A Hands-on Introduction to ESP8266

Sensors for the Home // OSHCamp 2015 Workshop
Omer Kilic <omerkilic@gmail.com>, Version 270915a

Introduction

This workshop explores the use of ESP8266, a pretty nifty WiFi enabled microcontroller device, in the context of a home sensor device. It is based around the ESP8266 Lab Kit (Referred to as the "ELK" in places) and the NodeMCU firmware which brings the Lua programming language to the ESP8266 platform.

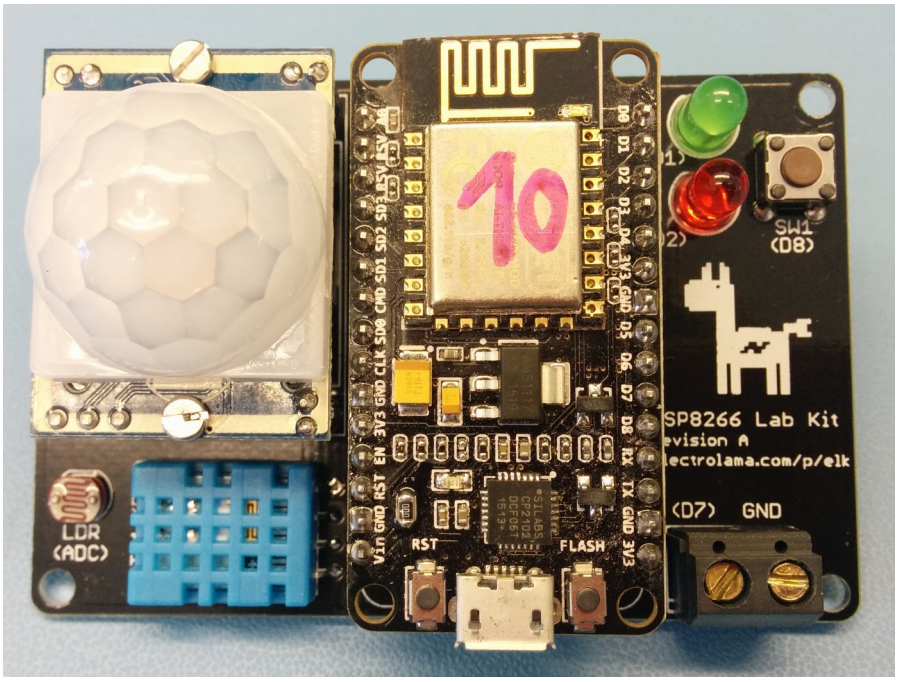
Material for this workshop, including the most up to date copy of this lab manual can be found at: http://electrolama.com/p/elk/

The ESP8266 Lab Kit

The ESP8266 Lab kit is a simple "breakout" for the NodeMCU board (which is in turn a breakout and the support circuitry for the ESP-12E module) that adds a couple of sensors and simple input/output devices to the base ESP8266 system. These are namely:

- HC-SR501 PIR Sensor: Detects motion
- DHT11 Temperature/Humidity Sensor
- A light dependent resistor: To detect ambient light levels
- 2 x LEDs, a push-button and a screw terminal for a magnetic door/window switch

The schematics and the design files can be found under the `hardware/` directory and the assembled ELK boards should look a bit like:



Useful bits of information and links/references

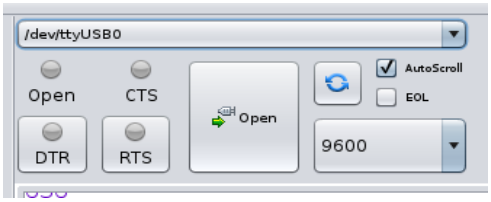
- High level information about this workshop
  - http://electrolama.com/p/elk/
- Repository for the ESP8266 Workshop
  - https://github.com/electrolama/esp8266-workshop/
- Link to a zipball of the `esp8266-workshop` repository (you should grab this)
  - https://github.com/electrolama/esp8266-workshop/archive/master.zip
- Design files for the ESP8266 Lab Kit (ELK), including the schematic
  - https://github.com/electrolama/esp8266-workshop/tree/master/hardware
- ELK builds on top of the NodeMCU project (both hardware and firmware)
  - http://nodemcu.com/index_en.html
- NodeMCU board reference material
  - https://github.com/nodemcu/nodemcu-devkit-v1.0
- NodeMCU Lua Library/API documentation
  - https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_en
- A very nice Lua programming primer/reference
  - http://esp8266.co.uk/tutorials/lua-basics/

