**OFF-GRID POWER SYSTEM**

**Design**

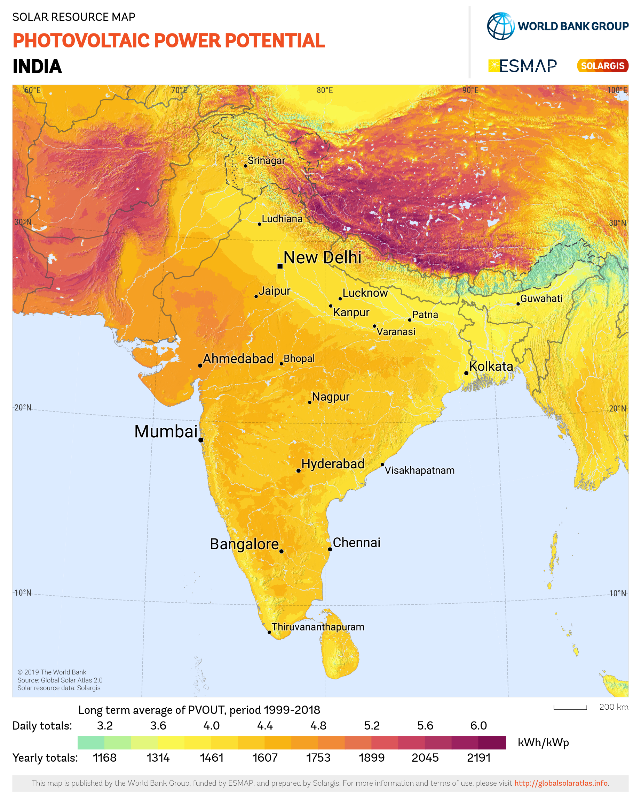
by Durgesh Kolte

**Components Price:**

|  |  |
| --- | --- |
| ESP8266 | 425rs |
| BMP180 air pressure | 220rs |
| DHT22 Humidity Moisture | 150rs |
| LM35 Temperature Sensor | 200rs |
| Photoresistor | 20rs |
| LM317 voltage regulator | 20rs |

**\*Solar Panel\***

Location: **Pune**



Yearly Solar Power= 1607 kilo watt (per sq. meter)

Daily Solar Power = 4.4 kilo watt (per sq. meter)

Harvest per hour = 4.4 / 24 = 183 watt (per sq. meter) [9am – 5pm]

Solar Panel Efficiency = 10% = 18 watt

Power required by ESP8266 = 3.6v \* 0.1A = 0.36 watt

## **LITHIUM-ION 18650 RECHARGEABLE CELL 3.7V 1500MAH (2C)**

Power Rating of Battery = 4.2v \* 1.5A = 6.3watt

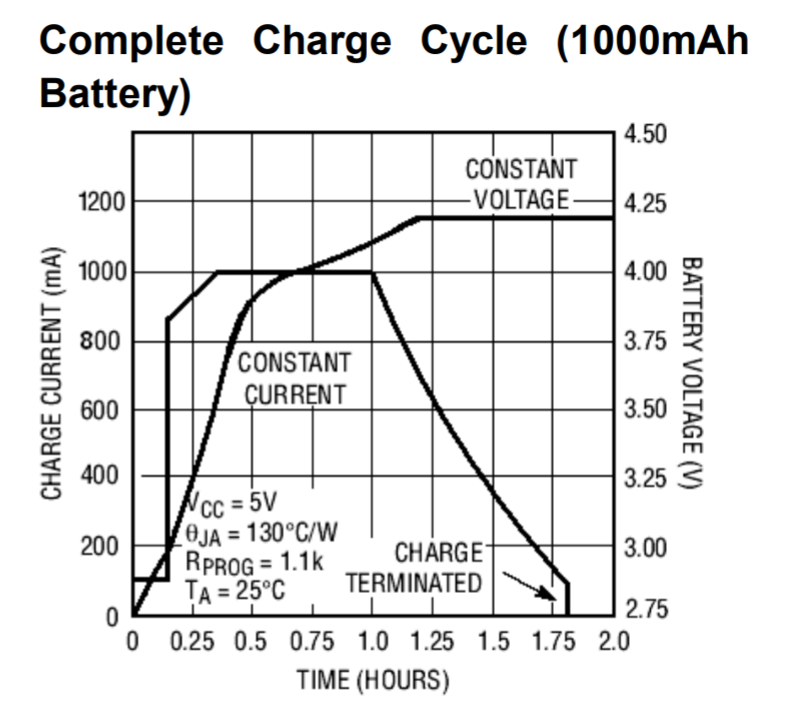
# **Electronicspices- 10 Watt - 6 Volt Solar Panel**

