



thinger.io
platform

MÓDULO DE “PROYECTO”

DEPARTAMENTO DE ELECTRICIDAD

PRODUCCIÓN SOLAR CON ESP8266

Dispositivo IOT con ESP8266 para la monitorización de una instalación solar fotovoltaica.

Carlos Sánchez Prudencio

Sistemas eléctricos y automatizados
IES CIUDAD JARDÍN



Módulo de proyecto

MONITORIZACIÓN DE UNA INSTALACIÓN SOLAR FOTOVOLTAICA.

Badajoz, Badajoz.



PETICIONARIO: departamento de electricidad del IES Ciudad Jardín.

Autor: CARLOS SÁNCHEZ PRUDENCIO.

IES Ciudad Jardín, Badajoz a 13 de Junio del 2018

ÍNDICE

	Págs.
1. Introducción	1
1.1. Primeros pasos	1
1.1.1. Instalar la placa del ESP8266	1
1.1.2. Instalar las bibliotecas necesarias	2
1.2. Comienzo del proyecto	3
1.2.1. RTC	3
1.2.2. Pantalla LCD	5
1.2.3. Zócalo para microSD	6
1.2.4. Multiplexor 74HC6047	8
1.2.5. Conversor de niveles lógicos 3.3 - 5v	9
1.2.6. NodeMCU – ESP8266	10
1.2.7. Sensor ACS712 20A	11
1.2.7.1. Calibrando el ACS712	12
1.3. Conceptos básicos	13
1.3.1. Bus Serial	13
1.3.2. Bus I2C	14
1.3.3. Bus SPI	15
2. Aplicando lo aprendido	16
2.1. Primera parte: Introducción	16
2.2. Segunda parte: El código	18
2.3. Tercera parte: las mediciones	21
3. Servidor Thinger.io	22
3.1. ¿Qué es Thinger.io?	22

3.2.	Agregar un dispositivo o “device”	23
3.3.	Aclaraciones	24
3.4.	Dashboard	24
3.4.1.	Añadir un dashboard	24
3.4.2.	Usar un dashboard	24
3.4.3.	Configuraciones	26
3.4.3.1.	Tipo display	26
3.4.3.2.	Tipo device control	27
3.5.	Modo stream	27
3.6.	Compartir un dashboard	27
3.7.	Data bucket	27
4.	Manejo de datos guardados	30
4.1.	Micro SD	30
4.2.	Plataforma Thinger.io	31
5.	Aplicación para Android	32
5.1.	Descargando la app	32
5.2.	Probando la aplicación	34
6.	Código Arduino para Thinger.io	35
6.1.	Bibliotecas	35
6.2.	Estructuras básicas	37
7.	Instalando el dispositivo	39
7.1.	Conexión de los sensores	40
8.	Soluciones a problemas generales	41
8.1.	Problema con pantalla LCD al iniciar	41
8.2.	Problema con tarjeta microSD	41
8.3.	Problema con entradas analógicas	41

8.4.	Problema con regulador ISFV	41
8.5.	Problema con dashboard vacío	42
8.6.	Problema con las tensiones	42
8.7.	Problema con RTC	42
8.8.	Problema con el inversor de tensión	42
9.	Posibles mejoras	43
9.1.	Sensor de temperatura	43
9.2.	Añadir Relés de control	43
9.3.	Led's indicadores	43
9.4.	Código vía OTA (Over The Air)	43
9.5.	Utilizar una nueva versión del ESP8266 → El ESP32	43
9.6.	Utilizar un SAI	44
9.7.	Modificación de código	44
9.8.	Utilizar RaspberryPi3	44
9.9.	Modificación para corriente alterna	44
9.10.	Midiendo la eficiencia	44
10.	Esquemas con especificaciones	45
10.1.	Esquema PCB	46
10.2.	Esquema de conexiones	47

1.Introducción

Este proyecto pretende construir un medidor de corrientes y tensiones de una instalación solar fotovoltaica o ISFV. Para ello se utilizan varios dispositivos de control y medida que iremos explicando poco a poco.

En primer lugar, el microcontrolador ESP8266, es el que contiene toda la programación por así decirlo es “el cerebro” de toda la operación.

El dispositivo ESP8266 es un chip con opción de WiFi integrado. El chip primero llegó a la atención de los fabricantes occidentales en agosto de 2014 con el módulo ESP-01.

<https://aprendiendoarduino.wordpress.com/2017/09/12/que-es-esp8266/>

En este proyecto Utilizaremos una variante de dicho chip, el llamado NodeMCU, que incorpora salidas y entradas para configurarlo a través del IDE de Arduino.

1.1 Primeros pasos

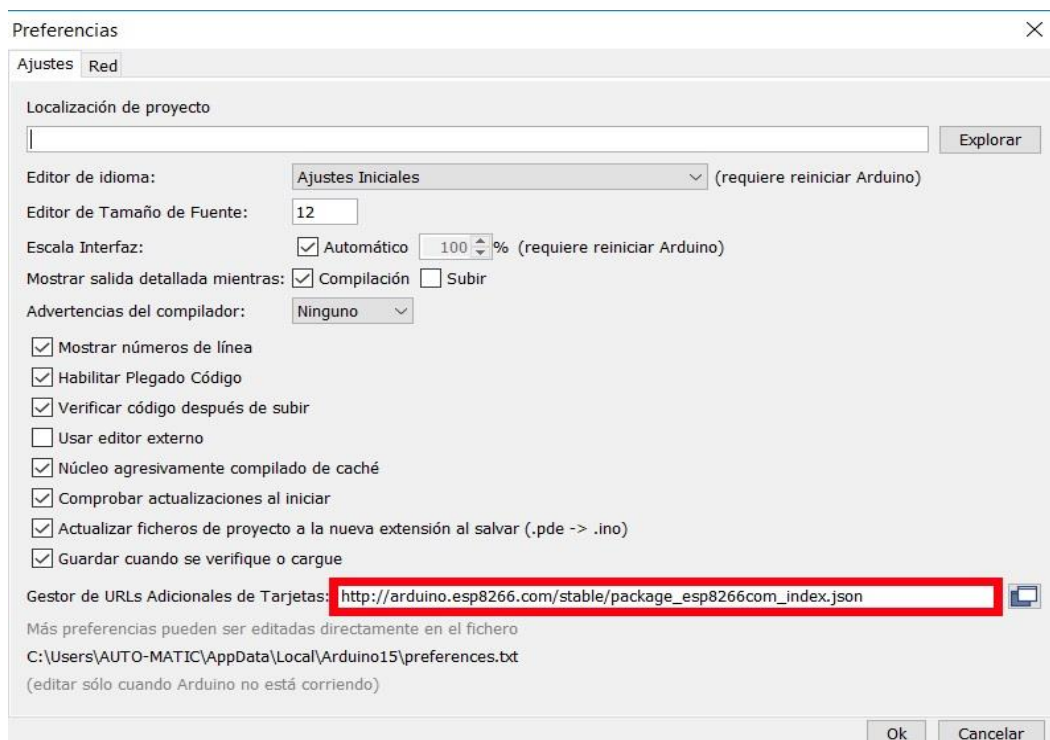
Para poder usar el ESP8266 con el IDE de Arduino necesitaremos descargar los drivers y bibliotecas necesarias, para ello:

1.1.1 Instalar la propia placa del ESP8266:

- a. Abrimos el IDE de Arduino y en Archivo → preferencias → Gestor de Url's adicionales de tarjetas, incluimos el siguiente enlace:

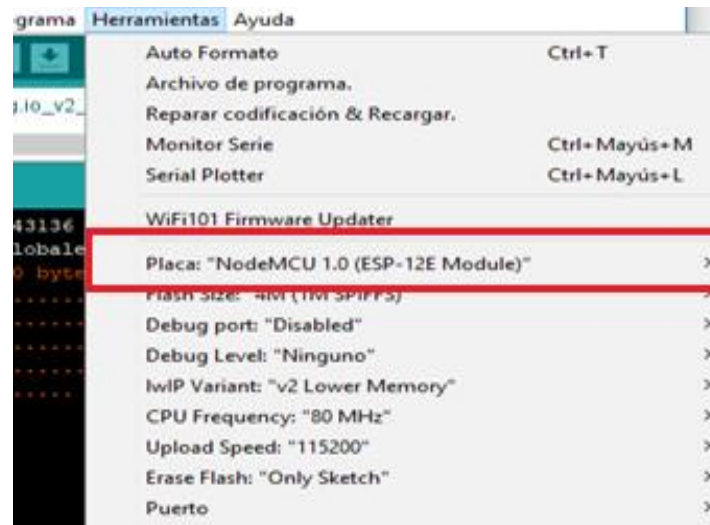
http://arduino.esp8266.com/stable/package_esp8266com_index.json

- b. Damos a OK y reiniciamos el IDE
- c. Abriremos otra vez el programa y en Herramientas → Placa, estará añadida en “ESP8266 Modules”.



d. Utilizaremos la versión de placa que estemos usando, en nuestro caso:

“NodeMCU 1.0 (ESP-12E Module)” o “NodeMCU 0.9 (ESP-12 Module)”



1.1.2 Instalar las bibliotecas para utilizar los demás elementos:

a. Para usar el reloj de tiempo real o RTC:

<https://github.com/ltoll/arduino-libraries>

b. Para usar la LCD con módulo I2C:

<https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library>

c. Para usar el servidor thinger.io:

<http://docs.thinger.io/arduino/#installation-library-manager>

d. Para usar el WiFi del ESP8266:

Lo lleva incorporado una vez instalada la placa.

e. Para usar la SD:

<https://github.com/adafruit/SD>

f. Para usar la biblioteca Wire.h:

La lleva incorporada en el propio IDE.

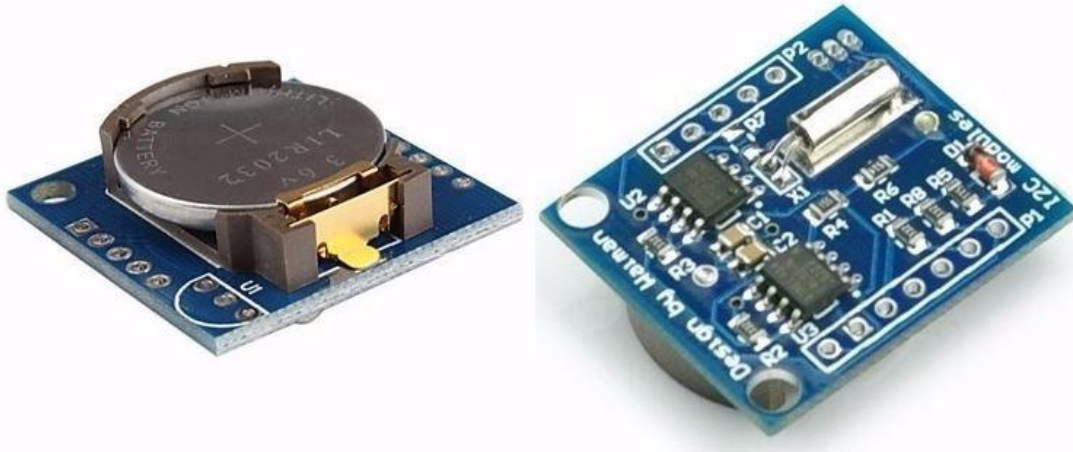
Nota: para descargar los repositorios de GitHub basta con dar clic en “Clone or download” → “Download ZIP”. Una vez descargados tan solo habrá que añadir los ZIPs a nuestro IDE, para ello:

1. Programa → Incluir librería → Añadir librería .ZIP
2. Buscaremos nuestras descargas y haremos clic para añadirlas.
3. Reiniciamos el IDE y ya podremos utilizarlas.

1.2 Comienzo del proyecto

Una vez instaladas todas las bibliotecas de los demás dispositivos procederemos a explicarlos uno a uno.

