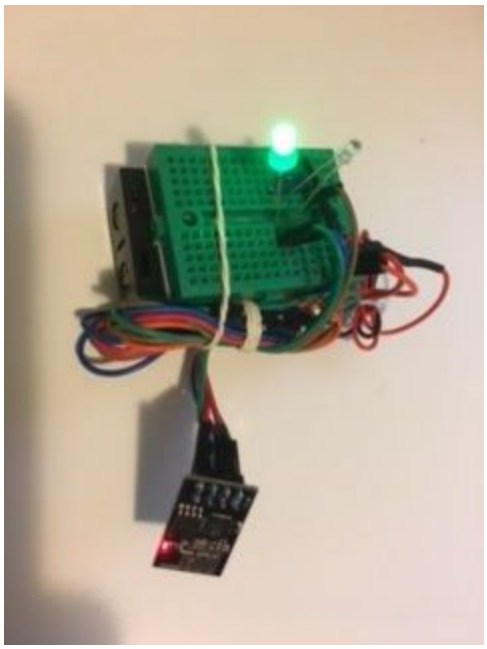
    the selected serial port warning: espcomm_send_command: cant receive slip payload data

    To upload new code, I have to unplug the USB FTDI, remove positive from the esp8266 to reset it, then do the reverse. Often there were errors such as the above, which could be due to the hardware communication, the esp8266 chip, Linux and USB port issues, or even the Arduino IDE doing silly things in Java.

    2) Figuring out how to get a status of either "good" or "bad" only from the Cloud Platform, where to get the number from (platform monitor, Datadog, something new?).

    3) Learning the Datadog API quickly and writing code to access it. Good examples in the Datadog API docs.

    4) Putting it all together into one simple little device with one simple little web site took a fair amount of effort and was worth it.



Conclusion

I really enjoyed working with the esp8266, the Arduino IDE, and the DataDog API. The possibilities are endless. Next year maybe I'll add servo motor control via the esp8266 and control a device of some kind to move like a clock based on the number of timeouts: eg, we should never see 5mins to midnight on the doomsday clock.