Curva descarga batería

Cuando compramos una batería o miramos la ficha técnica de una batería, está siempre suele venir acompañado de una letra C y un número (C1, C10, C100…) Las tasas de carga y descarga de una batería se rigen por las tasas C. La capacidad de una batería generalmente se califica a 1C, lo que significa que una batería completamente cargada con un valor nominal de 1Ah debe proporcionar 1A durante una hora. Entonces, si esta misma batería que se descarga a 0,5C entonces debe proporcionar 500 mA durante dos horas, y si la descarga fuera a 2C entonces la batería debería entregar 2A durante 30 minutos. Es importante remarcar que estos valores que entrega el fabricante para una carga o descarga son en base a mediciones en condiciones ideales, es decir, por ejemplo, cuando la batería está sometida a temperaturas ideales de 25ºC.

Es importante observar cómo afecta la temperatura a la vida de las baterías. A continuación, puede verse una tabla estándar de ejemplo donde se muestra la duración de vida de baterías AGM/GEL en función de la temperatura de funcionamiento:

En cuanto al ritmo de descarga, una tasa C de 1C también se conoce como descarga de una hora; 0.5C o C/2 es una descarga de dos horas y 0.2C o C/5 es una descarga de 5 horas. Algunas baterías de alto rendimiento pueden cargarse y descargarse por encima de 1 C con una tensión moderada. A continuación, se muestran los tiempos típicos en función de varias tasas C:

Obtuvimos las hojas de datos de la batería FullTotal 14500, que se presentan en las siguientes imágenes:

2.SPECIFICATION

No.	Item	Characteristics	Remarks
1	Nominal Capacity	Minimum: 720mAh Typical: 800mAh	Standard discharge (0.2C) after Standard charge
2	Nominal Voltage	3.7V	—
3	Charging Cut-off Voltage	4.2V	—
4	Discharge Cut-off Voltage	3.0V	—
5	Standard Charge	Constant Current 0.5C Constant Voltage 4.2V 0.01 C cut-off	Charge Time : Approx 4.0h
6	Maximum Constant Charging Current	800mA	—
7	Standard Discharge	Discharge at 0.2 C to 3.0V	—
8	Maximum Continuous Discharging Current	1200mA	—
9	Operating Temperature	Charge 0~45℃ Discharge -20~60℃	—
10	Storage Temperature	-20~45℃ for 1Month -10~35℃ for 6Months	—
11	Storage Voltage	3.7-3.85V	—
12	Environmental request	RoHS	If the materials of the product and packaging accord with RoHS standard, there will be a RoHS Id on the box.

3. Dimensions

Please refer the drawing in appendix.

4. Appearance

No scratches, dirt, defect, leakage of electrolyte or gassing should be observed as a new product.

5. Standard Testing Environment

Temperature : 25±2℃

Relative humidity : 65±20% (unless specially requested)

6. Characteristics

6.1 Electrochemical performance characteristics

No.	Item	Testing Method	Requirements
1	Fully Charged State	CCCV or Constant current charge to 4.2V @0.5C follow by a constant voltage holding at 4.2V until current drops below 8±2mA.	—
2	Rated Capacity	0.5c CCCV 0.01c at 4.2V (per 6.1.1) at room temp. (20±5C), rest for 1-2 hrs then discharge at a constant current of 0.2C to 3.0V, testing will be terminated by either 5 cycles or any one discharge time exceeds 5 hrs	≥720mAh
3	Cycle Life @25℃	Discharge to 3.0V @0.2C, then 0.5c CCCV 0.01C charge to 4.2V, rest for 10 min. discharge @ 0.2C to 3.0V and rest for 10 min. Continue the charge/discharge cycles until discharge capacity lower than 80% of rated capacity.	Cycle life ≥300
4	Internal Impedance	Internal impedance is measured on a 50% charged battery at 1KHz AC at ambient temperature（20±2）℃	—
5	Capacity Retention	Fully charge cells per 6.1.1, store them at (20±2)℃ for 28 days, then discharge the cells to 3.0V at 0.2C.	Discharge Capacity≥640mAh
6	High Temperature Characteristics	Fully charge cells per 6.1.1, store them at (55±2)℃ for 2 hours, then discharge the cells to 3.0V at 0.2C.	Discharge Capacity≥640mAh
7	Low Temperature Characteristics	Fully charge cells per 6.1.1, store them at (-10±2)℃ for 16~24 hours, then discharge the cells to 3.0V at 0.2C.	Discharge Capacity≥480mAh
8	Cell Voltage during Transportation	Check open circuit voltage (OCV) of cells prior to the delivery to customers	≥3.75V

6.2 Safety characteristic

No.	Item	Test Method	Requirements
1	Over charge	Discharge cells to 2.4V at 0.2C, then charge to 4.45V at 3C and rest for 8 hours.	No fire No explosion No leakage
	Overdischarge	Fully charge cells per 6.1.1, then discharge the battery to 3.0V with 0.2CmA at room temperature, connect with external load of 30Ω for 24 hours.	No fire No explosion No leakage
3	Hot Oven Test	Put a fully charged battery in a forced air oven and raise the temperature at 5±2℃/min. to130±2℃ Rest for10 minutes.	No fire No explosion No leakage

De la hoja de datos podemos obtener:

- Capacidad nominal: 0.5c CCCV 0.01c a 4.2V (por 6.1) a temperatura ambiente, descanse durante 1-2 horas y luego descargue a una corriente constante de 0.2c a 3V, la prueba terminará en 5 ciclos o cualquier tiempo de descarga que exceda las 5 horas.

De lo mencionado antes, sabemos que, si la capacidad nominal es 800mAh, si se descarga a 0.2c, significa una descarga a 160mAh durante 5hrs.

https://forococheselectricos.com/2016/02/la-bateria-panasonic-ncr18650b.html
http://www.kuantica-hst.com/ventajas-baterias-linadium/

Consumo ESP32

Analizando la hoja de datos del ESP32, se observa lo siguiente:

5.3 DC Characteristics (3.3 V, 25 °C)

Table 13: DC Characteristics (3.3 V, 25 °C)

Symbol	Parameter		Min	Typ	Max	Unit
C_{IN}	Pin capacitance		-	2	-	pF
V_{IH}	High-level input voltage		0.75×VDD[1]	-	VDD[1]+0.3	V
V_{IL}	Low-level input voltage		–0.3	-	0.25×VDD[1]	V
I_{IH}	High-level input current		-	-	50	nA
I_{IL}	Low-level input current		-	-	50	nA
V_{OH}	High-level output voltage		0.8×VDD[1]	-	-	V
V_{OL}	Low-level output voltage		-	-	0.1×VDD[1]	V
I_{OH}	High-level source current (VDD[1] = 3.3 V, V_{OH} >= 2.64 V, output drive strength set to the maximum)	VDD3P3_CPU power domain [1, 2]	-	40	-	mA
		VDD3P3_RTC power domain [1, 2]	-	40	-	mA
		VDD_SDIO power domain [1, 3]	-	20	-	mA
I_{OL}	Low-level sink current (VDD[1] = 3.3 V, V_{OL} = 0.495 V, output drive strength set to the maximum)		-	28	-	mA
R_{PU}	Resistance of internal pull-up resistor		-	45	-	kΩ
R_{PD}	Resistance of internal pull-down resistor		-	45	-	kΩ
V_{IL_nRST}	Low-level input voltage of CHIP_PU to power off the chip		-	-	0.6	V

Notes:

1. Please see Table IO_MUX for IO's power domain. VDD is the I/O voltage for a particular power domain of pins.
2. For VDD3P3_CPU and VDD3P3_RTC power domain, per-pin current sourced in the same domain is gradually reduced from around 40 mA to around 29 mA, V_{OH}>=2.64 V, as the number of current-source pins increases.
3. For VDD_SDIO power domain, per-pin current sourced in the same domain is gradually reduced from around 30 mA to around 10 mA, V_{OH}>=2.64 V, as the number of current-source pins increases.

Table 6: Power Consumption by Power Modes

Power mode	Description			Power consumption
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 15 for details.
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered on.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA

Power mode	Description			Power consumption
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	-			0.8 mA
Deep-sleep	The ULP co-processor is powered on.			150 μA
	ULP sensor-monitored pattern			100 μA @1% duty
	RTC timer + RTC memory			10 μA

Cuando Wi-Fi está habilitado, el chip cambia entre los modos Activo y Modem-sleep. Por lo tanto, el consumo de energía cambia en consecuencia.

5.5 RF Power-Consumption Specifications

The power consumption measurements are taken with a 3.3 V supply at 25 °C of ambient temperature at the RF port. All transmitters' measurements are based on a 50% duty cycle.

Table 15: RF Power-Consumption Specifications

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

Por otra parte, de la página (https://www.luisllamas.es/comparativa-esp8266-esp32/) donde se realiza una comparación entre el ESP32 y ESP8266, se describe al ESP32 con un consumo promedio de 80mA, con picos de 225mA (máximo).

Consumo de energía del módulo ESP32 WROOM

Algunos valores de consumo de energía determinados del chip ESP32:

Modo ESP32	Consumo
“Deepsleep”	7 µA
“Lightsleep”	1 mA
Normal (240 MHz)	50 mA
Reloj del procesador reducido (3 MHz)	3,8 mA
Funcionamiento WiFi	80-180 mA

Medimos estos valores con el módulo ESP32-WROOM, pero también los alcanzamos con nuestra tarjeta ECO Power en funcionamiento con batería. Sin embargo, la mayoría de las tarjetas ESP32 no se acercan a este bajo consumo de energía. Por lo general, las tarjetas ESP32 necesitan alrededor de 20 mA a pesar del modo “deepsleep”, es decir, ¡consumen más de dos mil veces más! Los factores importantes para el consumo de energía son los circuitos adicionales en la tarjeta, la implementación de la fuente de alimentación USB y la implementación de la operación por batería. Con las tarjetas ESP32 regulares, el consumo de energía no se puede reducir más, por lo tanto tuvimos que desarrollar algo propio. Esto es lo que hemos hecho con la tarjeta ECO Power.