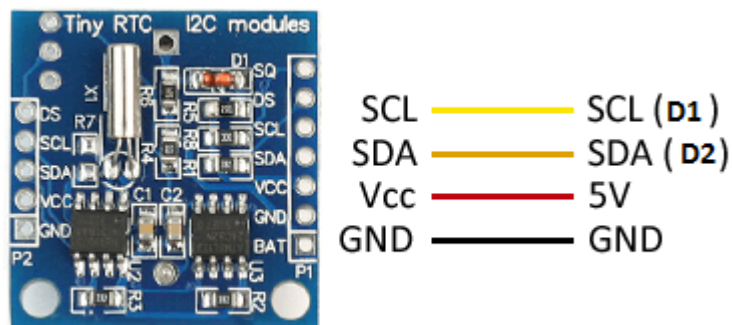
1.2.1 RTC: es un dispositivo que sirve para medir el tiempo, tal y como su nombre indica, reloj de tiempo real o real time clock.



Para utilizar la RTC cogeremos la biblioteca de funciones RTC.h y la incluiremos en nuestro sketch.

Funciona mediante el bus de comunicación I2C que explicaremos cómo funciona más adelante.

Tan solo debemos conectarlo a las patillas del ESP8266 habilitadas para ello como se muestra en la imagen:



Nota: para utilizarlo en un ESP8266 podremos de añadir un convertidor de nivel (5V-3.3V) para asegurarnos con total certeza que la tensión de los hilos no excede de los 3.3V, pero experimentos demuestran que no es necesaria esta incorporación puesto que es el maestro y no el esclavo el que decide el nivel lógico (véase apartado 1.3.2 “Bus I2C” de este proyecto).

Para configurar la RTC deberemos primero ver qué modelo es el que tenemos. Normalmente viene impreso en la propia placa del reloj pero si no es nuestro caso, tan solo debemos de consultar la ficha técnica.

El modelo más usual es el DS1307 o el DS3231, en este proyecto se utiliza el DS1307.

RTC_DS1307 RTC; //Variable de RTC tipo DS3231 o DS1307 ¡OJO!

Para ponerlo en marcha (una vez creada la variable RTC de tipo RTC_DS1307) en el **setup** debemos de:

1. Inicializarla:

RTC.begin(); //Iniciar RTC

2. Fijar la fecha de compilación:

RTC.adjust(DateTime(__DATE__, __TIME__)); //Para ajustar fecha y hora

Si la línea anterior la dejamos descomentada y se reinicia nuestro dispositivo, la fecha y la hora se volverá a ajustar a la hora de compilación del Sketch, por ello una vez que subamos nuestro código, deberemos de comentar esta línea y volver a cargar el Sketch.

La RTC nos puede dar dos errores muy comunes:

1. Fallo en la comunicación:

2165/165/165 165:165:85

2. Batería baja o desconectada:

2000/1/1 0:0:0

En el **loop**, declaramos una variable tipo DateTime:

DateTime now = RTC.now(); //Variable para RTC

Si queremos visualizar la fecha y la hora por el monitor serie tan solo debemos de escribir lo siguiente:

1. Fecha:

- a. Día: `"now.day();"`
- b. Mes: `"now.month();"`
- c. Año: `"now.year();"`

2. Hora:

- a. Hora: `"now.hour();"`
- b. Minuto: `"now.minute();"`
- c. Segundo: `"now.second();"`

Ejemplo: para saber la fecha y hora en este momento:

```
Serial.print(now.day());  
  
Serial.print(now.month());  
  
Serial.print(now.year());  
  
Serial.print(now.hour());  
  
Serial.print(now.minute());  
  
Serial.println(now.second());
```

Con esto deberíamos de poder usar la RTC con normalidad. Claro que podremos añadir los caracteres `'/'` para separar la fecha y `':'` para visualizar mejor la hora.

1.2.2 Pantalla LCD 16x2: es una pantalla LCD que posee dos líneas de dieciséis caracteres cada una y que utilizaremos para mostrar la fecha, la hora y los valores de la ISFV.

Esta pantalla lcd posee comunicación por paralelo que requiere dieciséis pines, para no necesitar tantos cables se le añade el módulo I2C que reduce la comunicación a cuatro hilos por lo que es más sencillo de manejar.



Así pues tan solo lo conectamos al bus I2C y a la alimentación de 5v.

Nota: Al estar conectado al mismo bus de comunicación que la RTC y necesitar 5V de alimentación, podríamos intercalar también un convertidor de niveles lógicos, pero no es necesaria su incorporación.

Configurar esta pantalla con opción al bus I2C es más sencillo de lo que parece:

1. Crear variable para la lcd:

LiquidCrystal_I2C lcd(0x27, 16, 2); //Variable para lcd

2. En el **setup** la inicializaremos:

lcd.begin(); //Iniciar pantalla lcd

Hay una opción en la que podemos configurar la luz de fondo:

lcd.backlight(); //Iniciar luz de fondo

Una vez inicializada la LCD en el **loop** podemos escribir las siguientes funciones para utilizar la pantalla:

1. Poner puntero en cualquier parte de la LCD. Para ello utilizamos la función “setCursor” y entre paréntesis primero el número de carácter y segundo la línea que queramos. Por ejemplo si quisiéramos poner el cursor en el primer carácter de la primera línea, debemos escribir:

lcd.setCursor(0, 0);

2. Para imprimir o escribir un carácter en la pantalla, la función “print”. Por ejemplo:

lcd.print(“Hello World”);

Nota: la dirección I2C de este tipo de pantallas suele ser “0x27” pero si no lo fuera podremos subir el código “I2C Scanner” que encontraremos en el siguiente enlace:

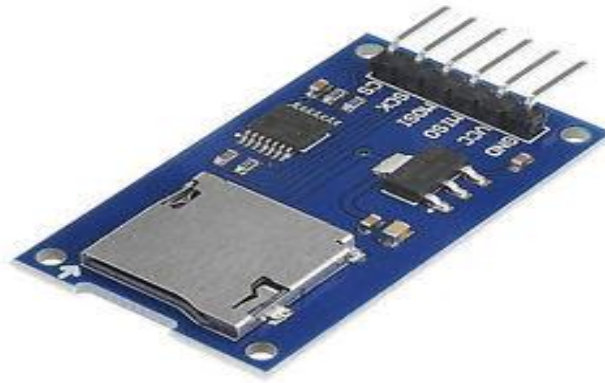
<https://playground.arduino.cc/Main/I2cScanner>

1.2.3 Zócalo para microSD

Este dispositivo sirve para poder utilizar una tarjeta microSD en el ESP8266.

Se sirve del bus de comunicación SPI que explicaremos más adelante con detalle. Utilizaremos la biblioteca de funciones “SD.h” para su funcionamiento.

Nota: este zócalo no es más que una especie de fuente de alimentación para la tarjeta, así pues, aunque esté conectado, si no está presente la microSD, nos dará error al inicializarla.



Los pines que debemos conectar son los siguientes:

- **Miso:** “Master in → slave out” este pin es el encargado de transmitir datos desde el esclavo hasta el maestro. En el ESP8266 se conectará al pin D6.
- **Mosi:** “Master out → slave in” se encarga de transmitir datos desde el maestro hasta el esclavo. Lo conectaremos al pin D7.
- **CS:** “Chip Select” es el pin propio de cada esclavo. Normalmente está a “1” o “activo” cuando no es elegido para transmitir datos pero cuando cae a “0” o “inactivo” es el elegido para comunicación entre maestro y esclavo. En el ESP8266 lo conectaremos al pin D8.
- **CLK:** es la señal de reloj que sincroniza el maestro con los esclavos. Se conectará al pin D5.

Nota: podríamos incorporarle también un convertidor de niveles lógicos pero no es necesario puesto que es el maestro el que decide qué tensión lógica debe haber para la comunicación, en este caso, 3.3V.

Para su uso:

1. Debemos definir en qué patilla pondremos el pin CS del bus SPI:

“#define Chip_Select_SD D8//Pin al que se conectará la SD”

2. Elegiremos el nombre de una variable del tipo File:

“File dataFile;//Variable para SD”

3. En el **setup** la inicializaremos:

“SD.begin(Chip_Select_SD);//Inicializar la SD”

Las funciones a utilizar son las siguientes:

1. SD.open → Para abrir un archivo existente, si no existe, lo creará
 - a. Entre paréntesis y con comillas dobles escribiremos el nombre del archivo al que queremos acceder.
 - b. Pondremos una coma y escribimos FILE_WRITE para poner el modo escritura activo en el archivo creado anteriormente.
2. dataFile.print → Para insertar un texto, variable, etc en el archivo una vez abierto y en modo escritura.
 - a. Entre paréntesis escribimos lo que queramos guardar en la SD.
3. dataFile.close → Para cerrar el archivo. No olvidar los paréntesis “()”.

En este proyecto se abrirá un archivo llamado “VALORES” con extensión “.csv” (valores separados por comas en inglés).

Así en nuestro código Arduino tendremos una función que se llamará “*escribir_tarjeta*” que servirá para almacenar los valores que vayan adquiriendo las entradas analógicas.

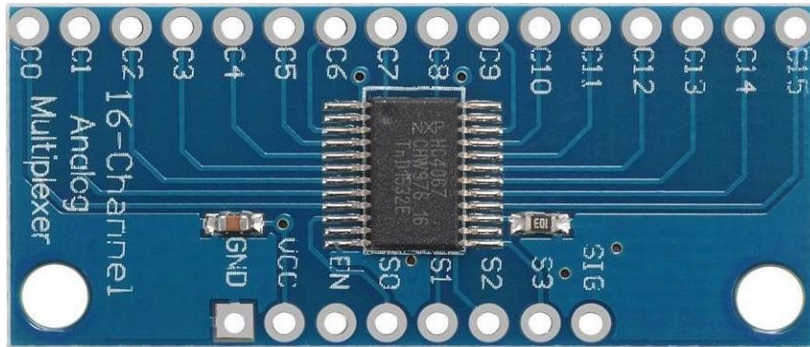
```
void escribir_tarjeta(String strValues) {  
    File logFile;  
    logFile = SD.open("datalog.csv", FILE_WRITE);  
    if (logFile) {  
        logFile.println(strValues);  
        logFile.close();  
    }  
    else {  
        Serial.println("Error al abrir el archivo");  
    }  
}
```

Código para SD

Nota: si se confundieran los pines MOSI y MISO, esto daría un error fatal a la hora de la programación puesto que se saturan los puertos de comunicación y el IDE de Arduino no sería capaz de acceder a la memoria de programa.

1.2.4 Multiplexor

Debido a que el ESP8266 tan solo posee una entrada analógica y para nuestro proyecto necesitamos seis, utilizamos este multiplexor analógico de dieciséis canales.



Básicamente el funcionamiento de un multiplexor es dar acceso a por ejemplo dos entradas o salidas utilizando tan solo un cable.

En nuestro proyecto utilizaremos un multiplexor 16 x 4 que con cuatro bits de información podremos tener dieciséis canales de entrada.

Hay una entrada que es la llamada “SIG” para que las entradas sean analógicas, es decir, que si está conectado a una entrada analógica y es leída esa entrada, lo hace a través del convertidor ADC.

Para más información seguir este enlace:

https://assets.nexperia.com/documents/data-sheet/74HC_HCT4067.pdf

Para hacer funcionar este multiplexor, debemos crear una función que vaya cambiando de canal:

```

143  /** FUNCIÓN PARA UTILIZAR EL MULTIPLEXOR */
144  int SetMuxChannel(byte channel)
145  {
146      digitalWrite(muxS0, bitRead(channel, 0));
147      digitalWrite(muxS1, bitRead(channel, 1));
148      digitalWrite(muxS2, bitRead(channel, 2));
149      digitalWrite(muxS3, bitRead(channel, 3));
150  }
```

En resumen, llamaremos a esta función cuyo valor entre paréntesis iremos cambiando de 0 a 15.

Si quisiéramos utilizar el pin analógico, primero habrá que llamar a la función analogRead() y después elegir el canal que vayamos a utilizar.

