**OFF-GRID POWER SYSTEM**

**CODE**

by Durgesh Kolte

**Voltage Aspect:**

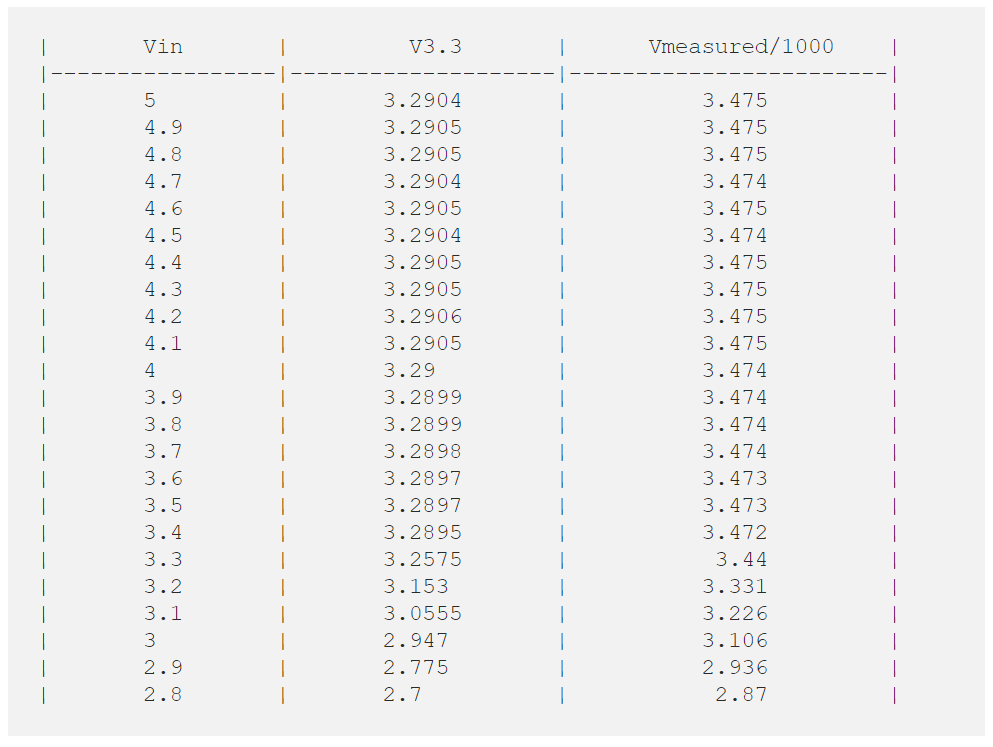
This is where things become interesting. By the way most ESP8266 boards are designed, one cannot read the battery voltage Vin without external components. But there is a method to detect a discharged battery indirectly, by measuring the input voltage of ESP8266 — that would be V3.3.

// Load Wi-Fi library  
#include <ESP8266WiFi.h>  
  
// Replace with your network credentials  
const char\* ssid = "myssid";  
const char\* password = "mypassword";  
  
ADC\_MODE(ADC\_VCC);  
int Batt;  
  
// Set web server port number to 80  
WiFiServer server(80);  
  
// Variable to store the HTTP request  
String header;  
  
void setup() {  
 Serial.begin(9600);  
 while(!Serial);  
  
 // Connect to Wi-Fi network with SSID and password  
 Serial.print("Connecting to ");  
 Serial.println(ssid);  
 WiFi.begin(ssid, password);  
 while (WiFi.status() != WL\_CONNECTED) {  
 delay(500);  
 Serial.print(".");  
 }  
 // Print local IP address and start web server  
 Serial.println("");  
 Serial.println("WiFi connected.");  
 Serial.println("IP address: ");  
 Serial.println(WiFi.localIP());  
 server.begin();  
}  
  
void loop(){  
 // Listen for incoming clients  
 WiFiClient client = server.available();  
   
 if (client) {  
 Serial.println("new client");  
 // an http request ends with a blank line  
 boolean currentLineIsBlank = true;  
 while (client.connected()) {  
 if (client.available()) {  
 char c = client.read();  
 Serial.write(c);  
 // if you've gotten to the end of the line (received a newline  
 // character) and the line is blank, the http request has ended,  
 // so you can send a reply  
 if (c == '\n' && currentLineIsBlank) {  
 // send a standard http response header  
 client.println("HTTP/1.1 200 OK");  
 client.println("Content-Type: text/html");  
 client.println("Connection: close");   
 // the connection will be closed after completion  
 // of the response  
 client.println("Refresh: 5");   
 // refresh the page automatically every 5 sec  
 client.println();  
 client.println("<!DOCTYPE HTML>");  
 client.println("<html>");  
   