A continuación, se coloca el link de una página argentina donde se venden distintos tipos de ESP32 https://www.monarcaelectronica.com.ar/search/?q=esp32

Regulador LDO

En la página https://ar.mouser.com/Semiconductors/Power-Management-ICs/Voltage-Regulators-Voltage-Controllers/LDO-Voltage-Regulators/_/N-5cgac?keyword=S33&No=25 , se encontraron algunos reguladores similares al MCP1825. Las diferencias se destacan en el siguiente cuadro:

Modelo	Capacidad de corriente de salida	Rango de tensión de entrada	Low Dropout Voltaje
MCP1825	500mA	2.1V a 6.0V	210 mV Typical a 500 mA
MCP1826	1000mA	2.3V a 6.0V	250 mV Typical a 1000 mA
MCP1827	1.5A	2.3V a 6.0V	330 mV Typical a 1.5A
MCP1755	300mA	3.6V a 16.0V	300 mV typical a 300 mA
TLV75533PDQNR	500mA	1.45V a 5.5V	0.215@500mA|0.238@500mA

Link que describe al MCP1755 https://www.diarioelectronicohoy.com/reguladores-de-tension-ldo/

En particular, nos interesa conocer sobre el MCP1825, para ello investigamos un poco en cuanto a la eficiencia del mismo. Como la hoja de datos no nos da información directa sobre la eficiencia, utilizamos la curva de corriente a

tierra, como sugiere el siguiente articulo; https://www.digikey.com/es/articles/use-adjustable-low-leakage-ldos-to-extend-battery-life-in-wearable-designs :

Cálculo de eficiencia de un LDO

Lo que determina la eficiencia de un LDO es su corriente de tierra (I_{GND}) y los voltajes de entrada y salida (V_{IN} y V_{OUT}). La fórmula para calcular la eficiencia es:

$$Eficiencia = I_{OUT}/ (I_{OUT} + I_{GND}) \times V_{OUT}/V_{IN} \times 100\%$$

IGND es la corriente requerida para operar los circuitos internos del LDO (que es la diferencia entre las corrientes de entrada y salida). Una parte clave de esto es la corriente de reposo del LDO (IQ), que alimenta los circuitos internos del LDO cuando la corriente de carga externa es cercana a cero. Incluye la corriente de funcionamiento del amplificador de error, el divisor de voltaje de salida y los circuitos de detección de sobre corriente y temperatura.

Debido a su impacto en la eficiencia, IGND e IQ son especificaciones clave de la hoja de datos de un LDO. Por ejemplo, un producto adecuado para alimentar un dispositivo portátil, como el LDOMCP1811BT-028/OT de Microchip, tiene cifras de IGND = 180 microamperios (µA) (a IOUT = 300 miliamperios [mA]) y IQ = 250 nano amperios (nA). La IQ (y, por lo tanto, la IGND) aumenta a medida que aumenta la IOUT.

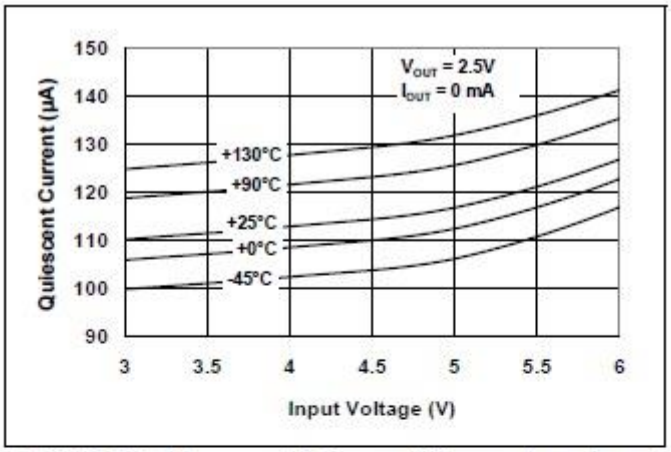


FIGURE 2-11: *Quiescent Current vs. Input Voltage.*

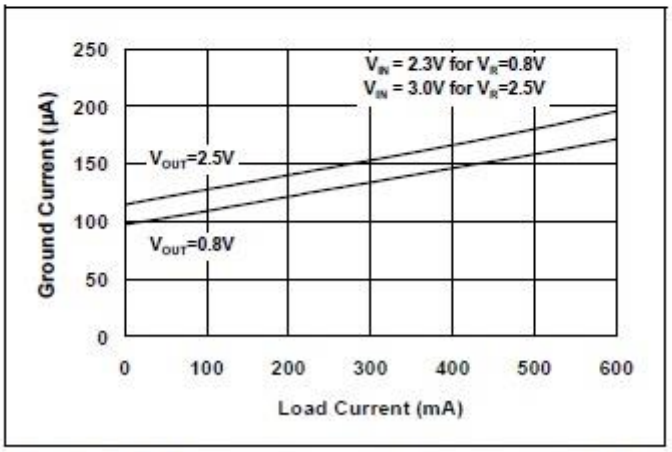
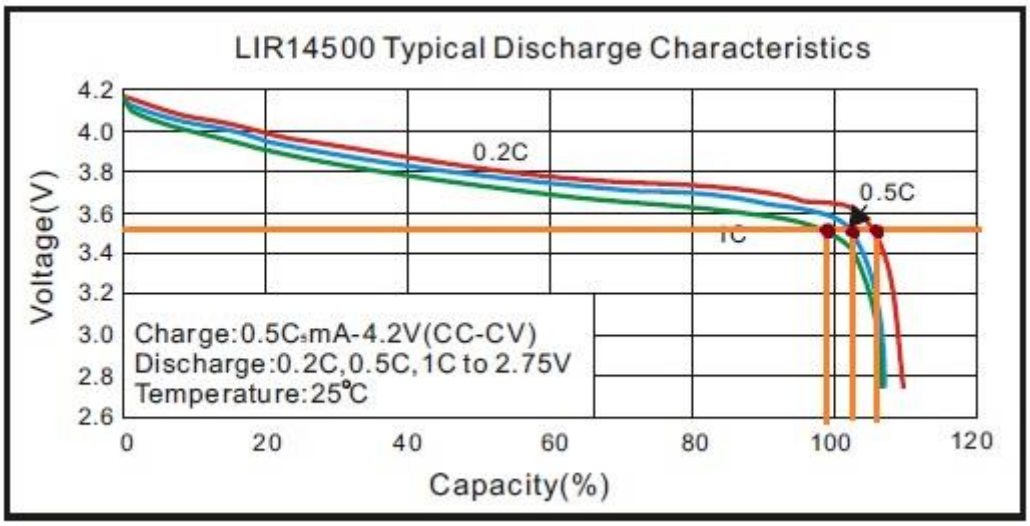


FIGURE 2-12: *Ground Current vs. Load Current.*

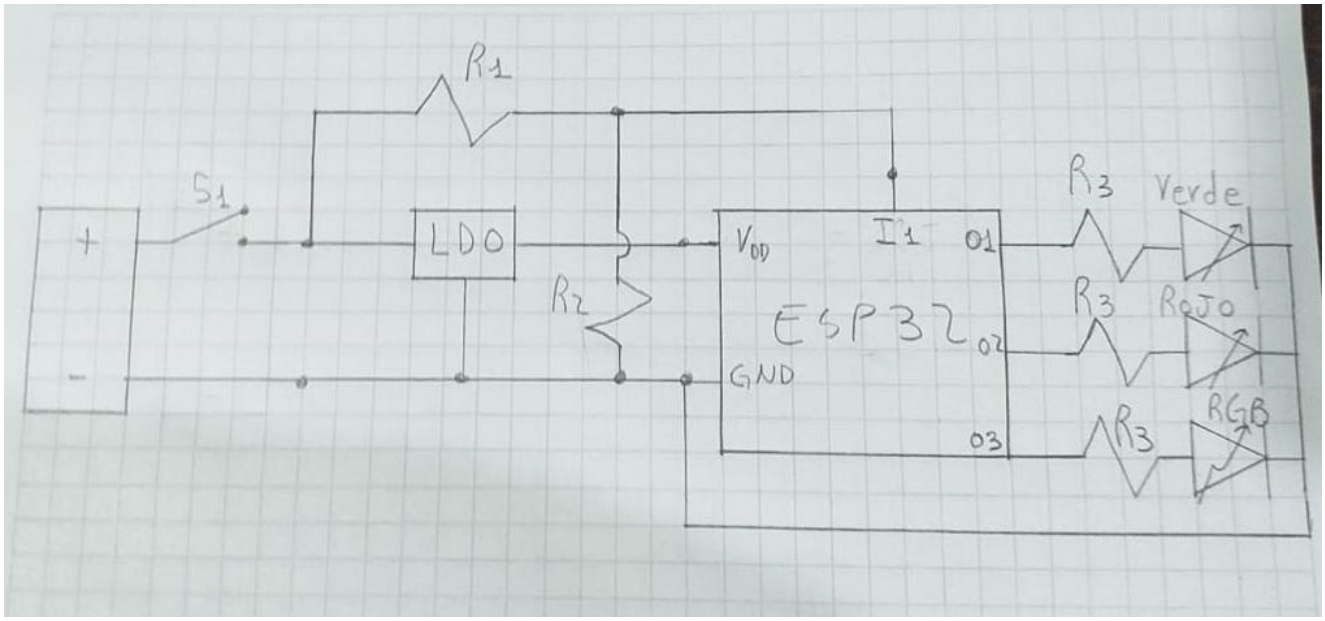
Para saber bien cual es la corriente que entra en el LDO (la que suministra la batería), debemos buscar en las anteriores curvas cual es la corriente de tierra, para la corriente de salida que deseada. Para obtener la duración estimada de la batería, nos fijamos la hoja de datos de la batería LIR14500, la cual es similar a la FullTotal. Se observa que la tensión de salida del regulador será 3.3V, sumándole el voltaje de low-dropout máximo (210mV aproximadamente) nos da una tensión de 3.51V a la entrada para un consumo de corriente de 500mA.