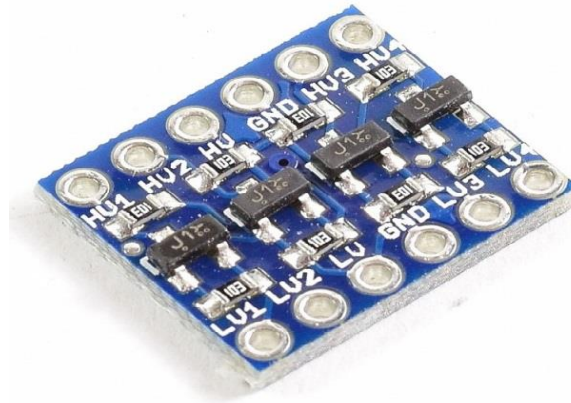
Para utilizarlo en el ESP8266 declaramos las siguientes variables:

- **muxS0 = D0;**// Primer bit para el multiplexor
- **muxS1 = D3;**// Segundo bit para el multiplexor
- **muxS2 = D4;**// Tercer bit para el multiplexor
- **muxS3 = D9;**// Cuarto bit para el multiplexor
- **muxSIG = A0;**// Señal analógica del multiplexor

1.2.5 Conversor de niveles lógicos 3.3 - 5v

Es un dispositivo que convierte una tensión a otra, en este caso de 5v a 3.3v. También puede usarse al revés, para convertir los 3.3v a 5v teniendo siempre presente no mezclar las tensiones.

Nosotros lo utilizaremos para la conversión de 5v a 3.3v para los buses de comunicación tanto del I2C como del SPI.

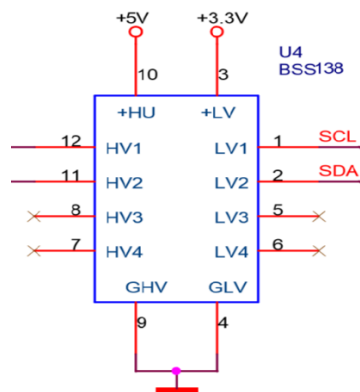


En este proyecto no necesitamos configurar nada en este dispositivo tan solo conectarlo a las patillas correspondientes.

Hv1, Hv2, Hv3 y Hv4 serán conectadas a los pines que lleven 5v, Gnd y Hv a la respectiva alimentación. Y con Lv1, Lv2, Lv3 y Lv4 lo mismo pero a 3.3v.

Para más información:

<http://www.agspecinfo.com/pdfs/B/BOB08745.PDF>

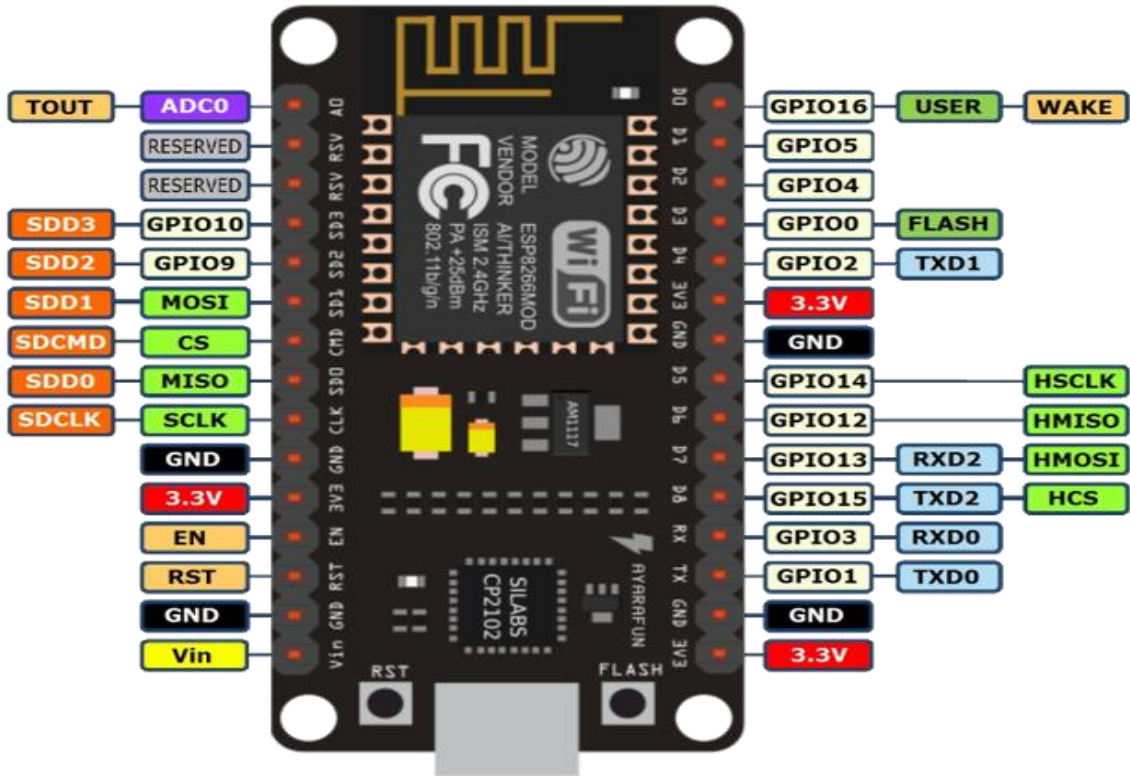


Nota: tener cuidado con conectar las tensiones que no correspondan a su mismo nivel, es decir, HV siempre estará conectado a 5V y LV a los 3,3V.

1.2.6 NodeMCU - ESP8266

Este es el dispositivo que utilizaremos como “cerebro” de todo el proyecto, aquí es donde subiremos nuestro código de programación y conectaremos todos los aparatos descritos anteriormente.

La siguiente imagen representa el PinOut del ESP8266:



Conexiones en el ESP8266 en nuestro proyecto:

Bus I2C

- SDA → D2
- SCL → D1

Bus SPI

- MOSI → D7
- MISO → D6
- SCK → D5
- CS → D8 (Pero puede conectarse en cualquier pin libre que esté en el bus SPI)

Multiplexor

- S0 → D0
- S1 → D3
- S2 → D4
- S3 → D9
- SIG → A0

Alimentación

- +5v → Vin
- Gnd → Gnd

En el siguiente enlace está todo lo relacionado con el NodeMCU – ESP8266:

https://cdn-shop.adafruit.com/product-files/2471/OA-ESP8266_Datasheet_EN_v4.3.pdf

Nota: este dispositivo trabaja como máximo a 3.3v aunque su alimentación pueda ser de hasta 5v, puesto que lleva incorporado un regulador en la propia placa.

1.2.7 Sensor ACS712

Este es un sensor de corriente. Para ello se sirve del efecto hall que no es más que la aparición de un campo eléctrico por un conductor por el cual atraviesa un campo magnético, esto es que se genera tensión con la aparición de una corriente a través del chip ACS712.



Para la configuración de este sensor tan solo debemos fijarnos cual es la máxima corriente que soporta puesto que a la hora de realizar la conversión debemos usar una constante u otra.

En nuestro caso serán de 20A como máximo, así pues según los siguientes valores, deberemos escoger 0,100:

Tipo 5A: 0.185 → Esto quiere decir que el rango de valores está en 185mV/A.

Tipo 20A: 0.100 → Esto quiere decir que el rango de valores está en 100mV/A.

Tipo 30A: 0.66 → Esto quiere decir que el rango de valores está en 66mV/A.

La patilla de Vcc se conectará a los 3.3v que produce el ESP8266, el GND al negativo y la salida (OUT) a uno de los canales habilitados en el multiplexor.

En el siguiente enlace explica cómo utilizar este sensor hall:

http://www.naylampmechatronics.com/blog/48_tutorial-sensor-de-corriente-ac712.html

La fórmula para usar este sensor es la siguiente:

$$A = \frac{\text{Voltage}_i \cdot \text{analogRead}(A0) - \text{Offset}_i}{\text{Relation}_i}$$

El **valor ADC** dependerá de la resolución del convertidor AD, que en este caso es 1023.

El **valor de Voltage_i** es debido a la tensión máxima del ESP8266.

El **valor de Offset_i** deberá de ser calibrado para una corriente de 0A.

1.2.7.1 Calibrando el ACS712

Para calibrar este sensor debemos de conocer tres parámetros básicos que son la tensión a la que se alimenta, la relación de nuestro sensor según la tensión que entrega y el Offset o punto cero.

Nuestra placa como máximo nos da una tensión de 3.3 voltios, pero si queremos ser más exactos en nuestras mediciones de valores debemos de medir esa tensión teórica.

Una vez medida esta tensión nos da 3.27 voltios, es la que pondremos en la fórmula de arriba.

La variable *relation_i* es la relación que existe entre la tensión que entrega el sensor y las unidades del sensor, es decir, la tensión que nos entrega por cada amperio de entrada.

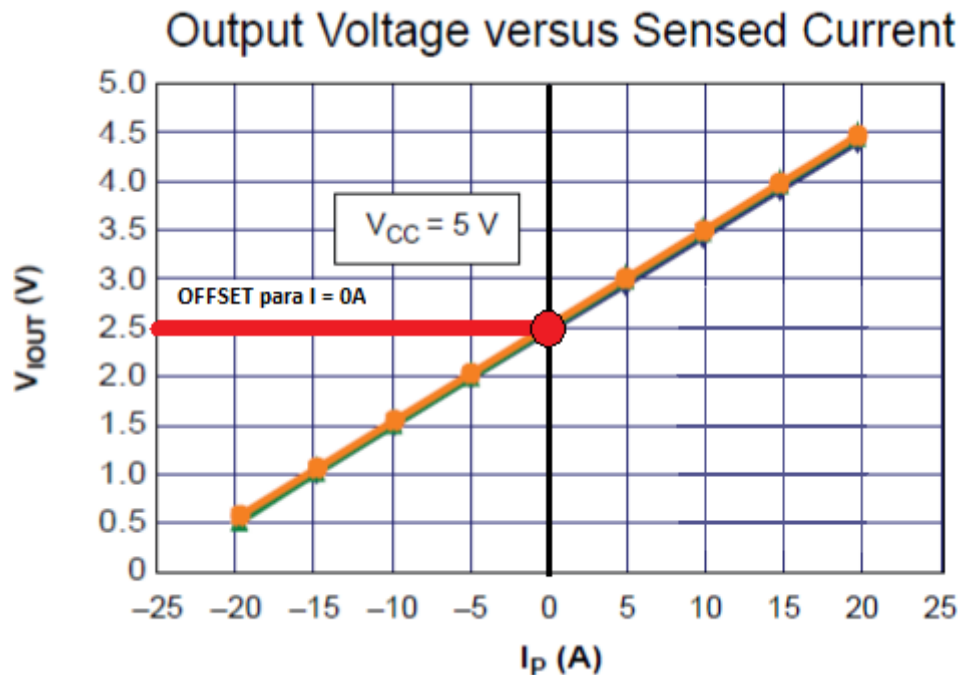
Que en este caso debido a que nuestro sensor hall es de 20A la relación es de 100mV/A.

Pero esta relación está medida para una tensión de alimentación de 5V y el esp8266 solo nos da 3.27V es fácil pensar que habría de escalarlo o al menos calcular la nueva relación. Pero no, la relación es la misma esté alimentado a 3.3V como a 5V.

Y así nos acercamos a la variable *Offset_i* que es la variable por así decirlo más importante de la calibración puesto que es la que marca qué tensión nos llega cuando nuestro sensor marca cero.

Por ejemplo, en este sensor y para 5V el *Offset_i* sería 2.5 pero puesto que nuestra tensión es distinta a esa tendremos que escalarla, así pues nuestro valor será 1.635.

En la gráfica siguiente se muestra la tensión que entrega el ACS712-20A según la corriente que circule por él:



1.3 Conceptos básicos

Daremos unas nociones básicas sobre el bus de comunicación serial, I2C y el SPI.

1.3.1 Bus Serial

El concepto de comunicación serial es sencillo. El puerto serial envía y recibe bytes de información un bit a la vez.

