

Lab Assignment 6 — *ETHOS-e*, a More Efficient *ETHOS*

“To sleep, perchance to dream!”

(quoted from Hamlet’s Soliloquy, act 3, scene 1)

Overview:

In this lab you’ll modify your Lab03 *ETHOS* program to create *ETHOS-e*, a less power-hungry temperature and humidity measuring device. Because *ETHOS-e* requires much less power than the original *ETHOS*, it is ideally suited for battery-powered operations, such as remote environmental monitoring. At its core, *ETHOS-e* measures and reports temperature and relative humidity information just like *ETHOS* did, but *ETHOS* was quite inefficient since its ESP8266 microcontroller was powered up 100% of the time, but it spent much of that time doing nothing or waiting for input from the serial menu system. But you’re going to change all that! Specifically, you will:

1. Remove the LCD display. All output will be via the serial monitor.
2. Remove the program’s menu since *Ethos-e* doesn’t need input commands.
3. Display temperature and humidity values using a minimum number of characters.
4. Put the microcontroller into “deep sleep” mode between measurements, to save power, and use the “automatic wake-up” feature to wake up the processor after a period of sleep.
5. Use an in-line USB-based Voltage-Current-Power meter to monitor and record (on paper) the input power during normal operation and during deep-sleep.
6. Finally, you will prepare and submit a one-page power savings analysis report.

Once in operation, the new *ETHOS-e* will periodically wake itself up, take a temperature and humidity reading, send the results to the serial monitor, then put itself into back to sleep. Complete design specifications are listed in the Lab Procedure section below, but before you start any design or coding efforts, you’ll need to learn about the microcontroller’s power-saving modes of operation.

Background

This lab centers largely on the ESP8266 deep-sleep mode and how to use it. Below are a few good sources to help you.

Making the ESP8266 Low-Powered with Deep Sleep

This is a nice tutorial on the various ESP8266 power saving modes and includes code examples.

<https://www.losant.com/blog/making-the-esp8266-low-powered-with-deep-sleep>

How to Enable ESP8266 Deep Sleep Mode? Timer Wake-up

This article gives similar information on the power saving modes of the ESP8266 and also includes a method for adding a “wake up” button (although resetting the processor, as you will learn, has the same effect).

<https://www.electronicshub.org/esp8266-deep-sleep-mode/>

Instructions for USB Tester with Full Color Display

This hardcopy article (available in the Files | Technical Documents section on Canvas) explains how to use the MakerHawk USB 3.0 in-line voltage, current, and power meter. You’ll use it to read and record the power consumption values needed for your power savings analysis report.

These are many more articles out there on using the ESP8266 sleep modes. Remember to cite any and all references you make use of!

As part of your power report, please answer these questions.