En las curvas de descarga podemos ver que se llega a dicha tensión utilizando aproximadamente el 100% de la capacidad, en los casos que la descarga se realiza entre 0.2C y 0.5C, la capacidad es un poco mayor.



De esta manera podemos realizar una estimación de cuanto durara la batería para una determinada corriente.

- Si la corriente promedio que consume el ESP32 fuera 80mA, tendríamos una corriente I_{GND} igual a 0.15mA. Utilizando la planilla de calculo que creamos para obtener la eficiencia y la duración de la batería, tenemos que la eficiencia es aproximadamente del 89%, mientras que la duración nos da aproximadamente 9hs 30min, para una capacidad nominal de 750mAh.



ESP32: Diseño PCB, confección y mediciones.

- ## ***Diseño y confección PCB***

Para realizar el montaje del circuito, al integrar al Node MCU ESP32 ensamblado en formato “Shield”, se procedió en principio por armar el diseño de la librería del mismo, para utilizarla luego en Altium Designer. Teniendo en cuenta que la distancia entre pines del mismo grupo es 2.5mm, y la distancia entre grupos es 25.5mm, se confecciono la librería que se observa en la **Figura 1.**

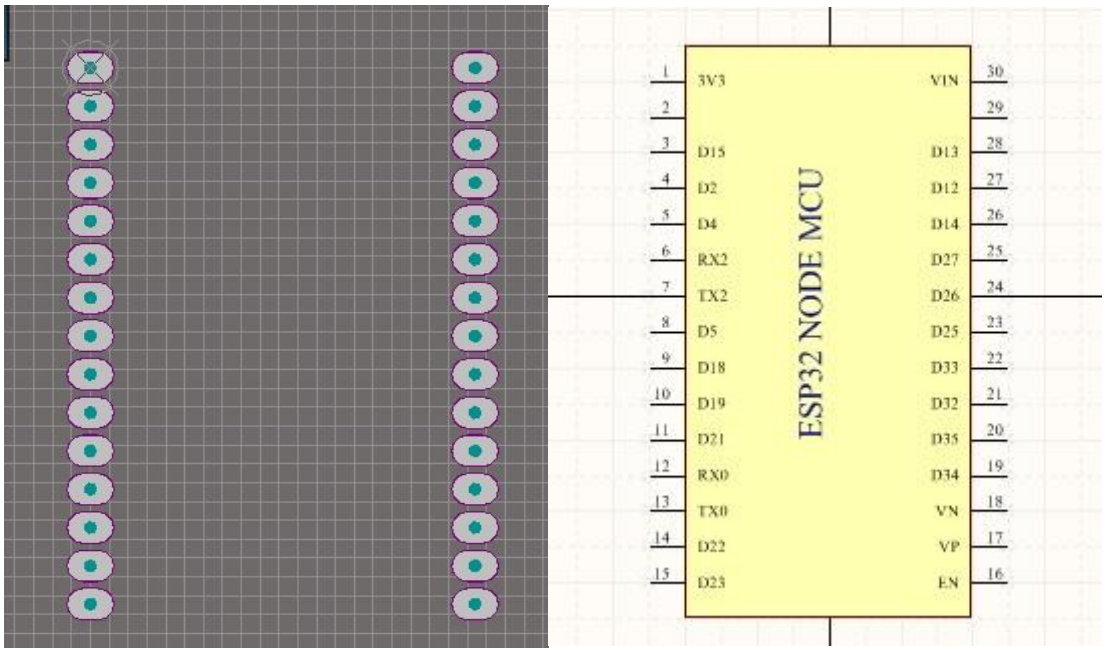


Fig. 1: Librería creada para el Node MCU ESP32

Creada la huella para poder realizar el PCB, surgió el interés de encontrar un circuito de enclavamiento que nos permita encender y apagar el microcontrolador mediante un pulsador, y así mismo, que el microcontrolador pueda apagar el circuito independientemente del pulsador. El circuito que se encontró, que mejor se adapta a estas necesidades es el que se muestra en la **Figura 2.**

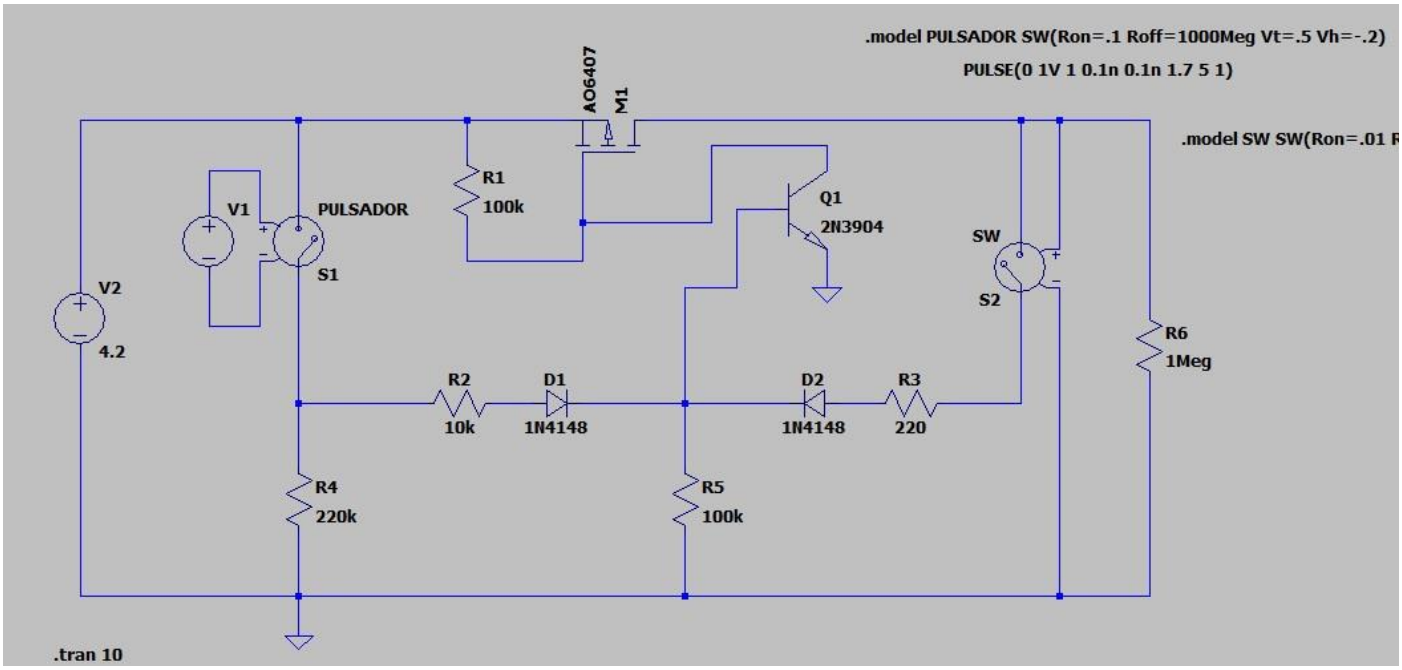


Fig. 2: Circuito de enclavamiento para encender/apagar el microcontrolador

El funcionamiento del circuito se caracteriza por el nodo que une los diodos D1 y D2. Cuando se energiza el circuito, el MOSFET se encuentra apagado, y la tensión en dicho nodo es nula dado que no hay corriente a través de R5. En el momento que se presiona el pulsador, se establece una corriente a través de D1 y R5, provocando una tensión en la base de Q1, sucesos que derivan en el encendido del MOSFET. Para que el circuito se mantenga encendido en el momento que se suelta el pulsador, se debe mantener la tensión en R5, por tal motivo, la rama que contiene a D2 y R3 se debe conectar a un pin GPIO del microcontrolador, el cual se pondrá en alto ni bien se energiza el mismo.

De esta manera no solo se logra encender el microcontrolador con un solo pulsador, sino que también se puede apagar el mismo poniendo el GPIO correspondiente en bajo. Esto es muy útil desde el punto de vista energético, ya que se podría apagar el microcontrolador, por ejemplo, si se detecta que el mismo lleva varios minutos sin uso.

También interesa poder apagar el microcontrolador al presionar el pulsador, esto hace que se deban agregar componentes al circuito. Para lograrlo, se pensó en agregar un divisor resistivo entre un pin GPIO del microcontrolador y tierra, cuyo punto medio debe ir conectado al nodo que une al pulsador, con R4 y R2 como se aprecia en la **Figura 3**, de esta manera se lograría que, al momento de accionar el pulsador, el microcontrolador lo detecte y luego lo apague.