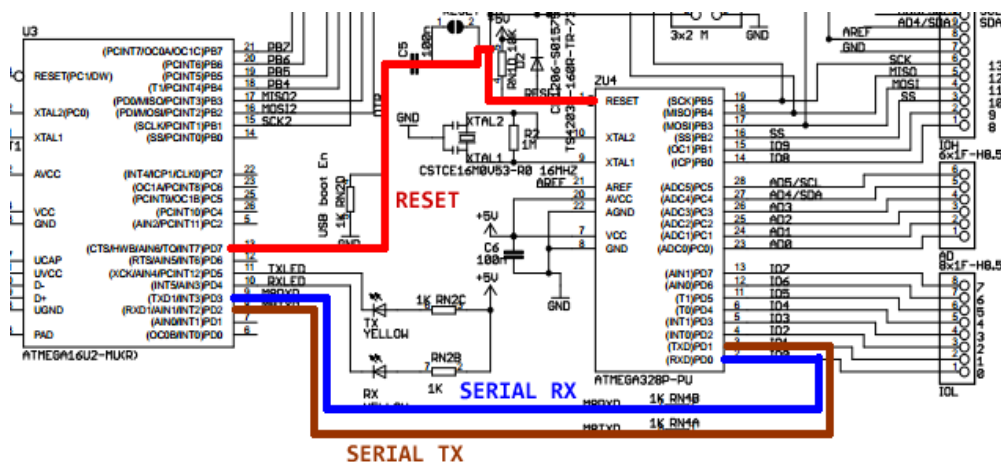
Aún y cuando esto es más lento que la comunicación en paralelo, que permite la transmisión de un byte completo por vez, este método de comunicación es más sencillo y puede alcanzar mayores distancias. Por ejemplo, la especificación *IEEE 488* para la comunicación en paralelo determina que el largo del cable para el equipo no puede ser mayor a 20 metros, con no más de 2 metros entre cualesquiera dos dispositivos; por el otro lado, utilizando comunicación serial el largo del cable puede llegar a los 1200 metros.

Típicamente, la comunicación serial se utiliza para transmitir datos en formato ASCII. Para realizar la comunicación se utilizan 3 líneas de transmisión: (1) Tierra (o referencia), (2) Transmitir, (3) Recibir. Debido a que la transmisión es asincrónica, es posible enviar datos por una línea mientras se reciben datos por otra.

Existen otras líneas disponibles para realizar *handshaking*, o intercambio de pulsos de sincronización, pero no son requeridas. Las características más importantes de la comunicación serial son la velocidad de transmisión, los bits de datos, los bits de parada, y la paridad. Para que dos puertos se puedan comunicar, es necesario que las características sean iguales.

En nuestro ESP8266 tenemos dos patillas que son para leer y recibir datos (Rx y Tx). Vienen por definición acopladas al puerto micro USB para conectarnos a nuestro ordenador. Si quisiéramos utilizar otros pines, debemos utilizar la biblioteca de funciones “<SoftwareSerial.h>”.

En este diagrama se muestra la conexión serie entre dos placas para comunicación entre ellas:



Cuando queramos subir un código a nuestra placa estos pines deben de estar libres de toda conexión, es decir, que si conectamos cualquier dispositivo hemos de desconectarlo de cualquier manera, ya sea a través de un interruptor o quitándolo directamente. De lo contrario, se producirán errores y no seremos capaces de subir nada a nuestro dispositivo.

1.3.2 Bus I2C

Es un sistema muy utilizado en la industria para la comunicación entre circuitos integrados. El nombre viene del nombre “*inter-integrated circuit*”.

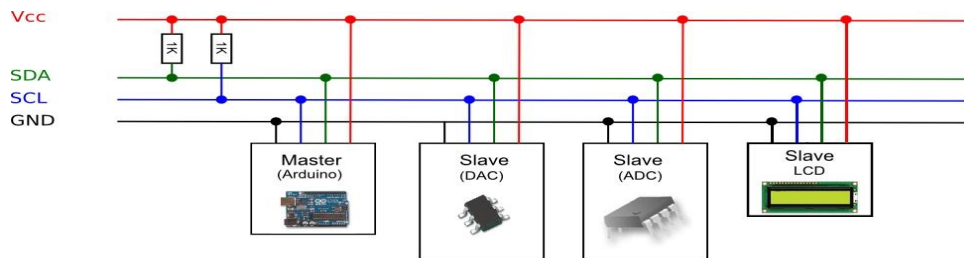
Su principal característica es que utiliza dos cables para la transmisión de datos:

- **SDA:** sirve para transferir los datos.
- **SCL:** para la transmisión de la señal de reloj.

Por “señal de reloj” se entiende una señal binaria de una frecuencia periódica muy precisa que sirve para coordinar y sincronizar los elementos integrantes de una comunicación de forma que todos sepan cuándo empieza, cuánto dura y cuándo acaba la transferencia de información.

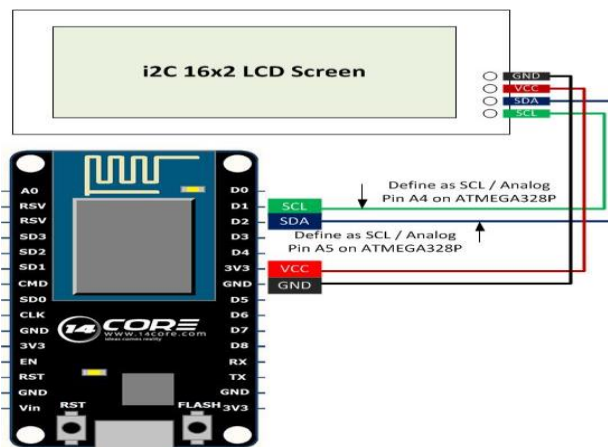
Cada dispositivo conectado al bus I2C tiene una dirección única que lo identifica respecto al resto de dispositivos, y puede estar configurado como maestro (master) o esclavo (slave).

El master es el que comienza la transferencia y además genera la señal de reloj, pero no es necesario que el maestro sea siempre el mismo dispositivo, es decir, que se pueden ir cambiando esta característica entre los demás que la posean.



Tal y como se muestra en el diagrama anterior, para funcionar correctamente, tanto la línea SDA como la SCL deben de estar conectadas a una resistencia “PULL_UP” a la fuente de alimentación común, pero en el NodeMCU ESP8266 estas resistencias las lleva integradas, por lo que no es necesario añadirlas.

Al haber una única línea de datos, la transmisión de información es “*half dúplex*”, es decir, que la información solo se transmite en un solo sentido al mismo tiempo, por lo que en el momento que un dispositivo empiece a recibir un mensaje, tendrá que esperar a que el emisor deje de transmitir información para poder responderle.



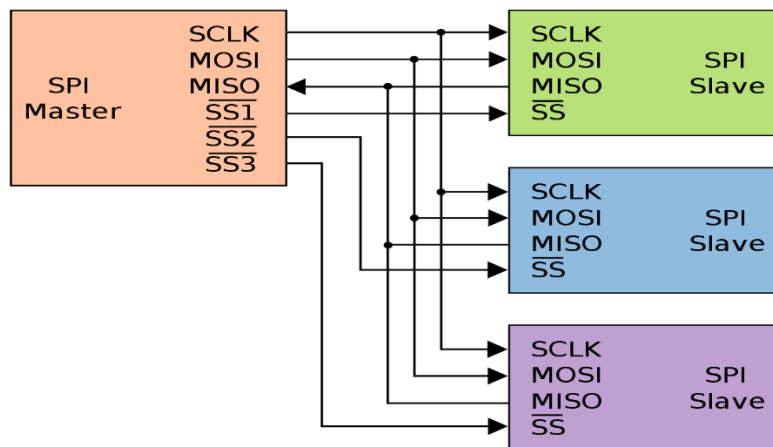
Conexión I2C para ESP8266

1.3.3 Bus SPI

Su nombre proviene de “Serial peripheral interface” y al igual que el sistema I2C, el SPI es un estándar que permite controlar a cortas distancias casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie sincronizado, es decir, regulado por un reloj.

La diferencia entre el I2C y el SPI es que el SPI necesita de cuatro cables en lugar de dos:

- **SCK:** es la línea de reloj generada por el master y enviada a los demás dispositivos.
- **SS:** es la línea utilizada por el maestro para elegir en cada momento con qué dispositivo esclavo se quiere comunicar de entre los varios que puedan estar conectados, ya que solo puede transferir datos a un esclavo a la vez.
- **MOSI:** es la línea utilizada para enviar los datos desde el maestro hacia el esclavo elegido.
- **MISO:** es la contraria al MOSI, puesto que envía los datos del esclavo al maestro.

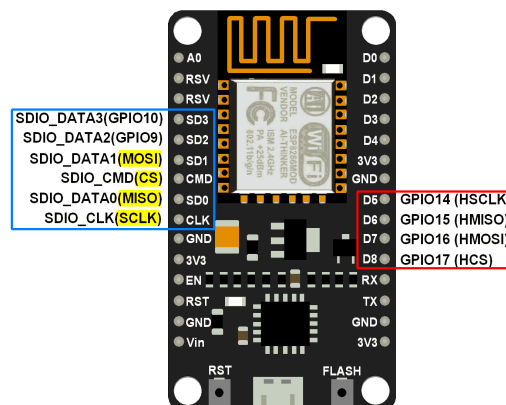


En este diagrama se observa cómo para distintos esclavos se necesita un bus SS distinto. Esto se debe a que el maestro deberá elegir al esclavo cada vez.

Si el bus SS recibe un voltaje BAJO, será el elegido y recibirá los datos correspondientes. Mientras que los demás esclavos recibirán un voltaje ALTO por el bus SS.

Nota: el bus SS también puede ser llamado CS (Chip-Select).

<https://nodemcu.readthedocs.io/en/master/en/modules/spi/>



Detalle de conexión bus SPI 1

2.Aplicando lo aprendido

Una vez entendidos los conocimientos básicos de cada elemento explicado anteriormente, aplicaremos la idea de este proyecto.

2.1 Primera parte: Introducción

La idea de este dispositivo es construirlo de tal manera que sea capaz de monitorizar a tiempo real los datos referidos a una ISFV (Instalación Solar Fotovoltaica), además tendrá que guardar dichos datos en una tarjeta de memoria (microSD) y en un servidor o plataforma para verlos remotamente en cualquier parte del mundo.

Los datos a los que nos referimos son las tres tensiones y las tres corrientes que surgen de los tres circuitos de una ISFV (baterías, carga y paneles).

La plataforma de internet que utilizaremos se llama “thinger.io” y puedes saber más en la sección “Servidor Thinger.io” de este proyecto.

<https://thinger.io/>

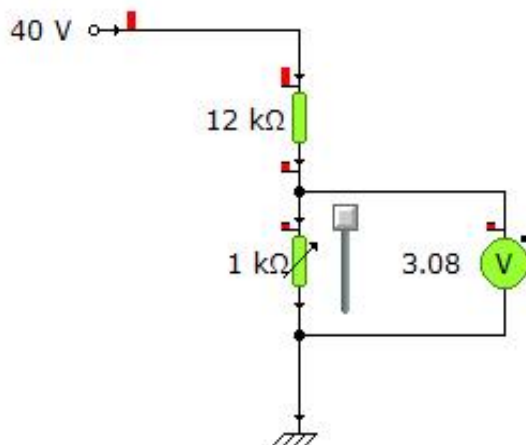
Para medir **las tensiones** utilizaremos un divisor resistivo para reducir la tensión de como máximo 40v a como máximo 3.3v.

Este divisor consistirá en una resistencia fija de valor calculado y una resistencia multivuelta para ajustar la tensión que llega a los pines de entrada.