* Efficiency 10.03%
* Low - Light Performance
* High Load Resistant
* solar For HOME
* Dimension-290x190x17 mm

Current = Power / Voltage = 10 watt / 6 V = 1.67Amp

[5:57 PM, 6/24/2020] Durgesh Kolte:

**\*TP4056 Li-ion charge Protection\***



[6/24, 6:12 PM] Durgesh Kolte**:**

**\*Battery Discharge Calculations: \***

**1500mAh battery.**

**Current Aspect:**

Continuous Discharge (without sleep mode)

(mAh / Current drawn) = Runtime

(1500 / 100) = 15hrs

Runtime with sleep mode (every 5 mins):

Wake current drawn: 100mA

Sleep: 0.3mA

Average current drawn = (100\*8 + 0.3\*290) / 300 = 2.96mA

Runtime = (1500 / 2.96) = 506.75hrs = 21.1 days

Runtime with sleep mode (every 2 min) = 9 days (6.94mA)

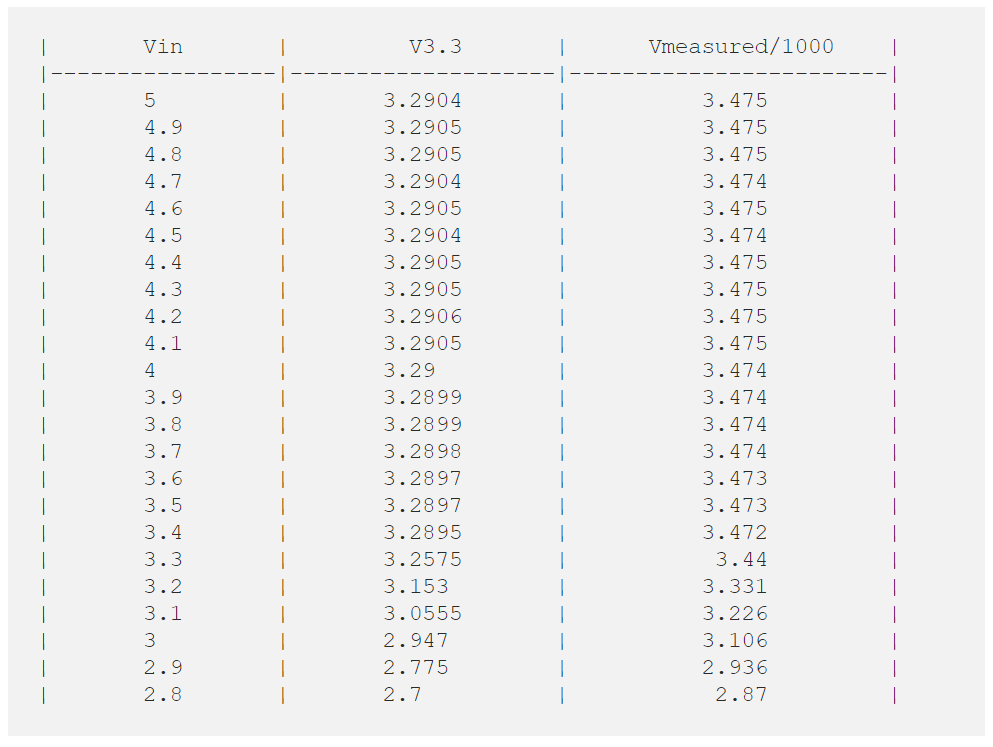
**Voltage Aspect:**

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.