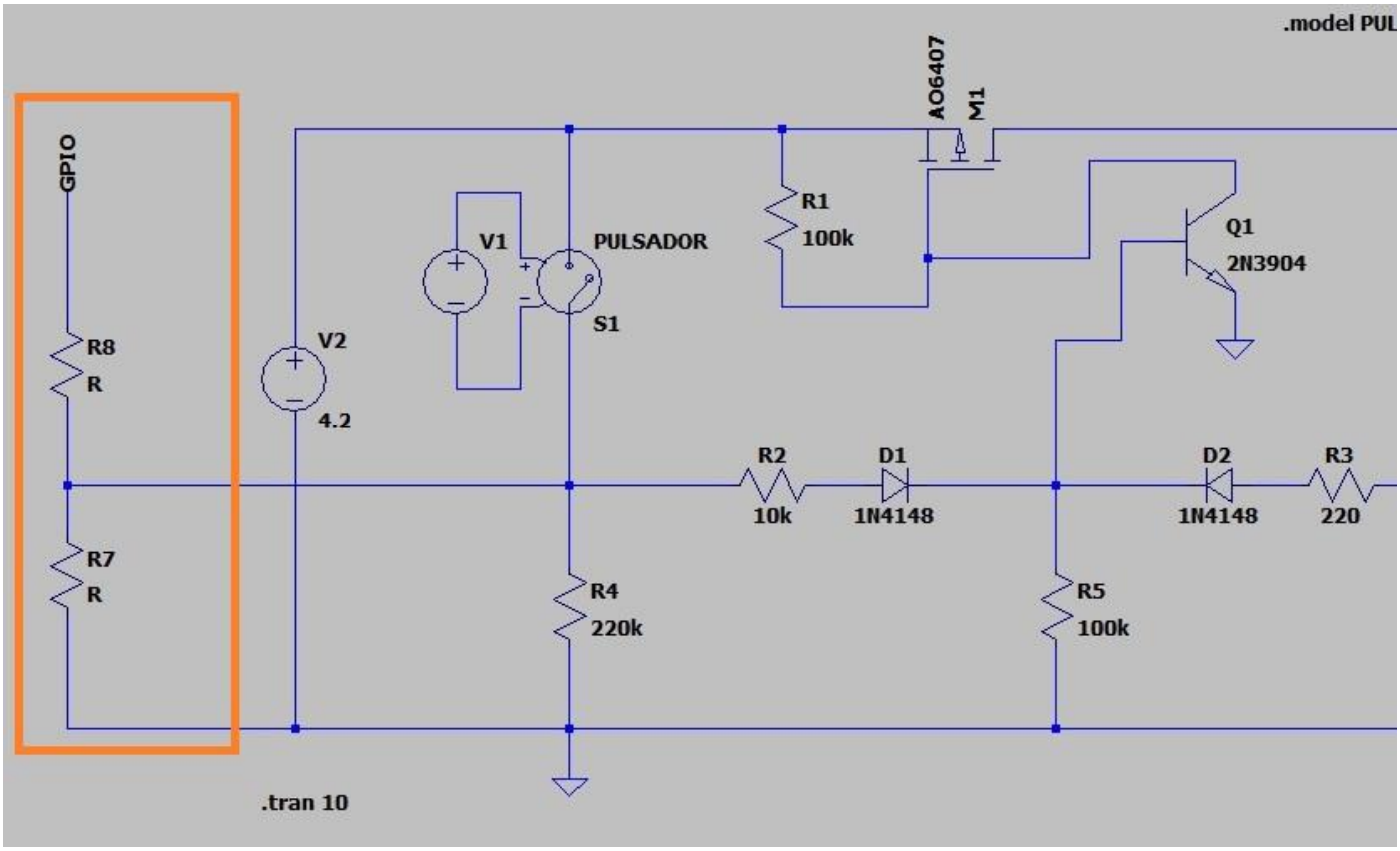


Fig. 3: Divisor resistivo para apagar el microcontrolador, a través del pulsador

Además del circuito de encendido, y el Node MCU ESP32, el circuito también está conformado por un Led RGB de 4 pines conectado al microcontrolador para indicar el estado de la batería, y de un regulador de tensión LDO de 3.3V. Como hay 2 reguladores en investigación, el MCP1825 y KF33BD, el PCB tendrá la opción de colocar ambos. El circuito final es el que se observa en la **Figura 4.**

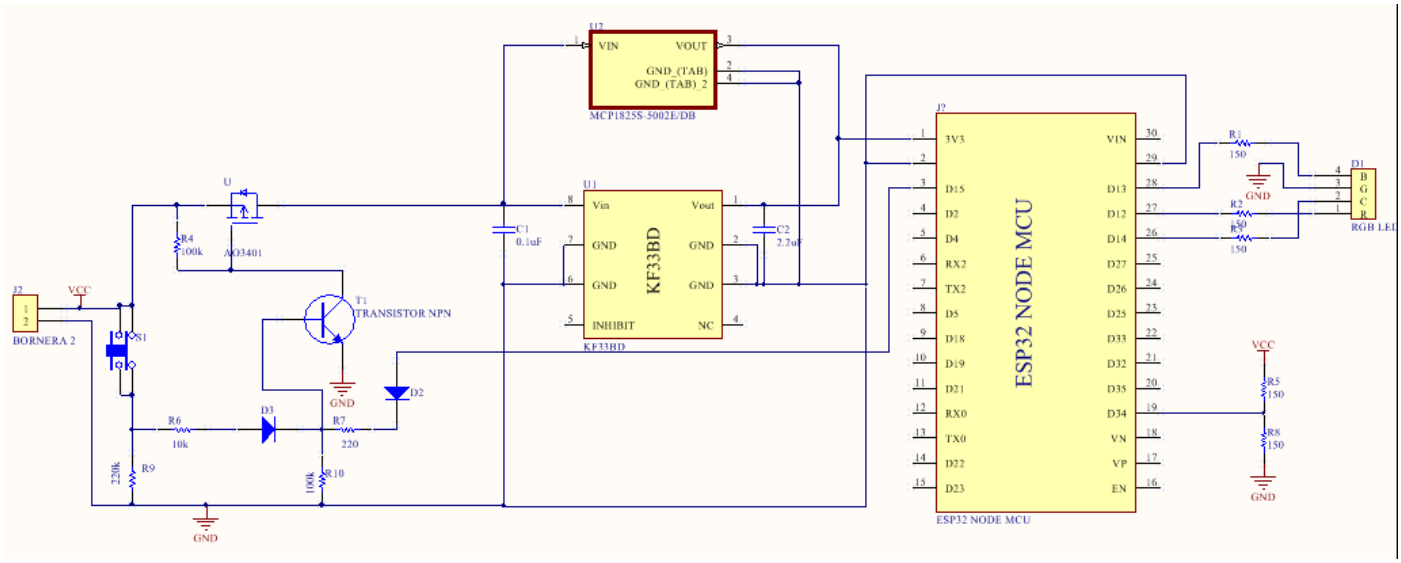


Fig. 4: Diseño esquemático del circuito final con Altium Designer.

Finalmente, se confecciono el PCB teniendo como principal objetivo optimizar las dimensiones del mismo. El resultado fue una placa de aproximadamente 60x60mm con la distribución de componentes que se indica en la **Figura 5-a, b, c y d,** la cual además de contener los pad's propios de los componentes, cuenta con algunos extras, por si se requiere utilizar GPIO's adicionales, o componentes.

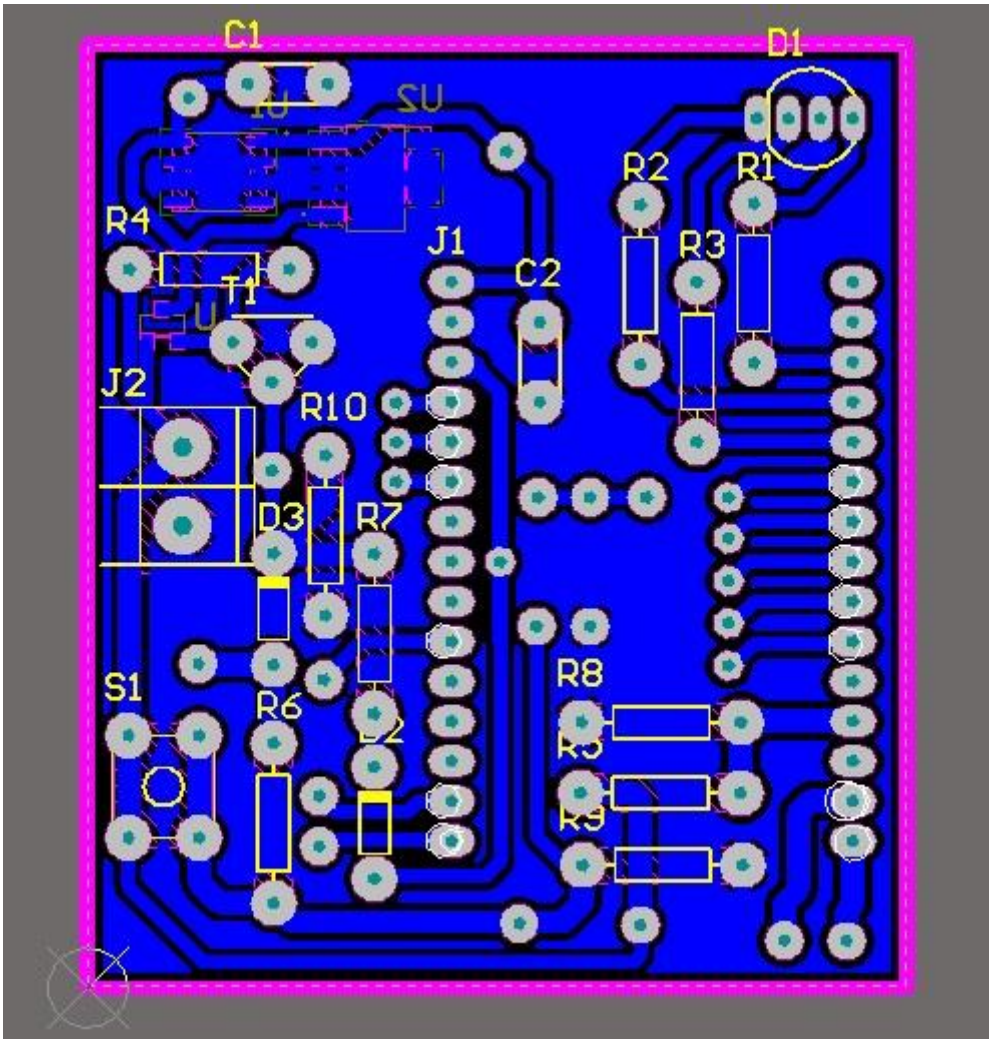


Fig. 5-a: Diseño PCB con Altium Designer.

Fig. 5-b: Diseño PCB transferido a la placa de cobre

.