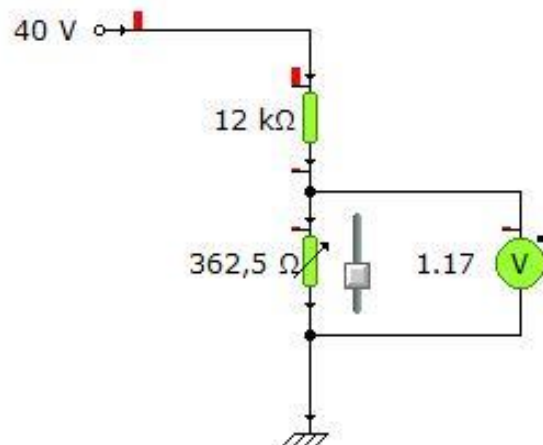
Se ha utilizado la siguiente fórmula para el divisor resistivo:

$$V_{OUT} = \frac{R_2}{R_1 + R_2} \cdot V_{IN}$$

Un divisor resistivo consta como mínimo de dos resistencias en serie y un punto común de la fuente y de la salida:

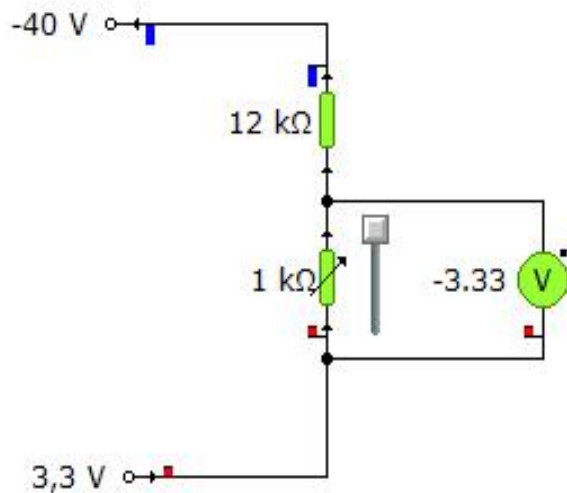


Simulación divisor resistivo 1

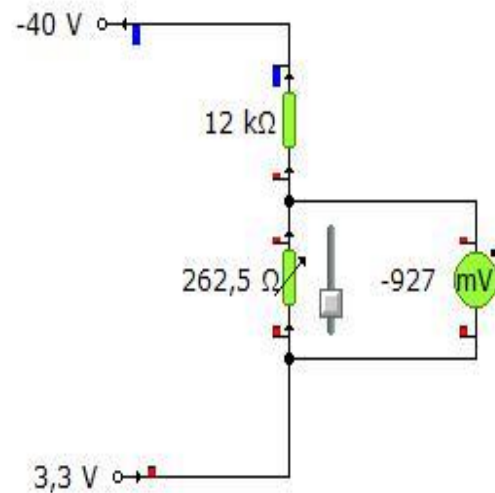


Simulación divisor resistivo 2

En nuestro caso, el punto común de la fuente no es el GND si no que lo es el positivo, por lo que la tensión que mediremos en el divisor no será positiva sino negativa y el punto común serán los 3.3V que nos genera el ESP8266:

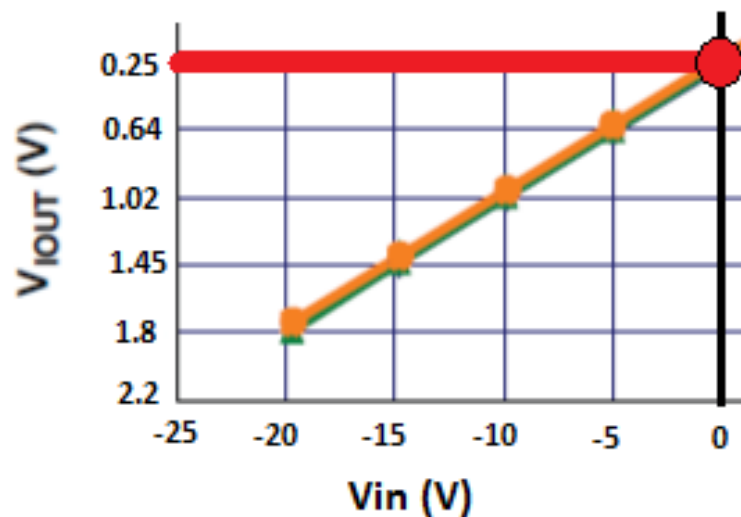


Simulación divisor inverso 1



Simulación divisor inverso 2

La siguiente gráfica muestra el comportamiento de la tensión de salida según la tensión negativa de entrada del regulador:



Gráfica tensión de entrada

2.2 Segunda parte: El código

Empecemos por explicar un poco el código para que en un futuro pueda ser modificado y/o mejorado.

Las bibliotecas de funciones:

```
/* Incluir bibliotecas de funciones */
#include <ESP8266WiFi.h> //Biblioteca de conexión WiFi del módulo ESP8266
#include <ThingyESP8266.h> //Biblioteca de la plataforma thingy.io
#include <Wire.h> //Biblioteca para I2C
#include <RTClib.h> //Para RTC
#include <LiquidCrystal_I2C.h> //Para la lcd
#include <SD.h> //Para la SD
```

Las constantes globales:

```
/* Definir constantes */
#define SSID1 "lupoxan" //SSID Wifi
#define PASS1 "mMuJsKSPz!%Q" //Password Wifi
#define usuario "lupoxan" //Nombre de usuario de thingy
#define device_Id "ESP_8266" //Nombre del dispositivo al que nos conectaremos
#define device_credentials "mMuJsKSPz!%Q" //Credenciales generadas por Thingy
#define Chip_Select_SD D8 //Pin al que se conectará la SD
#define TIME 1000 //Tiempo para mostrar valores
#define MAX 200 //Máximas muestras para hacer media
#define MAXVAL 60 //Maximo de valores que guarda
#define MAXSENS 6 //Maximo de sensores que hay
#define R1 10000 //Resistencia 1 en el divisor resistivo en ohmios
#define R2 1000 //Resistencia 2 en el divisor resistivo en ohmios
#define NOMBRE_ARCHIVO "datalog.csv"
```

Las variables:

```
ThingyESP8266 thing(usuario, device_Id, device_credentials); //Variable para acceder a Thingy

LiquidCrystal_I2C lcd(0x27, 16, 2); //Variable para lcd 0x3F
RTC_DS3231 RTC; //Variable de RTC tipo DS3231 o DS1307 ;OJO!
File dataFile; //Variable para SD

//Variables de conexión WiFi
const char WiFi_ssid[] = SSID1; //Nombre de red
const char WiFi_password[] = PASS1; //Clave de red

//Variables para utilizar el multiplexor
const int muxSIG = A0;
const int muxS0 = D0;
const int muxS1 = D3;
const int muxS2 = D4;
const int muxS3 = D9;

typedef struct { //Por futuras modificaciones
    float valores[MAXVAL]; //Valor de cada sensor
} mediciones; //Para guardar los valores de las mediciones

mediciones vec[MAXSENS]; //Vector de registro tipo mediciones
```

Funciones (cabeceras):

void inline cambio (void) → Esta función es llamada cada vez que el timer interno del ESP8266 sobrepasa los 5S y cambia el valor de “k”.

void setup() → Función de configuración para el IDE de Arduino.

void loop() → Función principal que se repite una y otra vez.

void Output_pins() → Función que define los pines necesarios como salidas.

int SetMuxChannel(byte channel) → Función utilizada para entrar en un canal determinado del multiplexor

Nota: la variable channel adquiere valores entre 0 y 15.

void leer_tarjeta() → Función que inicializa la tarjeta SD y lee los valores que haya en el archivo especificado.

Nota: esta segunda acción está comentada puesto que no es muy necesaria, pero se puede descomentar si se quisiera utilizar.

void escribir_tarjeta(String strValues) → Se utiliza para guardar los valores en la tarjeta SD en el archivo especificado. La variable “strValues” es de tipo String.

float medicion(boolean tipo, int indice) → Se utiliza para realizar las mediciones de tensión y de corriente, según la variable “tipo”.

Nota: si “tipo” es true, medirá corriente, si es false, tensión. La variable “índice” tan solo decide qué canal del multiplexor leer.




void elementos_clave() → Aquí se definen las “resources” para la plataforma Thinger.io.

void call_corrientes_endpoint() → Función para llamar al “endpoint” de la plataforma Thinger.io de nombre Corrientes.

void call_tensiones_endpoint() → Función para llamar al “endpoint” de la plataforma Thinger.io de nombre Tensiones.

void call_potencias_endpoint() → Función para llamar al “endpoint” de la plataforma Thinger.io de nombre Potencias.

Nota: la llamada a estas tres últimas funciones se hace en el void loop() cada vez que la variable “calltime” toma el valor establecido según se requiera.

Endpoint	Type	Description	Created
<input type="checkbox"/> Tensiones	 IFTTT Maker Channel Trigger	Tensiones de la ISFV	2018-05-31 17:35:21 +0200
<input type="checkbox"/> Potencias	 IFTTT Maker Channel Trigger	Potencias de la ISFV	2018-05-31 16:35:36 +0200
<input type="checkbox"/> Corrientes	 IFTTT Maker Channel Trigger	Corrientes de la ISFV	2018-05-31 16:34:52 +0200

Detalle de Endpoints 1

Variables globales:

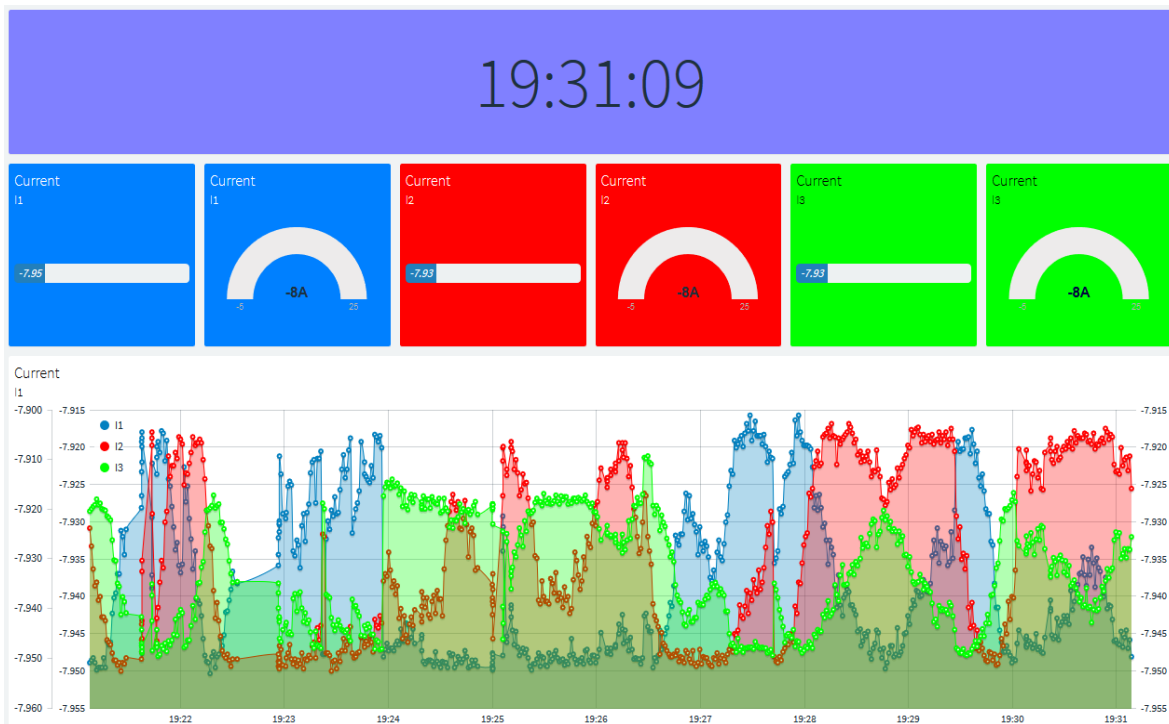
```
//Variables globales
unsigned long presentime;//Para contar el tiempo
unsigned long previoustime = 0;//Para contar el tiempo
int k, i = 0;//Índice para seleccionar los valores de cada sensor
char buffer[120];//Añadir a un String
String strValues;//String que guarda valores
int entero, decimal;//Para separar los valores enteros de los decimales para añadirlos a strValues
int num;//Número de sensor para visualizar en la lcd
char tipo,ud;//tipo: si es V o I; ud: unidades de v, A.
```

Variables para THINGER.IO:

Se ha creado un objeto llamado “*Corrientes_tensiones*” el cual está formado por nueve variables definidas en la función “*elementos_clave*” del código:

1. **I1:** es la corriente medida en el circuito de las placas solares.
2. **I2:** es la corriente medida en el circuito de las baterías.
3. **I3:** es la corriente medida en el circuito de carga.
4. **V1:** es la tensión medida en el circuito de las placas solares.
5. **V2:** es la tensión medida en el circuito de las baterías.
6. **V3:** es la tensión medida en el circuito de carga.
7. **P1:** es la potencia medida en el circuito de las placas solares.
8. **P2:** es la potencia medida en el circuito de las baterías.
9. **P3:** es la potencia medida en el circuito de carga.

Es posible añadir más variables según la sección “Código Arduino para Thinger.io” de este proyecto.