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();



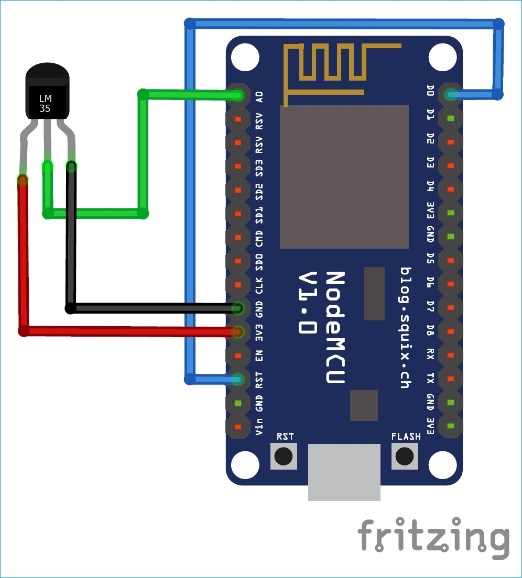


First of all, we see there’s an offset of about 0.18V between the Vmeasured and the actual V3.3. One can leave it like this or can compensate in the software.

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.

Pin Diagram for Sleep Mode:

**Command for Deep Sleep Mode NodeMCU:**

ESP.deepSleep(sleepTimeInSeconds \* 1000000);

**(Also available for ThinkSpeak)**

However, there’s a catch: to wake up the ESP8266 one has to connect the RST pin to GPIO16 (WAKE). On the ESP8266 Thing Dev you do this by closing the SJ2 jumper. On other boards, one has to use a wire to connect the RST and GPIO16 pins.

Another aspect is that the ESP8266 will lose everything in its memory, and it will run the code just as it does when it’s powered on for the first time.

Of course, one can save some context variables in the EEPROM. But, if a node wakes up every five minutes, this will result in having 288 writes to the EEPROM per day. The AT25SF041 used by ESP8266 Thing Dev is rated for 100,000 Program/Erase Cycles. That means that the EEPROM will wear in less than one year.

**Types of Sleep Modes in ESP8266**

Esp8266 module operates in the following modes:

1. **Active mode:** In this mode, whole chip is powered on and chip can receive, transmit the data. Obviously, this is the most power consuming mode.
2. **Modem-sleep mode:** In this mode, the CPU is operational and the Wi-Fi radios are disabled. This mode can be used in the applications which requires the CPU to be working, as in [PWM](https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation). It makes the Wi-Fi Modem circuit to turn off while connected with the Wi-Fi AP (Access Point) with no data transmission to optimize power consumption.
3. **Light-sleep mode:** In this mode, the CPU and all peripherals are paused. Any wake-up such as external interrupts will wake up the chip. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power consumption.
4. **Deep-sleep mode:** In this mode only the [RTC](https://circuitdigest.com/microcontroller-projects/automatic-pet-feeder-using-arduino) (Real Time Clock) is functional and all other components of the chip are powered off. This mode is useful where the data is transmitted after a long time intervals.

**Steps to use the Deep-sleep mode:**

1. Connect the module with the Wi-Fi AP
2. Perform a task like reading a sensor value, publishing an MQTT message, etc.
3. Sleep for a defined number of microseconds
4. Repeat the above process again

Sleep time is defined in microseconds. According to the ESP8266 SDK, you can only sleep for 4,294,967,295 µs which is about ~71 minutes.

**Setting up the ESP8266 Module:**

Connect the RST pin of ESP8266 with the GPIO 16 i.e. D0 pin**. GPIO 16 is important pin which has a WAKE feature**.

Connect the **LM35 temperature sensor** with the A0 pin of NodeMCU.

When the ESP module has HIGH on RST pin, it is in running state. As soon as it receives LOW signal on RST pin, the ESP restarts.

Set timer using deep sleep mode, once the timer ends then the D0 pin sends the LOW signal to RST pin and the module will wake up by restarting it.

Now, the hardware is ready and well configured. The temperature readings will be sent on the Thingspeak server. For this, make an account on [thingspeak.com](http://thingspeak.com/) and create a channel by going through the below steps.

Now, copy the Write API key. Which will be used in the ESP code.