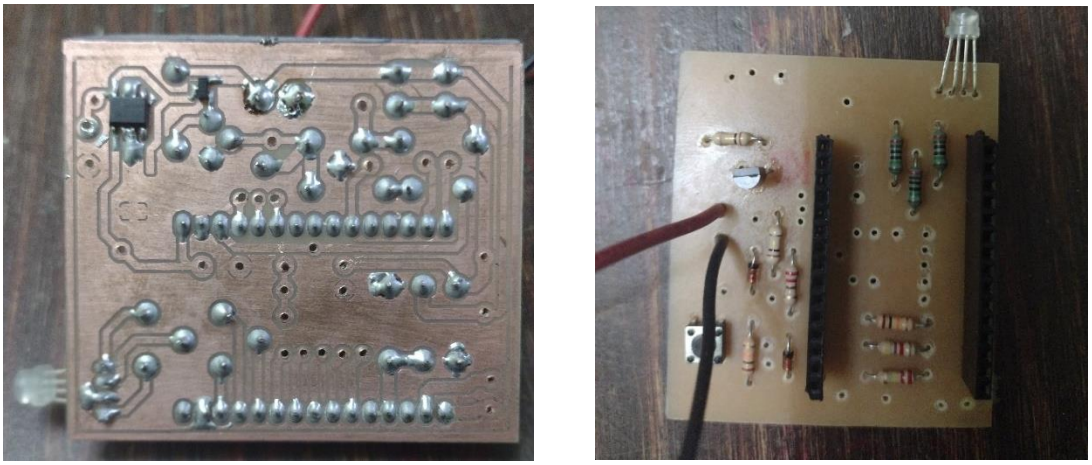


Fig. 5-c: Diseño PCB finalizado – Versión Final.

- ***Medidor de tensión en las baterías***

Luego de conseguir las baterías de Ion – Litio 14500 marca “Full Total”, se procedió por realizar un script en el IDE Arduino para medir el nivel de tensión en las baterías, a partir de la lectura en el pin D34 del Node MCU. De esta forma se podría, por ejemplo, programar el ESP32 para que se apague automáticamente cuando la tensión en la batería llegue a un nivel crítico. Como el nivel de tensión en las baterías toma un valor de 4.2V cuando está completamente cargada, y las entradas/salidas del ESP32 trabajan a una tensión de 3.3V, fue necesario utilizar un divisor de tensión para adaptar los niveles.

Se utilizaron 2 resistencias, una de 10kOhm y una de 4.7kOhm como puede apreciarse en la **Figura 6**, logrando una relación entre la tensión de la batería y la entrada al ESP32 aproximadamente de 0.6803. Es decir, cuando la tensión en la batería sea 4.2 V, en el GPIO D34 se tendrá 2.8571 V.

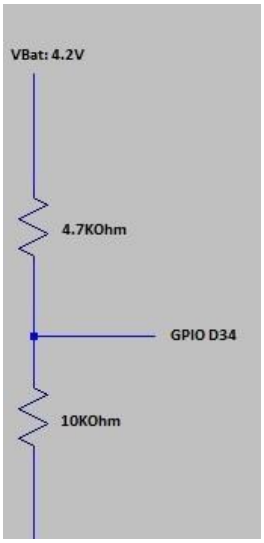


Fig. 6: Diseño PCB finalizado – Versión Final.

El ESP32 cuenta con 18 canales ADC de 12bits, de esta manera cuando la tensión en el GPIO sea 3.3 V, la salida del CH utilizado del ADC, tendrá una salida de 4095. Aplicando estas reglas al divisor utilizado en el circuito, tendremos:

Tensión Batería	Tensión pin D34	Salida ADC
4.2V	2.8571V	3545
3.7V	2.5169V	3123
3.4V	2.3129V	2870
3.3V	2.2448V	2786

Finalmente, el código utilizado para describir el nivel de la tensión en la batería es el siguiente:

```
#include "WiFi.h"
const char* ssid = "AQUI VA EL NOMBRE DE LA RED A CONECTARSE";
const char* password = "AQUI VA LA CONTRASEÑA DE LA RED A CONECTARSE";
WiFiServer server (80); //CREAMOS UN OBJETO PARA EL WEB SERVER

#define LEDWIFI  2     // LED indicador de conexión WIFI en la placa del ESP32.
#define LEDBATERIA1 12 // LED VERDE indicador de batería con carga mayor a %50 en el terminal D12.
#define LEDBATERIA2 13 // LED AZUL indicador de batería con carga menor a %50 en el terminal D13.
#define LEDBATERIA3 14 // LED ROJO indicador de batería con carga menor a %10 en el terminal D14.
#define TENSIONBATERIA 34 // Lee la tensión de la batería para determinar el estado de carga.
#define ENCENDER 15 // GPIO para mantener encendido el ESP32 al presionar el pulsador

int tiempo_inicial =0; //Variable para almacenar el tiempo desde que se encendió el ESP32.
float tension_bateria=0; //Variable para almacenar el valor de tensión de la batería, ya convertido a [V].
int tension_bateria_digital=0; //Variable para almacenar el valor de tensión de la batería en el ADC.
float tension_led_porcentaje=0; //Variable para almacenar el valor de tensión del batería convertido a [%].
String estadobateria = ""; //Variable para actualizar el Web Server.

void setup () {
  Serial.begin(9600);
//INICIALIZAMOS LOS GPIO
  pinMode (LEDWIFI, OUTPUT);
  pinMode (LEDBATERIA1, OUTPUT);
  pinMode (LEDBATERIA2, OUTPUT);
  pinMode (LEDBATERIA3, OUTPUT);
  pinMode (ENCENDER, OUTPUT);

  digitalWrite (ENCENDER, HIGH);
```

```
// Conectamos a la red WIFI.
  Serial.println();
  Serial.print("Conectando con ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
//VERIFICAMOS QUE LA CONEXION SEA EXITOSA.
  While (WiFi.status() != WL_CONNECTED) {
    digitalWrite (LEDWIFI, LOW);
    Serial.print(“.”);
    delay (500);
    digitalWrite (LEDWIFI, HIGH);
    delay (500);
  }
  Serial.println(“Conectado con WIFI.”);
  digitalWrite (LEDWIFI, LOW);

  // INICIAMOS EL WEB SERVER.
  Server.begin();
  Serial.println(“Servidor WEB iniciado.”);

  // Esta es la IP para conectarse al servidor WEB.
  Serial.print("Esta es la IP para conectar: ");
  Serial.print("http://");
  Serial.println(WiFi.localIP());
  tiempo_inicial= millis (); //ALMACENAMOS EL TIEMPO INICIAL DEL LOOP.
}

void loop () {
  tension_bateria_digital=analogRead (TENSIONBATERIA);
  tension_bateria=0.15+(tension_bateria_digital * 1.47 * (2.8571/3545)); //Se detecto un offset entre el valor medido y el real de 0.15V, por lo tanto, lo corregimos sumándolo.

  //Ecuación para calcular el porcentaje de carga de la batería
  tension_led_porcentaje= (((tension_bateria – 3.3) / (4.2 – 3.3)) * 100);
```

```
if(tension_led_porcentaje > 50){
  digitalWrite (LEDBATERIA2, LOW);
  digitalWrite (LEDBATERIA3, LOW);
  estadobateria="batcompleta";
  digitalWrite (LEDBATERIA1, HIGH);
}
if (10 < tension_led_porcentaje && tension_led_porcentaje < 50) {
  digitalWrite (LEDBATERIA1, LOW);
  digitalWrite (LEDBATERIA3, LOW);
  estadobateria="batmitad";
  digitalWrite (LEDBATERIA2, HIGH);
}
if (tension_led_porcentaje <= 10) {
  digitalWrite (LEDBATERIA1, LOW);
  digitalWrite (LEDBATERIA2, LOW);
  estadobateria="batagotada";
  digitalWrite (LEDBATERIA3, HIGH);
}

if (tension_bateria >= 3.3) {
Serial.print(tension_bateria);
Serial.println("  V");
}
else {
  digitalWrite (ENCENDER, LOW); //APAGO EL ESP32 CANCELANDO EL ENCLAVAMIENTO.
}

delay (1000);

// Consulta si se ha conectado algún cliente.
WiFiClient client = server.available();
if (!client) {
  return;
}
```

```
// CONSULTA SI SE CONECTO ALGUN CLIENTE A LA WEB, CASO CONTRARIO REGRESA AL PRINCIPIO DEL LOOP.
WiFiClient client = server.available();
if (! client) {
  return;
}



// SI SE CONECTA ALGUN CLIENTE, ESPERA QUE ESTE ENVIE DATOS.
while(!client.available()){ delay(1); }

// Página WEB.
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println(""); //  Importante.
client.println("<!DOCTYPE HTML>");
client.println("<html>");
client.println("<head><meta charset=utf-8></head>");
client.println("<body><center><font face='Arial'>");
client.println("<h1>Servidor web con ESP32.</h1>");
client.println("<h2><font color='#009900'>  </font></h2>");
client.println("<h3>Estado de carga de la bateria</h3>");
client.println("<h2>");

//CAMBIO LA IMAGEN DEL WEB SERVER EN FUNCION DEL PORCENTAJE DE LA BATERIA.
if(estadobateria == "batcompleta"){
  client.println("<img src='https://i.ibb.co/r5gVvjf/batcompleta.jpg'><br>");
}
if(estadobateria == "batmitad"){
  client.println("<img src='https://i.ibb.co/9bwY8c7/batmitad.jpg'><br>");
}
if(estadobateria == "batagotada"){
  client.println("<img src='https://i.ibb.co/f0fd4GZ/batagotada.jpg'><br>");
}
client.print(tension_bateria);
```

```
  client.println("V");
  client.println("</h2>");
  client.println("</font></center></body></html>");

 //CIERRO LA CONEXIÓN CON EL CLIENTE.
  client.flush();
  client.stop();
}
```

En resumen, el código realizar una lectura con ADC en el pin D34 por segundo, luego convierte el valor digital leído, a través de la constante de conversión, a su valor de tensión equivalente. Luego de obtener el valor de tensión en la batería, calcula el porcentaje de la misma en base a 3.3V, es decir, tomando como 100% el valor 4.2V, y 0% el valor 3.3V. En función el porcentaje, cambia la iluminación de un led RGB, como también así la imagen en el Web Server, las cuales son las que se aprecian en la **Figura 7.**

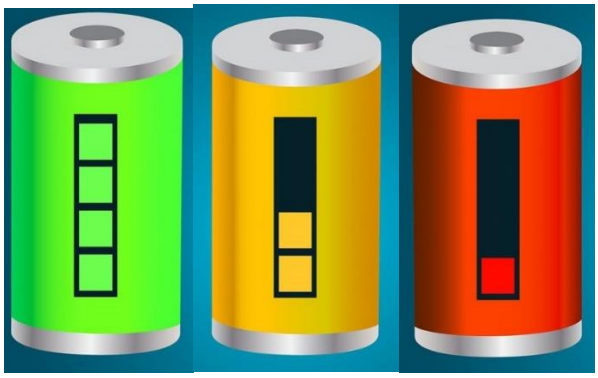


Fig. 7: Las 3 imágenes que se muestran en el Web server.