Exercise 0 – Getting Started with ESPlorer

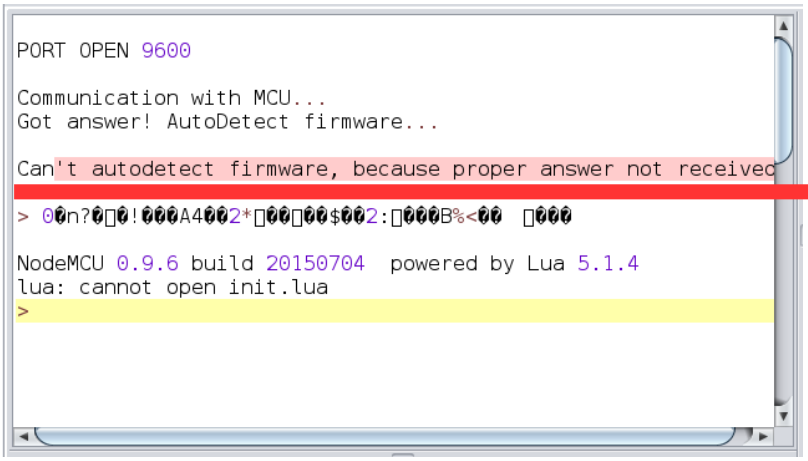
This exercise will get you set up with the ESP8266 Lab Kit and the ESPlorer software envionment that we will be using today. ESPlorer is a simple GUI environment for the NodeMCU Lua firmware that contains a code editor and a serial terminal in one place and while it has its oddities, it's a great tool to start tinkering with Lua on the ESP8266.

Important note: ESPlorer requires a Java JVM installation to be present on your machine. Make sure you have a copy of Java installed on your machine before proceeding with the rest of the instructions.

1. **Fire up ESPlorer:** Head over to the `tools/ESPlorer` directory and either run `./launch_esplorer.sh` (Linux/OSX) or `launch_esplorer.bat` (Windows)
2. **Familiarise yourself with ESPlorer:** The left pane contains the text editor that we will be centering our code snippets in and the right hand size is the serial connection to the device.
3. **Plug your ESP8266 Lab Kit to your computer:** You are going to require a Micro USB cable for this. Hook one end of the cable to a spare USB port on your machine and the other end to the NodeMCU board.
4. **Figure out what COM port your NodeMCU board has acquired:** On Linux this will be something along the lines of `/dev/ttyUSB0`, Windows will give you COMx where x is an integer and OSX will give you `/dev/tty.usbserial-XXX`.
5. **Open the COM port you've noted on ESPlorer:** Select the correct serial port from the dropdown menu and press Open button:

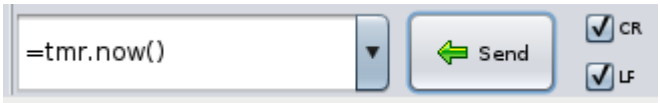


6. **Observe the serial connection:** There is a chance you will get the “Can't autodetect firmware...” error message here, press the “RST” button on the NodeMCU board and you should see the NodeMCU banner printing the version numbers:

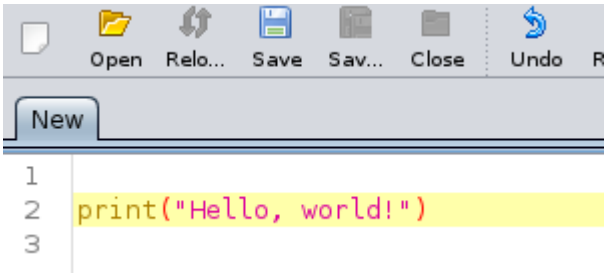


PRESS “RST” BUTTON HERE

7. **Send a sample command to the device:** The bottom right section of the ESPlorer will allow you to send commands to the device, which will be executed (since it is running a Lua REPL) and results printed back to the screen. To test this facility send the command `=tmr.now()` to the device which should return a large-ish integer number.

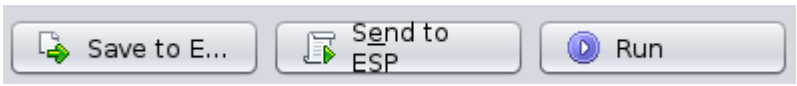


8. **Enter a sample snippet of code to the code editor:** The right pane in ESPlorer is where you will be typing your code into. Enter the following snippet into the code editor:



9. **Save the test code:** Press the "Save" button and save this code snippet as `hello.lua`

10. **Send hello.lua to the device:** Press the "Save to ESP" button, which will send hello.lua to the device, save it into the FLASH memory and execute it. For reference, "Send to ESP" will only send your code to the device (without saving it to the FLASH) and "Run" will execute the filename that you have open from the FLASH memory, if it happens to exist on the device.



11. **Bask in the glory of your first Lua program:** Isn't it pretty awesome?

```
file.remove("hello.lua");
> file.open("hello.lua","w+");
> w = file.writeline
> w([[]]);
> w([[print("Hello, world!")]]);
> file.close();
> dofile("hello.lua");
Hello, world!
>
```

Congrats, you are now ready to tackle the rest of the exercises!

Exercise 1 – Turn an LED on/off

(Ref: `code-examples/01-led.lua`)

This example will show you how to set up a GPIO pin to be an output and write a value to it. It also demonstrates how a delay can be created using the `tmr` module.