2.3 Tercera parte: las mediciones

Para medir las **tres tensiones**: definida la variable de nombre “*tipo*” de tipo booleana y adquiriendo el valor FALSE entra en el bucle “for do” para recoger “MAX” muestras, así se van sumando para después hacer una media de esos valores.

```
else { //Tensión
    for (int k = 0; k <= MAX - 1; k++) {
        SetMuxChannel(indice);
        value = analogRead(muxSIG) * voltage_v / ADC;
        value = (value - offset_v) * relation_v;
        media = media + value;
    }
}
```

Esta medida se hace para que la medición sea más estable y no haya valores “basura” que puedan interferir en la monitorización de nuestra ISFV.

El cálculo de estas tensiones funciona de la siguiente manera:

1. El ESP8266 posee un convertidor AD de 10 bits, esto significa que las muestras adquieren valores de 0 a 1023. Así convertiremos esos valores a tensión con la siguiente fórmula:

$$V = \frac{V_{\text{dispositivo}} \cdot V_{\text{valor}}}{1023}$$

Tal que:

V: tensión de entrada analógica [V]

V_{dispositivo}: tensión que ofrece el dispositivo [V]

V_{valor}: valor que traduce el convertidor ADC [0 – 1023]

2. Para continuar debemos escalar esos valores de tensión a la escala que necesitemos, así, en nuestro caso como nuestra tensión de entrada es negativa, debemos de configurar un “offset” negativo que obtendrá el valor del corte con el eje de ordenadas según la recta:

$$V_{in} = m \cdot V_{out} + offset$$

Tal que:

V_{in}: es el valor de tensión del sensor.

V_{out}: es el valor de tensión de entrada del sensor.

m: es la pendiente de la recta que forma el sensor.

Offset: es el valor que se obtiene cuando V_{in} es cero.

Estos valores deberán ser de tipo **float** puesto que queremos medir valores con decimales.

Para medir las **tres corrientes**: la variable “tipo” adquiere ahora un valor TRUE y funciona exactamente igual que la medida de las tensiones.

Nota: no hay que olvidar que primero se debe calibrar el sensor según la página web del sensor ACS712.

3. Servidor Thingier.io

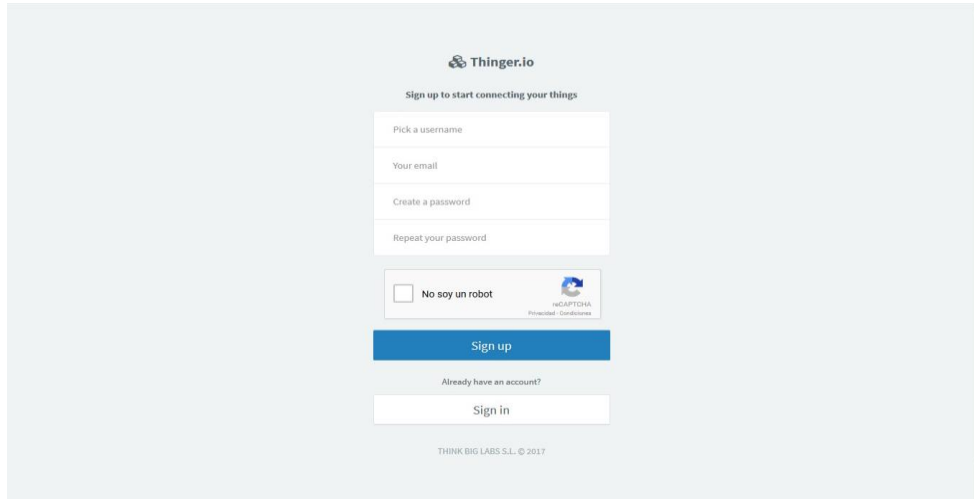
3.1 ¿Qué es Thingier.io?

Es un servidor que nos dota de la posibilidad de controlar nuestros dispositivos IOT'S (Internet Of Things) de una manera simple y cómoda.

Primero hemos de crearnos una cuenta y para ello nos meteremos en:

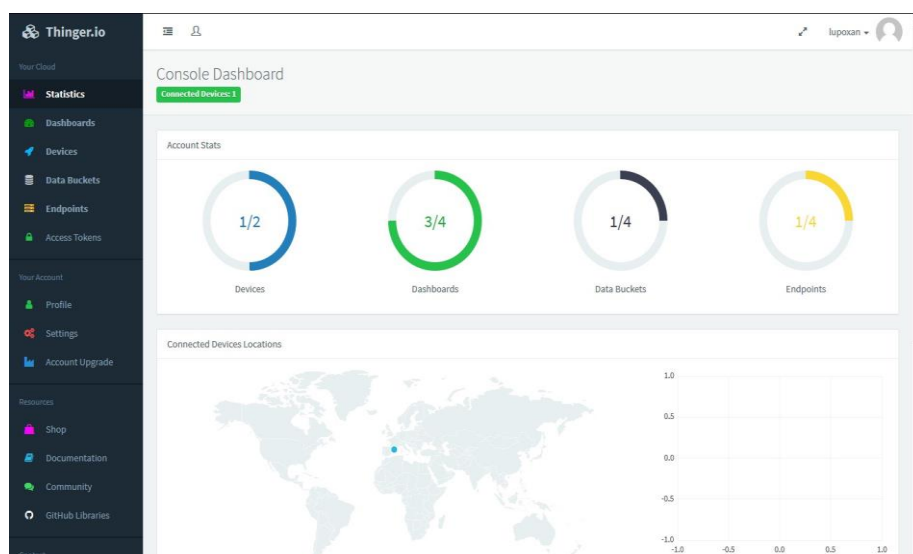
- <https://console.thingier.io/#/signup>

Allí nos saldrá una página como esta:



- 1- Elegiremos un username o Nick el cual usaremos en nuestro código de Arduino en la línea de usuario (#define usuario “usuario_thingier”).
- 2- Introduciremos nuestro E-mail al que nos llegará un correo de activación para activar nuestra cuenta en Thingier.
- 3- Elegiremos una contraseña la cual usaremos para entrar en nuestra cuenta desde un navegador.
- 4- Verificamos que nos somos robots y clicamos en “Sign up”.

Una vez dentro de nuestra cuenta nos saldrá una página parecida a la siguiente:

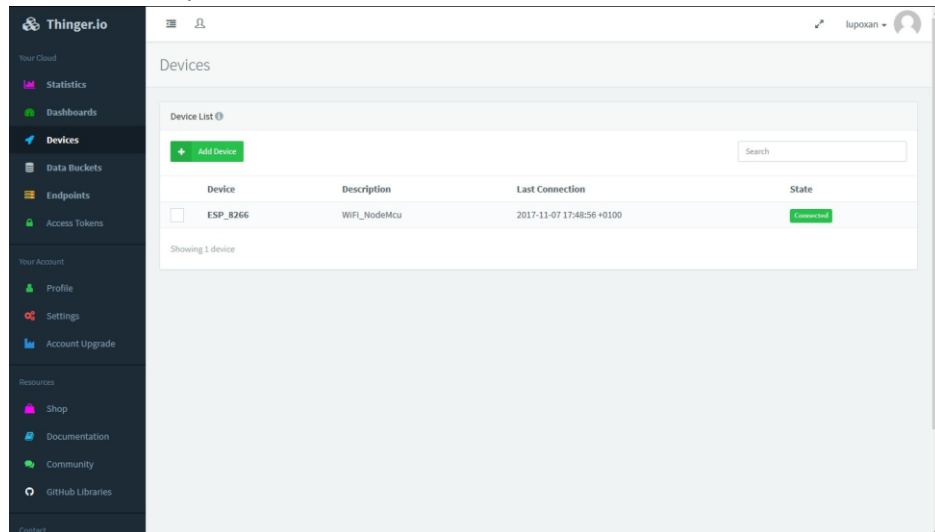


3.2 Agregar un dispositivo o “device”

En nuestro caso será la placa ESP8266 (puede ser cualquier otra en cualquier modelo).

Para ello:

- 1- Clicamos en “Devices” tanto en el “Donut” como en el índice izquierdo.
- 2- Nos saldrá una pantalla como esta, clicamos en “Add device”



- 3- Nos saldrá una página como esta y rellenamos los campos:

- a. Device Id: aquí escribiremos el nombre que queramos para nuestro dispositivo. Más tarde incluiremos este nombre en el código de nuestro Arduino (`#define device_Id "nombre_dispositivo"`).
 - b. Device description: Aquí incluiremos una pequeña descripción de nuestro proyecto IOT o de nuestra placa (lo que queramos).
 - c. Device credentials: Es nuestra “contraseña” para conectarnos con este dispositivo en concreto puesto que Thingier nos permite tener más de un dispositivo activo. Usaremos estas credenciales en nuestro código de Arduino (`#define device_credentials "credenciales"`).
- 4- Nos saldrá una notificación: (Done! You can now connect your new device! More info here, or go to the dashboard!)
 - 5- Pues listo, nuestro dispositivo está preparado en el servidor Thingier.

Como aún no hemos conectado nuestro dispositivo ni le hemos metido el código fuente de Arduino, nos saldrá como “disconnected” y en rojo.

Antes de configurar nuestro código Arduino, mencionar que este servidor tiene “limitaciones”:

- 1- Número máximo de dispositivos: 2.
- 2- Número máximo de “dashboards”: 4.
- 3- Número máximo de “data buckets”: 4

Si quisiéramos ampliar estas limitaciones habremos de pagar una cantidad establecida en la página de thinger.io. En el índice izquierdo en “Account Upgrade”.

3.3 Aclaraciones

- 1- Un dashboard se traduce como “tablero”. Es donde visualizaremos todos nuestros datos, gráficas y donde podremos actuar sobre las salidas o entradas de nuestro dispositivo. Se puede configurar al gusto del consumidor.
- 2- Un “data bucket” se traduce como cubo de datos, es donde se almacenan todos los datos a lo largo del tiempo (máx. 1 año en cuentas Free). Como somos cuenta Free, cada dato se almacenará pasado 1 min.

3.4 Dashboard

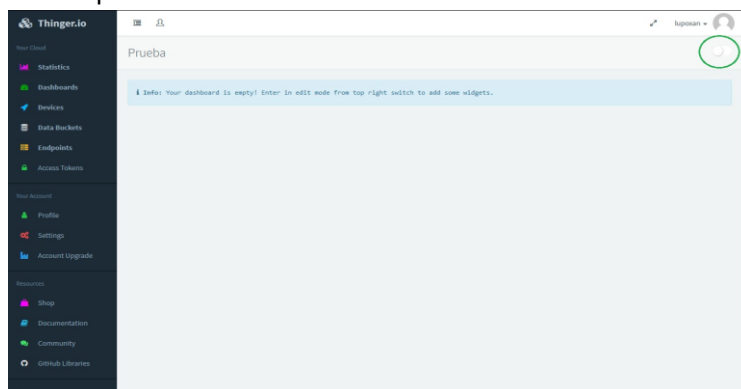
Hay que tener en cuenta que todos los datos reflejados en un tablero como éste son volátiles, es decir que cuando se sale de un dashboard, todos los datos se perderán.

3.4.1 Añadir un Dashboard:

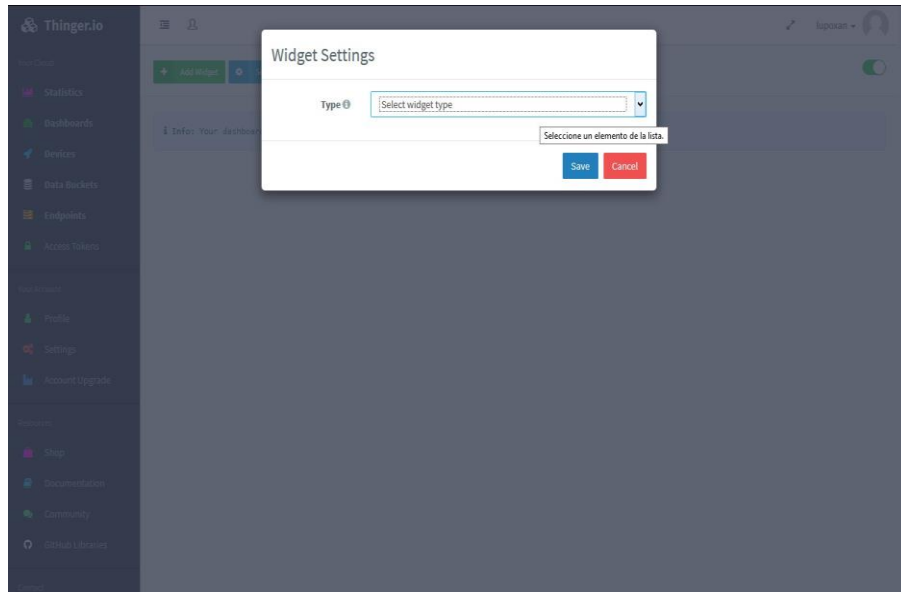
- 1- Clicar en el índice izquierdo en “Dashboard” y clicar en “Add dashboard”.
- 2- Rellenar los campos:
 - a. Dashboard ID: Nombre al que nos referimos cuando hablemos de nuestro Dashboard. Es como si fuera una dirección propia.
 - b. Dashboard name: nombre que queramos ponerle a nuestro tablero.
 - c. Dashboard description: una breve descripción de nuestro dashboard.
- 3- Clicar en “Add dashboard” y listo, ya podemos irnos de nuevo a la opción “dashboard” y allí lo tendremos listo para ser usado.