- ***Ensayos duración batería:***

Para realizar los ensayos sobre la duración de la batería cuando se alimenta el NODE MCU ESP32, dado que la corriente que consume el mismo es variable y depende del envió de datos a través del WIFI, se procedió por conectarle una carga de 30Ohm en la salida del regulador LDO de 3V3 para obtener de esta manera, una corriente constante de aproximadamente, 110mA. Idealmente, con esta corriente se debería obtener una duración aproximada de 7hs dado que la capacidad de las baterías es de 800mAh.

El ESP32, se alimentó del puerto USB de la notebook donde se corre el sketch de Arduino, y se utilizó solamente para medir las tensiones de la batería con una tasa de 1 muestra cada 5 segundos. Estas muestras, se enviaron por puerto serie a MATLAB para poder almacenarlas, y luego realizar una aproximación de la curva de descarga utilizando el siguiente Script:

```
clear all, clc;
%% Limpiar e iniciar el puerto serie.
```

```
delete(instrfind({'Port'}, {'COM3'})); % COM3 es el puerto serie utilizado en nuestro caso
pserial=serial('COM3','BaudRate',9600);
fopen(pserial);

%Declaración de Variables
lectura=0;
numero_muestras=50000; % Ponemos un valor de 50000, a 1 muestra cada 5 segundos equivale a 70 hs de ensayo, lo cual es suficiente.
i=1;
%Títulos Gráficos
figure ('Name','Gráfica de valores obtenidos');
title ('Curva descarga Baterías ');
xlabel ('Muestras a 0.2HZ');
ylabel ('Tensión Batería');
tension_bateria=zeros (1, numero_muestras)';

%Grafico de la curva en tiempo real
while lectura < 100   %%El valor de lectura igual a 100 se utiliza por si se quisiera terminar el muestreo por alguna situación particular, por ejemplo, si la tensión en la batería llega a 3.3V

   ylim ([3.5 4.5]);
   xlim ([0 i+10]);
   lectura=fscanf(pserial,'%f');
   tension_bateria(i)=lectura;
   hold on;
   grid on;
   plot(i,tension_bateria(i),'.');
   drawnow
   i=i+1;
end

tiempo_hs=i/720; %%Almacena el tiempo en hs, cuando se detiene el muestreo, es decir, se envía lectura=100.
tiempo_seg=fscanf(pserial,'%f'); %%
tiempo_seg=tiempo_seg/1000;
```

fclose(pserial);

delete(pserial);

En el script de MATLAB, se pueden observar algunos detalles del código que son para detectar situaciones en particular. Principalmente si se quisiera terminar el relevamiento de la curva de descarga cuando la tensión en las baterías alcance un valor en específico, más precisamente 3.3V. Para realizar esto, en el Sketch de Arduino cuando se mida una tensión inferior a 3.3V, enviará un valor de lectura igual a 100, esto provocara que se detenga el relevo de la curva en MATLAB. A si mismo, se pueden configurar ambos códigos para que, cuando esto suceda, también envié el tiempo desde que se inicio el ESP32 (medido en el sketch de Arduino), y detenga el enclavamiento en el circuito poniendo el pin GPIO15 en bajo, apagando el circuito.

De esta manera, cuando finalice el proceso de relevamiento, los datos medidos quedaran guardados en el vector **"*tension_bateria(i)*"**, para luego poder graficar, o procesar los valores.

Utilizando el esquema que se aprecia en la **Figura 8- a y b**, se realizaron dos ensayos (uno en cada batería), con una duración aproximada de 7hs, en el cual se controlo con el uso de un multímetro la tensión en la salida del regulador, y la corriente sobre las resistencias. En la lectura del multímetro se observa un valor de 3.07V el cual es diferente a los 3.3V teóricos que se deben tener, esto es así en parte, por el error introducido en el multímetro al utilizar una escala de 20V. De todas maneras, no tiene gran importancia la lectura del multímetro ya que se utiliza para controlar que el regulador LDO siga funcionando cuando la batería se va descargando, la lectura importante en base a la tensión, es la que realiza el ESP32 a través del divisor resistivo.

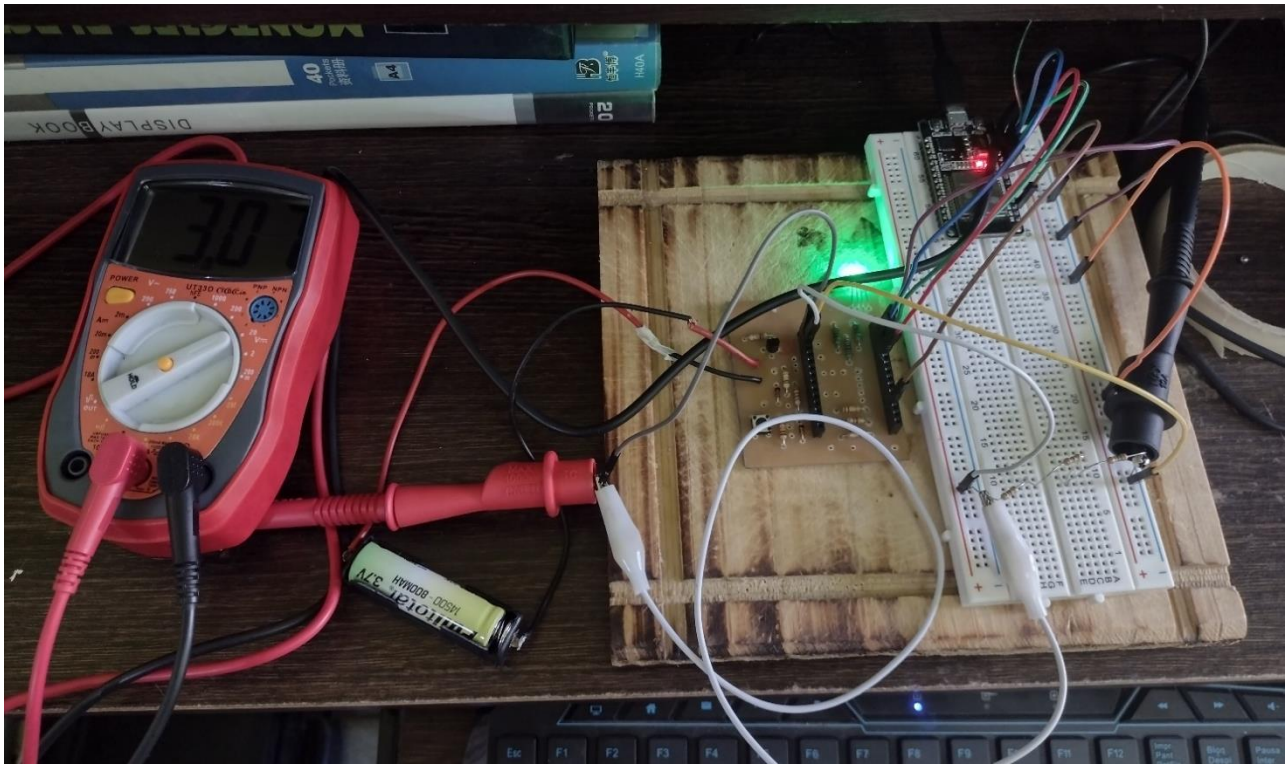


Fig. 8-a: Tensión a la salida del regulador durante el ensayo.