The ELK has three LEDs:

- `LED_BLUE`: A tiny SMT LED on the ESP12-E module connected to pin index 0
- `LED_GREEN`: 5mm LED on the ELK, connected to pin index 1
- `LED_RED`: 5mm LED on the ELK, connected to pin index 2

This example only uses LED_BLUE. Add the other two LEDs into this program and build a disco blinkenlights, perhaps?

Exercise 2 – Handle a button press (Poll)

(Ref: `code-examples/02-button.lua`)

This exercise checks the value of an input pin (GPIO index 8) that is connected to the puch-button on the ELK. It uses a technique called polling, essentially checking the state of an input pin continuously and acting upon the state change. This happens inside an infinite loop, which will continue running until you explicitly stop the program or if the device runs out of battery.

You might notice some spurious output when you press the button, this is because push buttons are mechanical devices that are somewhat noisy and the value you read from them needs to go through a process called debouncing. If you're interested in learning about debouncing buttons, this is a good article:
http://www.eng.utah.edu/~cs5780/debouncing.pdf

Polling is not actualy the best way to go about handling inputs because you're wasting a lot of processing cycles trying to figure out if things have changed or not. A better way to handle inputs is by using interrupts, another mechanism that is discussed below in Exercise 3.

Exercise 3 – Handle the door sensor (Interrupt)

(Ref: `code-examples/03-door-sensor.lua`)

As Exercise 2, we are trying to read the value of an input pin but this time around instead of continuously checking the state of a pin to detech change (polling), we will be using a mechanism called interrupts.

Every time a button is pressed, the GPIO controller detects that change and fires up an interrupt. In traditional microcontrollers, whenever an interrupt is fired, the operation of the system is halted, program counter and stack are saved and the CPU jumps to a predefined address that contains what's known as the “interrupt service routine” or “ISR” for short. Upon succesful execution of the ISR, PC and stack are restored and the execution of the microcontroller application continues. Since Lua is an interpreted language and we actually have a virtual machine running on the NodeMCU board, the mechanics of how this is handled is a little bit different but ultimately the end result is the same. When a button is pressed (the state of an input changes), a predefined ISR is called.

The previous note about debouncing input is also applicable here. In this case the debouncing of the input happens within the ISR.

The door sensor is a mechanical switch, when the two parts of the sensor come in contact (or in close proximity of each other) the state of the switch changes and that is what we want to detect. It is connected to the GPIO pin that has the index 7.

Exercise 4 – Get the temperature and the relative humidity

(Ref: `code-examples/04-dht11-sensor.lua`)

The ELK has a temperature/relative humidity sensor on board – the DHT11 (details of which (i.e: its datasheet) can be found in `hardware/datasheets/DHT11.pdf`).

NodeMCU firmware, serendipitously, has native support for DHT11 (and friends) so using a simple API call we can get values off the sensor. You might encounter some errors reading the sensor from time to time so implementing error checking is strongly recommended.

Exercise 5 – LDR

(Ref: `code-examples/05-ldr-sensor.lua`)

ESP8266 contains a single channel ADC and while not exactly super accurate, it is good enough for rough measurements. On the ELK, there is an LDR connected to the only analog pin available and it is good for getting a sense of the ambient light in the room.

You will see that a threshold is defined and used to distinguish whether the room is “dark” or “light”. This is super rudimentary but also good enough to detect whether the lights are on or off in a room.

Exercise 6 – It has the Wifis[1]

(Ref: `code-examples/06-wifi-connection.lua`)

This one is pretty simple, we'll get the ELK connected to the Internets via the local WiFi network!

Make sure you get a valid IP address before you proceed to the next exercise.

Exercise 7 – Sending data to dweet.io

(Ref: `code-examples/07-hello-dweet.lua`)

http://dweet.io is a nice service to have a quick and dirty backend for your sensor data. This code snippet generates a dummy (i.e: random!) temperature and realtive humidity value and sends it over. dweet.io also allows you to generate a nice looking dashboard so once you have your data up there do have a play around with it.

HINT: You should change the `DWEET_ID` within the code to something unique so that your data doesn't get mixed up with someone else's!

1 https://www.youtube.com/watch?v=FL7yD-0pqZg

Exercise 8 – Putting it all together

Your mission, should you choose to accept it, is to put everything we've tinkered with so far into a single application and report all sensor data (temperature, relative humidity, ambient light and the door sensor state) to a unique dweet.io dump. You should have everything you need at this point but give us a shout if you need a hand!

Hint: The PIR sensor (connected to GPIO index 11) can use the same mechanism as the door sensor (the GPIO interrupt). Details on how to set the PIR module up can be found in `hardware/datasheets/HC-SR501.pdf`

Conclusion

You now know how to use Lua on the ESP8266. If Lua is not your cuppa, you can also write code for the ESP8266 using Python (via MicroPython), an-Arduino like environment or the official SDK for a lower level C flow. Go out and build exciting stuff!

Any suggestions/improvements and corrections about this material can be sent as a pull request at https://github.com/electrolama/esp8266-workshop or as an email to omerkilic@gmail.com.