3.4.2 Usar un Dashboard:

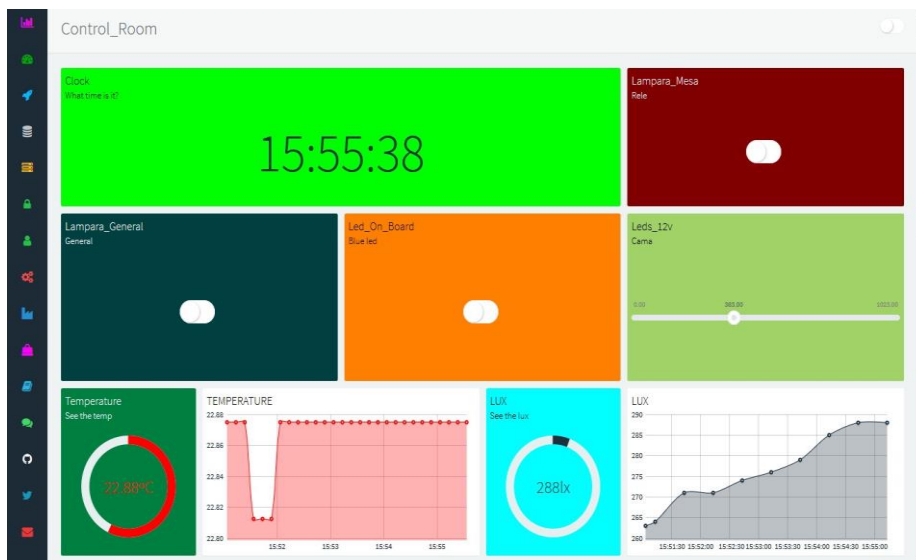
- 1- Para ello clicaremos en el dashboard que queramos editar o usar y nos saldrá una ventana como la siguiente, donde le daremos al botón que aparece arriba a la derecha para entrar en modo edición:



- 2- Si clicamos sobre él, activamos el modo edición y aparecerá a la izquierda dos recuadros uno con “Add widgets” y el otro con “settings”. Clicamos en añadir widgets.



- 3- Hay varios tipos de widget:
- a. De tipo display: son los que muestran datos recogidos por nuestro Arduino así como Temperatura, humedad, luz, corriente, estados de las salidas, etc.
 - b. De tipo Device Control: son los que pueden modificar con el estado de las salidas de nuestro dispositivo.
- 4- Así nos debería quedar nuestro Dashboard:



3.4.3 Configuraciones de un DashBoard.

Se hablará de cómo configurar todos y cada uno de nuestros widgets, de los “stream resources” y de cómo compartir nuestro dashboard con los demás.

Este tipo de tablero se dice que es volátil puesto que los valores se pierden si se cierra. Para eso guardamos los valores que nos interesen en el data bucket.

El botón de arriba a la derecha lo activaremos cada vez que queramos entrar en modo edición donde nos saldrán los botones “Add widgets” y “Settings”. Lo desactivaremos para salir.

3.4.3.1 Widgets tipo display.

- 1- **Times Series chart.** Se define como una gráfica de puntos normal y corriente. Si la seleccionamos nos encontraremos con campos para rellenar como:
 - a. Title: es el título que queramos poner a nuestra gráfica.
 - b. Subtitle: es un subtítulo que queramos poner a nuestra gráfica.
 - c. Background: es el color que queramos que nuestra gráfica tenga de fondo.
 - d. Chart input: es para seleccionar de dónde vendrán todos los datos que pongamos en nuestra gráfica:
 - i. From device: vendrá de nuestro dispositivo Arduino.
 - ii. From bucket: vendrá de nuestro cubo de datos.
 - e. Time period: periodo de tiempo que se quiera representar en la gráfica.
 - f. Options: axis (ejes), legend (leyenda), fill splines (relleno de la curva) y multiple axes (ejes múltiples).
 - g. Chart color: para elegir el color de la curva representada.
- 2- **Donut chart.** Es similar que el times series chart pero en forma de donut y con la diferencia que nos pide las unidades del valor a representar, el valor mínimo y máximo que puede tomar ese dato. Solo representa un valor del dato a la vez.
- 3- **Progressbar.** Igual que el donut chart pero en forma de barra.
- 4- **Gauge.** Igual que el donut chart pero en forma de semicírculo.
- 5- **Google Map.** Se representa la localización de:
 - a. Fixed location: introduciremos la longitud y la latitud que queramos.
 - b. Current location: donde está situado nuestro dispositivo.
 - c. From device: desde nuestro dispositivo si tuviéramos GPS.
 - d. From bucket: desde nuestro cubo de datos si guardáramos la longitud y la latitud en un data bucket.
- 6- **Image/MJPEG:** aquí podemos añadir una URL que destine a una foto y así visualizarla en nuestro dashboard.
- 7- **Text/Value:** si elegimos “manual” podremos escribir un texto nosotros mismos, pero si elegimos “from device” o “from bucket” se visualizará el valor que tome la variable elegida.

8- Clock: para visualizar la hora. Hay opción de UTC (universal time clock).

Nota 1: si seleccionamos “from device” se añadirá un campo que nos pedirá el nombre de nuestro dispositivo, después nos pedirá el nombre de las variables que hayamos creado como salida en nuestro código de Arduino (temperatura, humedad, luz, etc). Así como el tiempo que tarde en mostrar cada dato (mínimo 1 segundo) en el campo de “sampling interval”.

Nota 2: si seleccionamos “from bucket” se añadirá un campo que nos pedirá el nombre del bucket que hayamos creado anteriormente, así como las variables a representar en el gráfico. En el campo de “timeframe” si seleccionamos absolutos, nos pedirá una fecha en concreta para representar, pero si marcamos relativos, nos pedirá el periodo último que se quiera representar, así haremos una gráfica a tiempo real de los últimos 4 días por ejemplo.

Nota 3: si en el campo de “Refresh mode” seleccionamos “Update by device” en nuestro código de Arduino habremos de utilizar la función “thing.stream(thing[“temp_lux”]);”, para poder utilizar el objeto “temp_lux” en modo stream.

3.4.3.2 Widgets tipo device control:

- 1- **On/Off state:** nos pedirá título y subtítulo, nombre del dispositivo y nombre de la variable que configuramos como entrada en nuestro código de Arduino. Nos saldrá un botón para encender o apagar a nuestro gusto.
- 2- **Slider:** lo mismo que On/Off state pero debemos de añadir un valor mínimo y un valor máximo para nuestra salida de Arduino. Ojo, para las entradas PWM el valor máximo es de 1023 y no de 255. Esta función solo es para las salidas PWM.

3.5 Modo Stream

El modo Stream es como cuando se mira una película por Internet sin descargarla. Pues aquí lo mismo, se visualizan los datos a través del servidor a tiempo mayor del que nos permite en las cuentas Free (1 minuto).

Para ello en nuestro código fuente de Arduino, debemos de añadir en la línea oportuna la función “thing.stream(thing[“temp_lux”]);” y en nuestro dashboard seleccionar en el campo de “Refresh mode”, “update by device”.

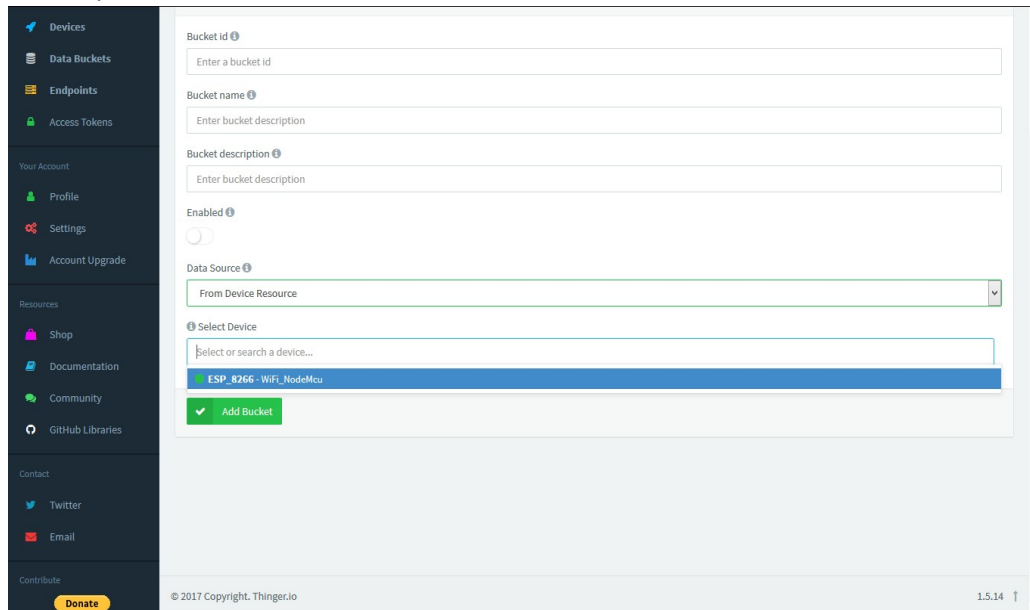
3.6 Compartir nuestro Dashboard

Para compartir nuestro dashboard debemos de entrar en “settings” y activar el campo “share”, esto generará un link que podemos compartir con los demás.

3.7 Data Bucket:

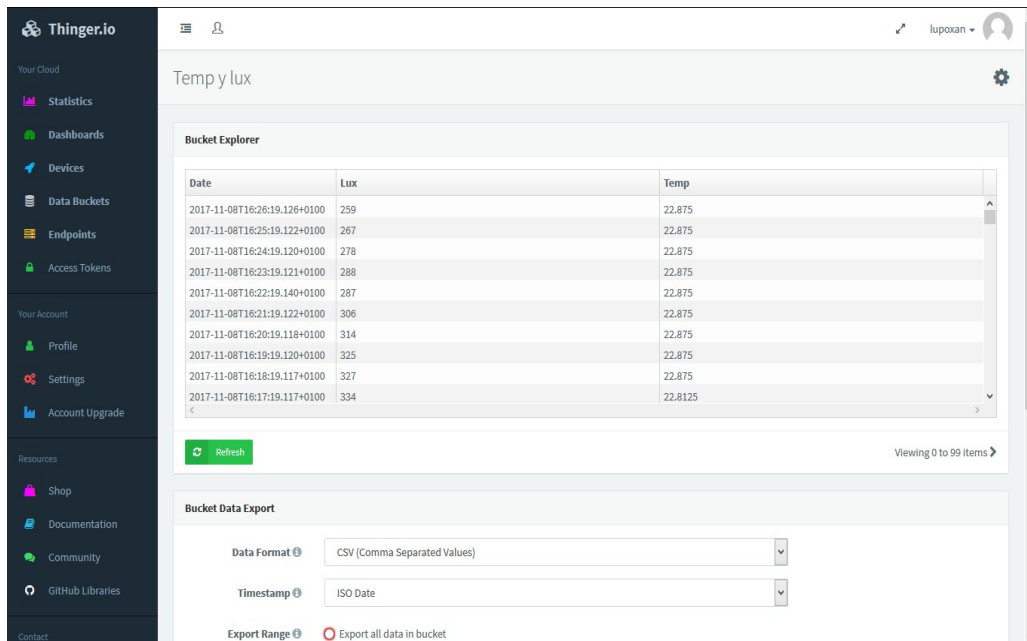
Nota: Si no hemos subido ningún código a nuestra placa Arduino, no podremos configurar esta sección.