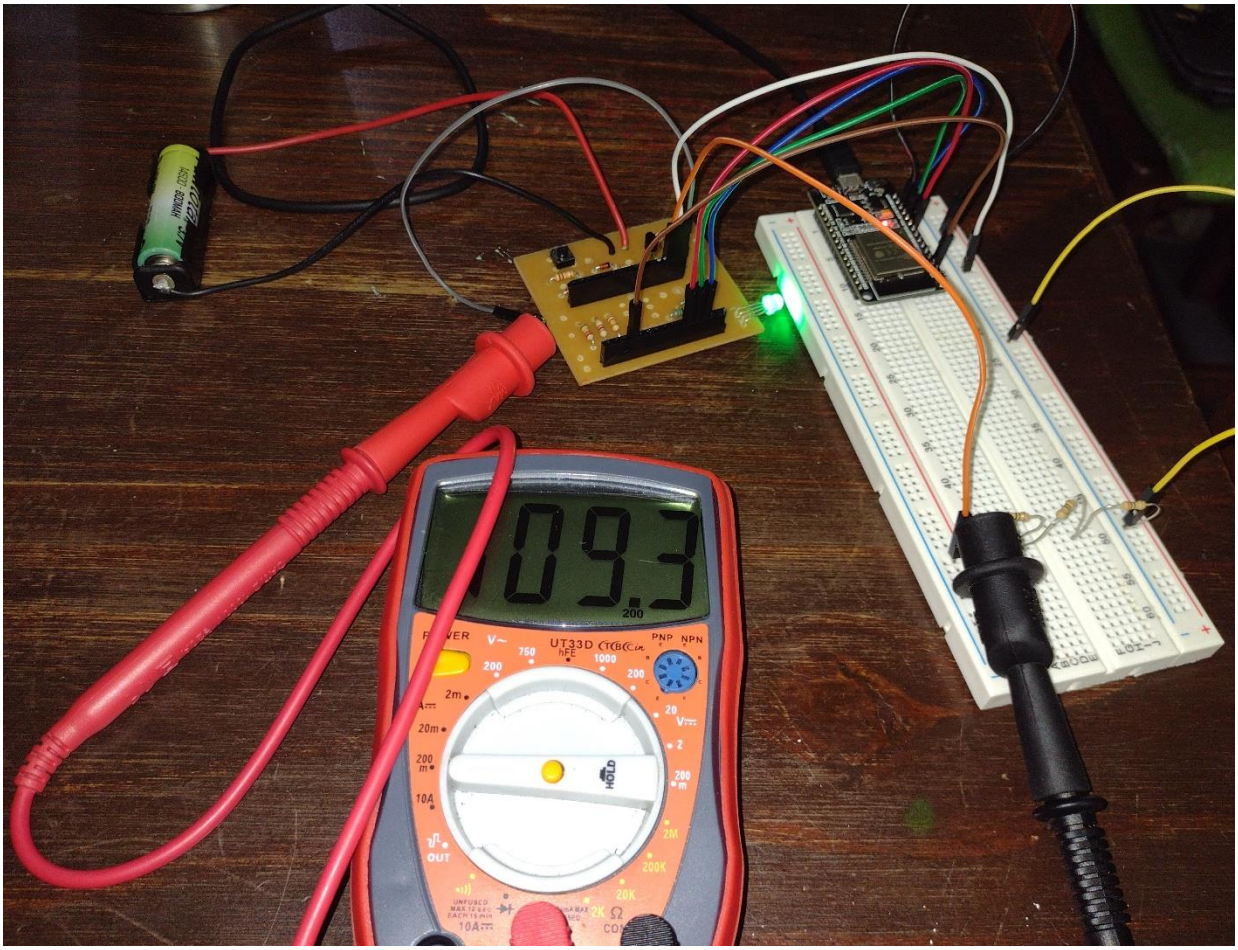


Fig. 8-b: Corriente a la salida del regulador durante el ensayo.

En las **Figuras 9-a y b**, se observan las curvas de descarga en los 2 ensayos realizados, en la misma se ven lugares donde las mediciones se dispersan, esto se debe a que durante el

ensayo se realizaron controles de tensión y corriente, intercambiando el multímetro entre Voltímetro y Amperímetro. Por otra parte, la diferencia entre las curvas en ambas imágenes se asocia en principio, a que en el primero, el ESP32 se alimentó a través del puerto USB de una computadora de escritorio, mientras que en el segundo se alimentó a través del puerto USB de una notebook, observándose una especie de ruido en el primer caso. A si mismo, se realizo un ajuste de los cables en la Protoboard ya que se detecto que los mismos no lograban un contacto adecuado.

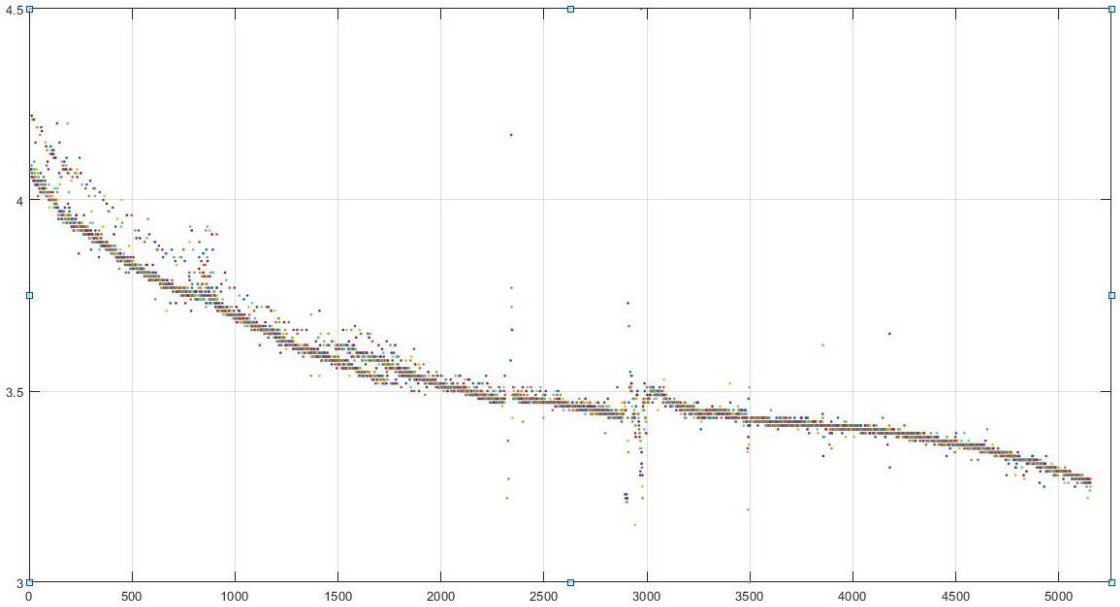


Fig. 9-a: Ensayo de relevamiento versión 1.

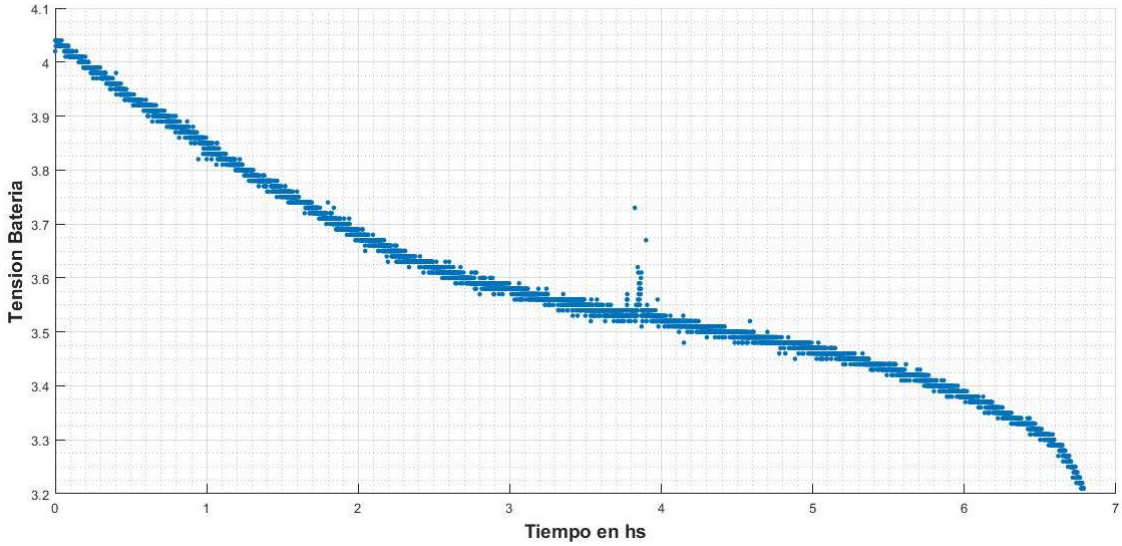


Fig. 9-b: Ensayo de relevamiento versión 2.

De los ensayos realizados hasta el momento, resulta satisfactorio obtener una duración de 7hs para un consumo de 110mA, pero esto se logro para un consumo de corriente constante, lo cual se sabe que no sucede en el ESP32, ya que el consumo aumenta cuando utiliza la transmisión WIFI. Para poder obtener una estimación de la duración de la batería cuando alimenta al Node MCU ESP32, se ensamblo el circuito final como se aprecia en la **Figura 10.**

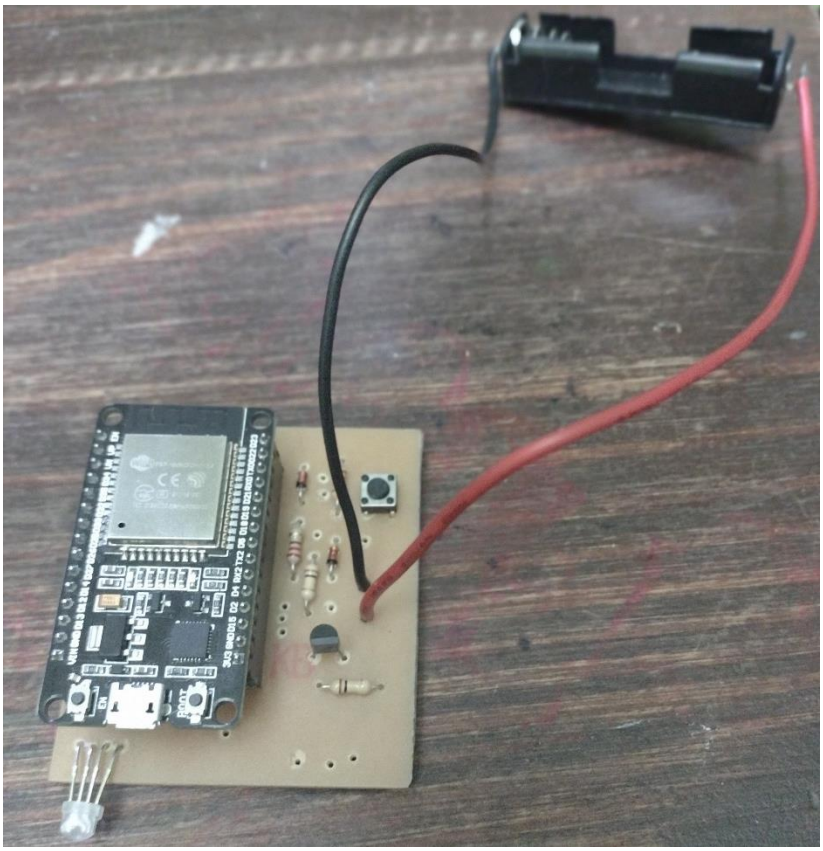


Fig. 10: Circuito final ensamblado.

Para realizar el ensayo de la duración de la batería en el circuito final, ya no sirve el sketch de Arduino para obtener muestras que utilizamos anteriormente dado que la única forma de conectarnos al ESP32 es a través de WIFI, y una vez que se agote la batería del mismo, se apagara automáticamente. Por tal motivo, se realizó una modificación en el Servidor Web para que este actualice automáticamente la lectura de la tensión en la batería, como así también, el tiempo (medido en milisegundos), desde que se encendió el mismo. De esta manera, cuando un cliente se conecte al servidor Web, este actualizara sus valores automáticamente, logrando que, cuando la batería se agote y el ESP32 se apague, queden los últimos valores actualizados en la página.