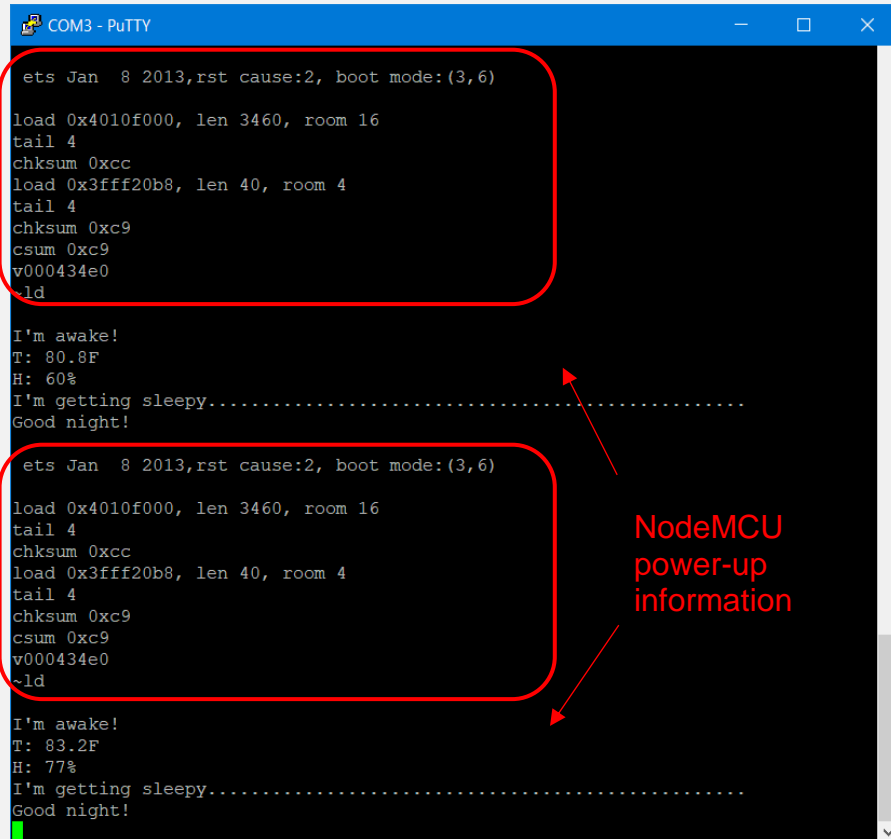
Questions

1. What is the longest time (in microseconds and seconds) the ESP8266 can be in deep sleep?
2. Explain the need for the jumper between pins D0 and RST on the NodeMCU.
3. Regarding the jumper mentioned in question 2. above, when must it NOT be connected between those pins, and why?
4. Does the I2C-based Si7021 temperature and humidity sensor have built-in pull up resistors? If so what value are they? If not, what value should you use when connecting the Si7021 to your NodeMCU?

Lab Procedure:

Begin creating *ETHOS-e* using the following design specifications:

DESIGN SPECIFICATIONS: Once powered up, *ETHOS-e* must measure the current temperature and humidity, and send those values to the serial monitor. Here is a screenshot of a typical serial monitor display. Yours should look very similar.



```
COM3 - PuTTY
ets Jan  8 2013,rst cause:2, boot mode:(3,6)

load 0x4010f000, len 3460, room 16
tail 4
chksum 0xcc
load 0x3fff20b8, len 40, room 4
tail 4
chksum 0xc9
csum 0xc9
v000434e0
~ld

I'm awake!
T: 80.8F
H: 60%
I'm getting sleepy.....
Good night!

ets Jan  8 2013,rst cause:2, boot mode:(3,6)

load 0x4010f000, len 3460, room 16
tail 4
chksum 0xcc
load 0x3fff20b8, len 40, room 4
tail 4
chksum 0xc9
csum 0xc9
v000434e0
~ld

I'm awake!
T: 83.2F
H: 77%
I'm getting sleepy.....
Good night!
```

NodeMCU power-up information

Set all your serial communication to be 74800 N81 (the ESP8266 default). Notice that the ESP8266 reports its status whenever it starts. You've seen this before, probably as "gibberish." By setting the serial monitor's baud rate to you can read the power-up data correctly. But, alas, you can't get rid of it, so we'll just ignore it.

Next, notice that the *ETHOS-e* output begins with "I'm awake!" and ends with "Good night!" In between it measures and displays the temperature and humidity, then announces that it's getting sleepy, followed by displaying 50 "period" characters, each appearing 1/10 sec after the last. Once **5 seconds** have elapsed, "Good night!" is displayed and the processor goes into deep sleep for **10 seconds**. This behavior allows you to measure and record both the active and sleeping power values.

Figure 1. *ETHOS-e* design specifications.

Figure 2 shows the USB Power monitor plugged in and the MCU awake.