 // output the battery value  
 Batt = ESP.getVcc();  
 client.println("Battery voltage is: ");   
 // refresh the page automatically every 5 sec  
 client.println(Batt);  
   
 client.println("</html>");  
 break;  
 }  
 if (c == '\n') {  
 // you're starting a new line  
 currentLineIsBlank = true;  
 } else if (c != '\r') {  
 // you've gotten a character on the current line  
 currentLineIsBlank = false;  
 }  
 }  
 }  
 // give the web browser time to receive the data  
 delay(1);  
 // close the connection:  
 client.stop();  
 Serial.println("client disconnected");  
 }  
  
}

In the code example above, the ESP8266 is configured to read the V3.3 voltage by placing ADC\_MODE(ADC\_VCC); on top of the sketch.

Then, the battery voltage in mV is read as in uint32\_t getVcc = ESP.getVcc();





First of all, we see there’s an offset of about 0.18V between the Vmeasured and the actual V3.3. One can leave it like this or can compensate in the software.

Then we notice that, when the input voltage Vin = 3.3V, we have V3.3 = 3.2575 (it becomes to drop). The ESP8266 senses this small voltage drop, and it measures 3.44V. Further, when the battery voltage drops to 3V (which is the safe margin to discharge LiPo batteries), the readout of the ESP.getVcc() is 3.106V. We can use this value to trigger a deep sleep to keep the battery from discharging, as in the code below:

// Low voltage detection  
// Note that we read Vin, and not the battery voltage,  
// as the battery voltage is not accessible to be measured  
// ESP8266 thing uses AP2112K-3.3V  
// https://www.diodes.com/assets/Datasheets/AP2112.pdf  
// Low DropoutVoltage (3.3V): 250mV (Typ.) @IOUT=600mA  
ADC\_MODE(ADC\_VCC);  
int Batt;  
  
// do other things here  
  
void setup() {  
 // Check battery status  
 // Reads Vin (not battery voltage!!!)  
 // But Vin = battery voltage if battery\_voltage < 3.3V  
 Batt = ESP.getVcc();  
 // If the battery is discharged don't go any further!!!  
 if(Batt < 3100){  
 // Deep sleep for as long as you can  
 ESP.deepSleep(ESP.deepSleepMax());  
 }  
 // your code goes here  
}  
  
void loop() {  
 // your code goes here  
}

What I did here is read the battery voltage when the ESP8266 module starts, and if the value returned by the ESP.getVcc() is below the safe threshold, the ESP8266 goes into deep sleep for as long as it can. Then it will wake up and go back to deep sleep until the battery is recharged.

**Command for Deep Sleep Mode NodeMCU:**

ESP.deepSleep(sleepTimeInSeconds \* 1000000);

**(Also available for ThinkSpeak)**

Off course, after that you can try a more complex example by loading a **ESP8266 Arduino** **WiFi Client example** program that sends data via WiFi to the data.sparkfun.com iot platform:

/\*

\* This sketch sends data via HTTP GET requests to data.sparkfun.com service.

\*

\* You need to get streamId and privateKey at data.sparkfun.com and paste them

\* below. Or just customize this script to talk to other HTTP servers.

\* ESP8266 Arduino example

\*/

#include <ESP8266WiFi.h>

const char\* ssid = "your-ssid";

const char\* password = "your-password";

const char\* host = "data.sparkfun.com";

const char\* streamId = "....................";

const char\* privateKey = "....................";

void setup() {

Serial.begin(115200);

delay(10);

// We start by connecting to a WiFi network

Serial.println();

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

int value = 0;

void loop() {

delay(5000);

++value;

Serial.print("connecting to ");

Serial.println(host);

// Use WiFiClient class to create TCP connections

WiFiClient client;

const int httpPort = 80;

if (!client.connect(host, httpPort)) {

Serial.println("connection failed");

return;

}

// We now create a URI for the request

String url = "/input/";

url += streamId;

url += "?private\_key=";

url += privateKey;

url += "&value=";

url += value;

Serial.print("Requesting URL: ");

Serial.println(url);

// This will send the request to the server

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

unsigned long timeout = millis();

while (client.available() == 0) {

if (millis() - timeout > 5000) {

Serial.println(">>> Client Timeout !");

client.stop();

return;

}

}

// Read all the lines of the reply from server and print them to Serial

while(client.available()){

String line = client.readStringUntil('\r');

Serial.print(line);

}

Serial.println();

Serial.println("closing connection");

}