La página web requiere una actualización frecuente para cargas los datos del ESP32. Para resolver este problema, tiene dos opciones, la primera es actualizar la página con etiqueta HTML, lo cual no es muy efectivo. Para mostrar datos y actualizar, sin actualizar la página web necesitamos usar JavaScript Ajax.

¿Qué es AJAX?:

AJAX = Asíncronos JavaScript And XML

AJAX permite que las páginas web se actualicen de forma asincrónica intercambiando pequeñas cantidades de datos con el servidor detrás de escena. Esto significa que es posible actualizar partes de una página web, sin recargar toda la página. Las páginas web clásicas (que no utilizan AJAX) deben recargar toda la página si el contenido cambia.

Ejemplos de aplicaciones que utilizan AJAX: pestañas de Google Maps, Gmail, YouTube y Facebook.

Para el caso que se desarrolla en el presente informe, se deben crear tres páginas en el servidor. La primera, es la que se carga como una página web normal y la segunda página web está detrás de escena, es decir, AJAX.

AJAX se basa en los estándares de Internet y utiliza una combinación de:

- Objeto XMLHttpRequest (para intercambiar datos de forma asíncrona con un servidor)
- JavaScript / DOM (para mostrar / interactuar con la información)
- CSS (para diseñar los datos)
- XML (a menudo utilizado como formato para transferir datos)

Finalmente, el código utilizado para describir el nivel de la tensión en la batería es el siguiente:

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include "index.h" // Archivo de encabezado de página web
//---------------------- Datos Conexión WIFI --------------------------------
const char* ssid = "AQUI VA EL NOMBRE DE LA RED A CONECTARSE";
const char* password = "AQUI VA LA CONTRASEÑA DE LA RED A CONECTARSE";
WebServer server (80);
//---------------------- GPIO's a utilizar en el ESP32 ---------------------------------
#define LEDWIFI  2     // LED indicador de conexión WIFI en la placa del ESP32.
#define LEDBATERIA1 12 // LED VERDE indicador de batería con carga mayor a %50 en el terminal D12.
#define LEDBATERIA2 13 // LED AZUL indicador de batería con carga menor al %50 en el terminal D13.
#define LEDBATERIA3 14 // LED ROJO indicador de batería con carga menor al %10 en el terminal D14.
#define TENSIONBATERIA 34 // Lee la tensión de la batería para determinar el estado de carga.
#define ENCENDER 15 // GPIO para mantener encendido el ESP32 al presionar el pulsador.
//---------------------- Variables para almacenar datos ---------------------------------
int tiempo_inicial,tiempo_final=0;
float tension_bateria=0;
int tension_bateria_digital=0;
float tension_led_porcentaje=0;
String estadobateria = "";

void handleRoot () {
 server.send (200, "text/html", Web_page); // Enviar página web.
}
```

```
void handleTiempoBateria () {
 String TiempoValue = String (tiempo_final/1000);
 server.send (200, "text/plain", TiempoValue); // Enviar valor del tiempo solo a la solicitud AJAX del cliente.
}

void handleTensionBateria () {
 String TensionValue = String (tension_bateria);
 server.send (200, "text/plain", TensionValue); // Enviar valor de tensión solo a la solicitud AJAX del cliente.
}
// ============================================================
//                    Configuración
// ============================================================
void setup () {
  Serial.begin(9600);
  pinMode (LEDWIFI, OUTPUT);
  pinMode (LEDBATERIA1, OUTPUT);
  pinMode (LEDBATERIA2, OUTPUT);
  pinMode (LEDBATERIA3, OUTPUT);
  pinMode (ENCENDER, OUTPUT);

  digitalWrite (ENCENDER, HIGH); //GPIO para lograr el enclavamiento de encendido.

//Conecta a la red wifi.
  Serial.println();
  Serial.print("Conectando con ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);
  while (WiFi.status()! = WL_CONNECTED) {
    digitalWrite (LEDWIFI, LOW);
    Serial.print(".");
    delay (500);
    digitalWrite (LEDWIFI, HIGH);
    delay (500);
  }
```

```
  Serial.println("Conectado con WIFI.");
  digitalWrite (LEDWIFI, LOW);

  // Inicio del Servidor web.
  server.on ("/", handleRoot); // Esta es la página de visualización.
  server.on ("/tensionbateria", handleTensionBateria); // Para obtener la actualización del valor de tensión en la batería.
  server.on ("/tiempobateria", handleTiempoBateria); // Para obtener la actualización del valor de tensión en la batería.
  server.begin (); // Iniciar servidor
  Serial.println("Servidor WEB iniciado.");

  // Esta es la IP para conectarse al servidor WEB
  Serial.print("Esta es la IP para conectar: ");
  Serial.print("http://");
  Serial.println(WiFi.localIP());
}
void loop () {
  /*
  Como el valor más alto que toma la batería es 4.2V, corresponderá a un valor del ADC de 4095. Pero como la tensión máxima de entrada en los pines es 3.3V utilizo un divisor resistivo conformado por una resistencia de 10k y una de 4k7, por lo tanto, cuando la tensión en la batería sea 4.2V, en el PIN tendremos 2.8571V, y cuando la batería caiga al valor mínimo de 3.5V, la tensión en el PIN será 2.381V. En valores digitales tendremos
  2.8571V ---> 3545 ---> 4.2V
  2.5169V ---> 3123 ---> 3.7V
  2.3129V ---> 2870 ---> 3.4V
  2.2448V ---> 2786 ---> 3.3V
  */
  tiempo_final= millis ();
  tension_bateria_digital=analogRead (TENSIONBATERIA);
  tension_bateria=0.15 + (tension_bateria_digital * 1.47 * (2.8571/3545));
  //Ecuación para calcular el porcentaje de carga de la batería.
  tension_led_porcentaje= (((tension_bateria – 3.3) / (4.2 – 3.3)) * 100);

  if (tension_led_porcentaje > 50) {
    digitalWrite (LEDBATERIA2, LOW);
    digitalWrite (LEDBATERIA3, LOW);
```

```
  estadobateria="batcompleta";
  digitalWrite (LEDBATERIA1, HIGH);
 }
 if (10 < tension_led_porcentaje && tension_led_porcentaje < 50) {
  digitalWrite (LEDBATERIA1, LOW);
  digitalWrite (LEDBATERIA3, LOW);
  estadobateria="batmitad";
  digitalWrite (LEDBATERIA2, HIGH);
 }
 if (tension_led_porcentaje <= 10) {
  digitalWrite (LEDBATERIA1, LOW);
  digitalWrite (LEDBATERIA2, LOW);
  estadobateria="batagotada";
  digitalWrite (LEDBATERIA3, HIGH);
 }
 if(tension_bateria<3.3) {
  digitalWrite (ENCENDER, LOW); //APAGO EL ESP32 PARA PROTEGER LA BATERIA.
 }
 server.handleClient (); // Manejar las solicitudes de los clientes.
 delay (1000);
}
```

Utilizando el código anterior, se procedió por ensayar la duración de la batería, utilizando un objeto XMLHttpRequest con el fin de actualizar los datos cada 1 segundo. Se dejo el dispositivo encendido durante toda la noche, y a la mañana siguiente se detectó un tiempo medido de 31452 segundos, lo que equivale aproximadamente, a 8:30hs. Como se observa en la **Figura 11-a y b**, la tensión en la batería disminuyo a 3.14V, lo cual no sucederá en nuestro caso ya que se pretende que el dispositivo se apague automáticamente cuando la tensión llegue a 3.3V.

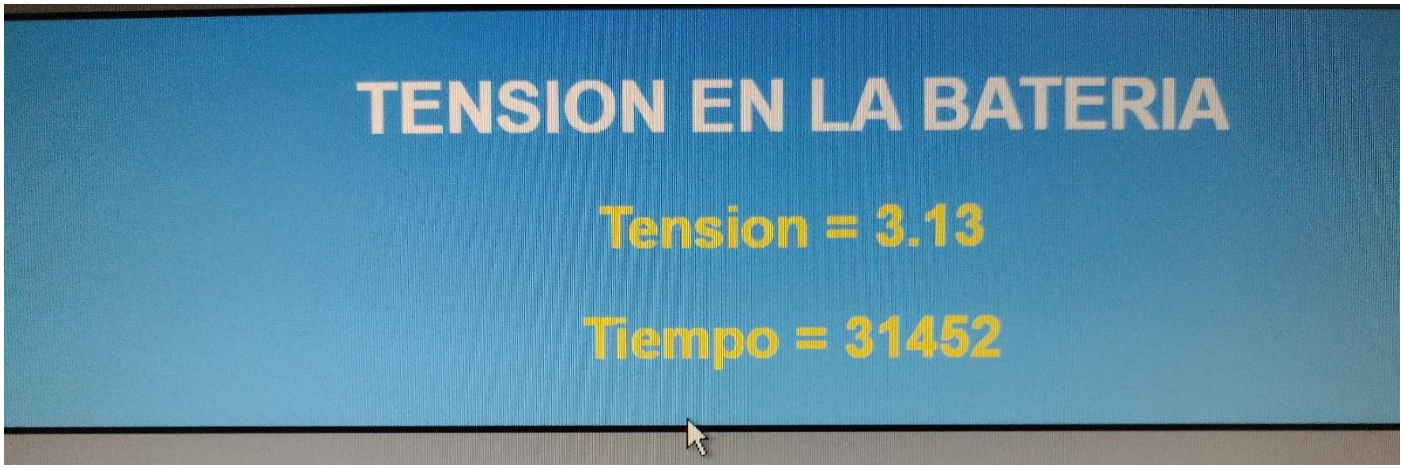


Fig. 11-a: Ensayo duración batería alimentando al Node MCU ESP32.

Fig. 11-b: Ensayo duración batería alimentando al Node MCU ESP32.

En la última imagen vemos que, a los 20821 segundos, equivalente a casi 6hs, la tensión se mantiene por encima de los 3.3V.