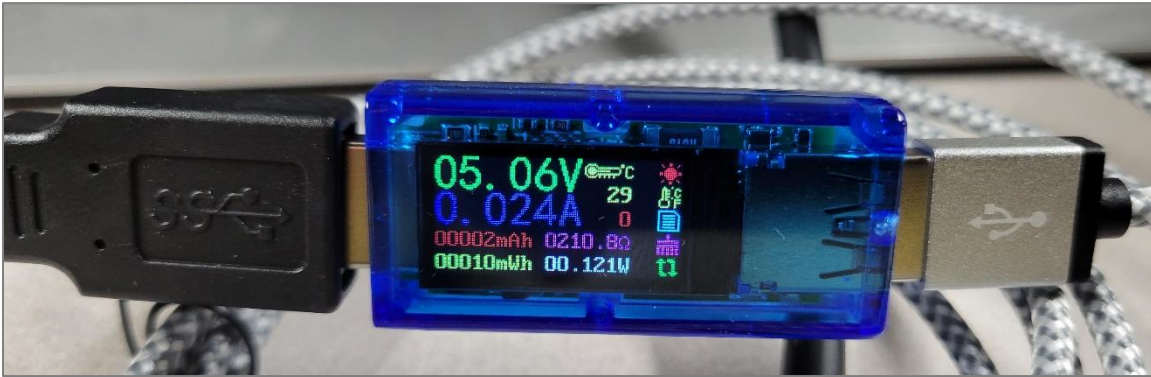


Figure 2. Example *ETHOS-e* “awake” power readings.

Figure 3 shows an example of the MCU asleep. You may get different numbers than this for your particular setup and coding.

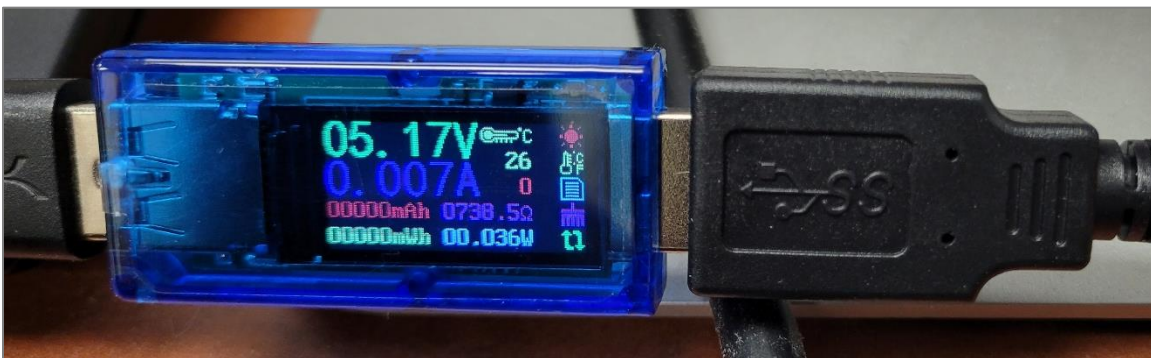


Figure 3. Example *ETHOS-e* “asleep” power readings.

Hints

You do need a specific hardware change to make this work (clearly spoken about in the link above). However, when this is attached, you will NOT be able to program your device. In power shell, what I see is in Figure 4.

```
Serial port COM3
Connecting.....

A fatal error occurred: Failed to connect to ESP8266: Timed out waiting for packet header
*** [upload] Error 2
```

Figure 4. Problem flashing when HW mode in place.

One option is to pull the wire every time you program.

A second option is to turn the device off, press the “FLASH” button, and turn the device back on. This forces the MCU to boot in ready-to-flash mode. Then program. However, the part cannot be rebooted by the flashing routine in this mode, so running the code requires you to power off/on one more time.

Turn-In

This is a **one-week** lab. **No formal lab report is required**, but you must turn in a short report with all questions answered and your one-page Power Analysis report. The report should show data points you collected, your analysis of the resultant power in each of the two modes (awake and asleep), and a final expression showing the power savings in an appropriate form. (Note: Your analysis should include at least five separate readings of both awake and asleep power draw.) Of course, you must save your code in the lab GitHub repo and **demonstrate your working system** to your instructor at or before the beginning of lab **one weeks from today**. During the demo be prepared to answer questions about your design, what challenges you faced, what worked and what didn't, etc. **Have fun!**