- 1- En el índice izquierdo, clicamos sobre “Data Buckets” y le damos a “Add Bucket”, así podremos añadir nuestra cesta de datos.
- 2- Rellenamos los campos como en el dashboard, solo que esta vez tendremos que fijar de dónde vienen los datos. Normalmente vendrán desde nuestro dispositivo. Entonces en el campo de “Data source” o fuente del dato seleccionaremos “From device resource” o desde nuestro dispositivo.
- 3- Se añadirá un nuevo campo, el de “Select Device” para seleccionar nuestro dispositivo:



- 4- Una vez seleccionado nuestro dispositivo se volverá a añadir un nuevo campo (select resource) en el cual podremos seleccionar los datos que sean salida del dispositivo hacia Tinker como la temperatura, humedad, etc.
- 5- Se añadirá el último campo a rellenar que es para seleccionar el tiempo que tarda en guardar cada dato (1 min como mínimo en nuestro caso). Para ello:
 - a. Seleccionar “sampling interval” – 1 minutes
- 6- Antes de añadir, hemos de habilitar esta “data bucket”. Para ello clicamos sobre el botón de “enabled” que se pondrá en verde.

- 7- Ahora si, clicamos sobre “Add bucket”. Nos saldrá un mensaje como este:
“Done! You can now explore your bucket! Go to your new bucket!”
- 8- Debería quedarnos algo parecido a esto:



The screenshot shows the Thingier.io web interface. On the left is a dark sidebar with navigation links: Your Cloud, Statistics, Dashboards, Devices, Data Buckets, Endpoints, Access Tokens, Your Account, Profile, Settings, Account Upgrade, Resources, Shop, Documentation, Community, GitHub Libraries, and Contact. The main area is titled 'Temp y lux' and contains a 'Bucket Explorer' section. It displays a table with three columns: Date, Lux, and Temp. The table lists 10 data points with timestamps from 2017-11-08T16:26:19.126+0100 to 2017-11-08T16:17:19.117+0100, Lux values ranging from 259 to 334, and Temp values around 22.875. Below the table is a 'Refresh' button and a 'Viewing 0 to 99 items' indicator. At the bottom is a 'Bucket Data Export' section with dropdowns for 'Data Format' (set to CSV) and 'Timestamp' (set to ISO Date), and a radio button for 'Export Range' (set to 'Export all data in bucket').

Date	Lux	Temp
2017-11-08T16:26:19.126+0100	259	22.875
2017-11-08T16:25:19.122+0100	267	22.875
2017-11-08T16:24:19.120+0100	278	22.875
2017-11-08T16:23:19.121+0100	288	22.875
2017-11-08T16:22:19.140+0100	287	22.875
2017-11-08T16:21:19.122+0100	306	22.875
2017-11-08T16:20:19.118+0100	314	22.875
2017-11-08T16:19:19.120+0100	325	22.875
2017-11-08T16:18:19.117+0100	327	22.875
2017-11-08T16:17:19.117+0100	334	22.8125

Cuando seleccionamos nuestro data bucket lo primero que nos encontramos es el Bucket explorer que es donde se visualizan todos los datos y valores establecidos cuando creamos nuestro cubo de datos.

En este explorador, nos encontramos la fecha y la hora en una columna a la izquierda con el formato: AAAA – MM – DDTHH:MM:SS.sss+0100.

Al lado de la fecha y la hora nos encontramos todos los datos que tengamos en nuestro “resource”. Sólo se enseñan los últimos 99 datos, aún no entiendo por qué.

Para visualizar todos los datos que queramos deberemos o exportarlos como .csv, .arff o como .json. También podremos crear en un dashboard una gráfica que nos muestre todos los datos desde el principio (mucho más cómodo).

Nota: al ser una cuenta Free los datos que se guardan son del último año.

Podremos exportar los datos en un rango de fechas o podremos exportar todos los datos desde el principio. Recibiremos un e-mail con un link que nos redirigirá a nuestro archivo para descargarlo.

También podremos borrar datos en un rango de fecha determinada o todos los datos desde el principio. Se recibirá un correo con la confirmación de dicha eliminación.

4. Manejo de datos guardados

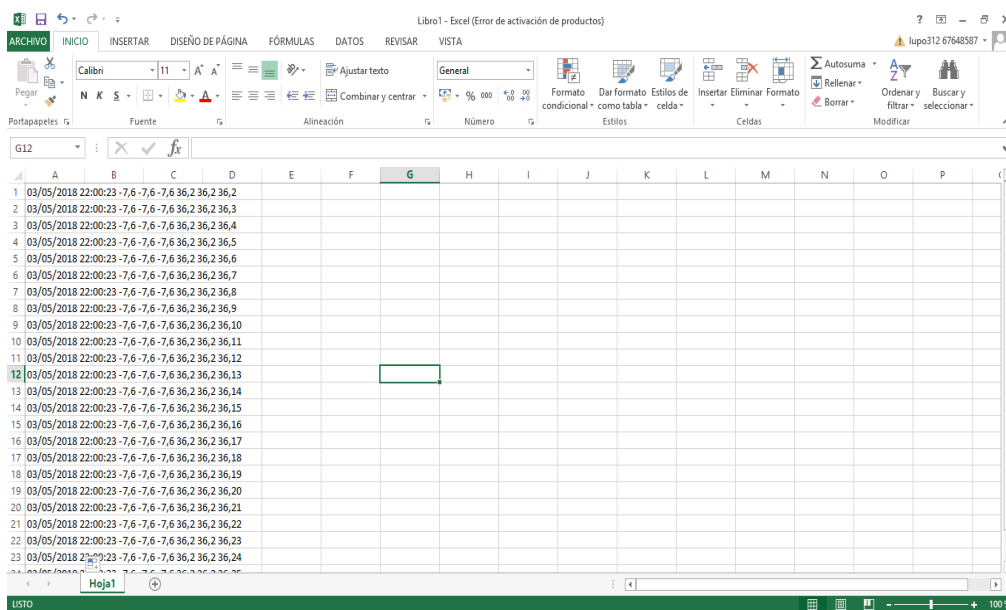
Aquí aprenderemos un poco a manejar esos datos guardados, tanto en la tarjeta microSD como en la plataforma Thinger.io.

4.1 MicroSD

Cuando queramos ver los datos que se han guardado en la microSD, deberemos de apagar el sistema o al menos, reiniciar cuando la volvamos a insertar en el zócalo.

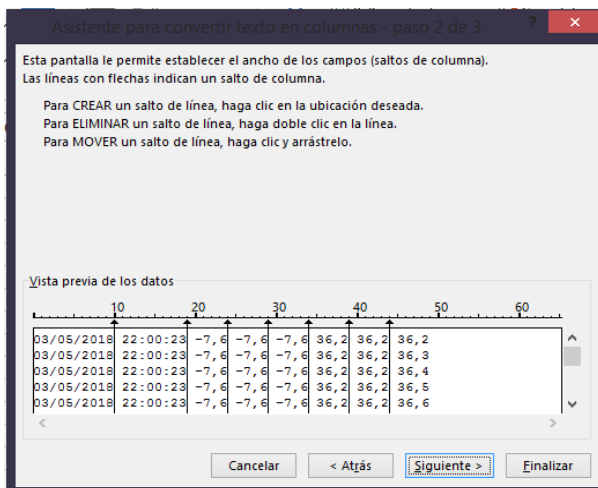
Así pues, nos dispondremos a insertar la tarjeta en el ordenador y cuando se abra nos encontraremos un archivo llamado “DATALOG.CSV”. Es el que debemos de abrir para observar los datos recogidos.

Una vez abierto veremos que los datos estarán todos mezclados en una misma columna:



Seleccionaremos la primera columna que es donde tenemos los datos y en la pestaña “datos” clicaremos sobre el botón “texto en columnas”.

Dejamos seleccionada la opción “de ancho fijo” y damos en siguiente para visualizar los datos:



Daremos a siguiente y nos mostrará una pantalla para elegir el tipo de dato que es cada columna. Nosotros dejamos todos los datos en “general”. Hacemos clic en finalizar y ya tendremos todos los datos separados por columna.

Hacer notar que estos datos se separan mediante tabulaciones, así si surge algún problema, seleccionaremos tabulación como separador.

4.2 Plataforma THINGER.IO

Los datos que obtengamos en Thinger se hará según usuario, es decir, a través de un dashboard o a través de un data bucket.

En el dashboard se pueden visualizar todos los datos que tengamos con lo diferentes Gadgets y en el data bucket solo se obtienen cada minuto todos los datos que hayamos definido en una lista, es decir como en una base de datos.

El manejo de estos datos vienen explicados en el “manual de usuario para Thinger.io” anexo a este proyecto.

Recomendamos que para la visualización de datos sea a través de un dashboard, más concretamente a través de un “time series chart” cuya fuente de datos sea un data bucket.

Cuando queramos exportar nuestros datos de esta plataforma, se recomienda que sea a través de un .CSV puesto que es lo más común.

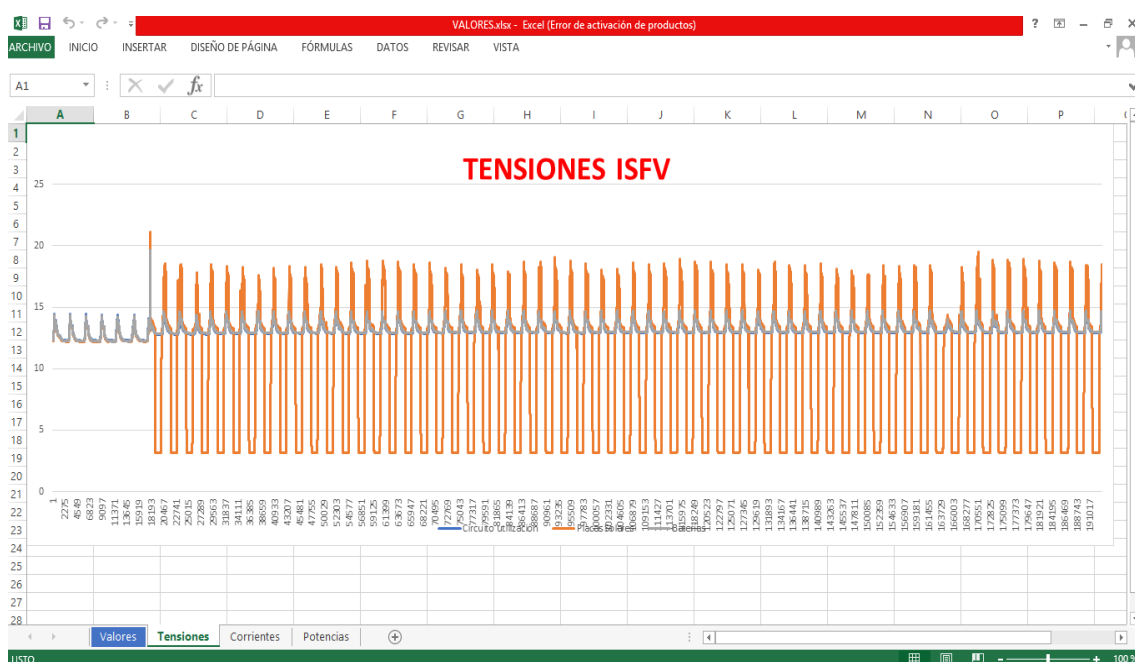
Una vez exportado nuestro .CSV recordar que nos enviarán un correo con el enlace para descargarlo.

Cuando lo abramos no necesitaremos agrupar los datos en columnas puesto que ya estarán colocados y observaremos que en la primera fila tendremos los nombres de las variables que habíamos ido subiendo.

En la primera columna aparecerá “ts” que proviene de “TimeStamp” es la columna de la fecha y la hora.

En nuestro caso, las demás columnas serán de I1, I2, I3, P1, P2, P3, V1, V2 y V3.

Así ya estaremos listos para generar gráficas, estadísticas, etc.



5. Aplicación para Android

5.1 Descargando la app

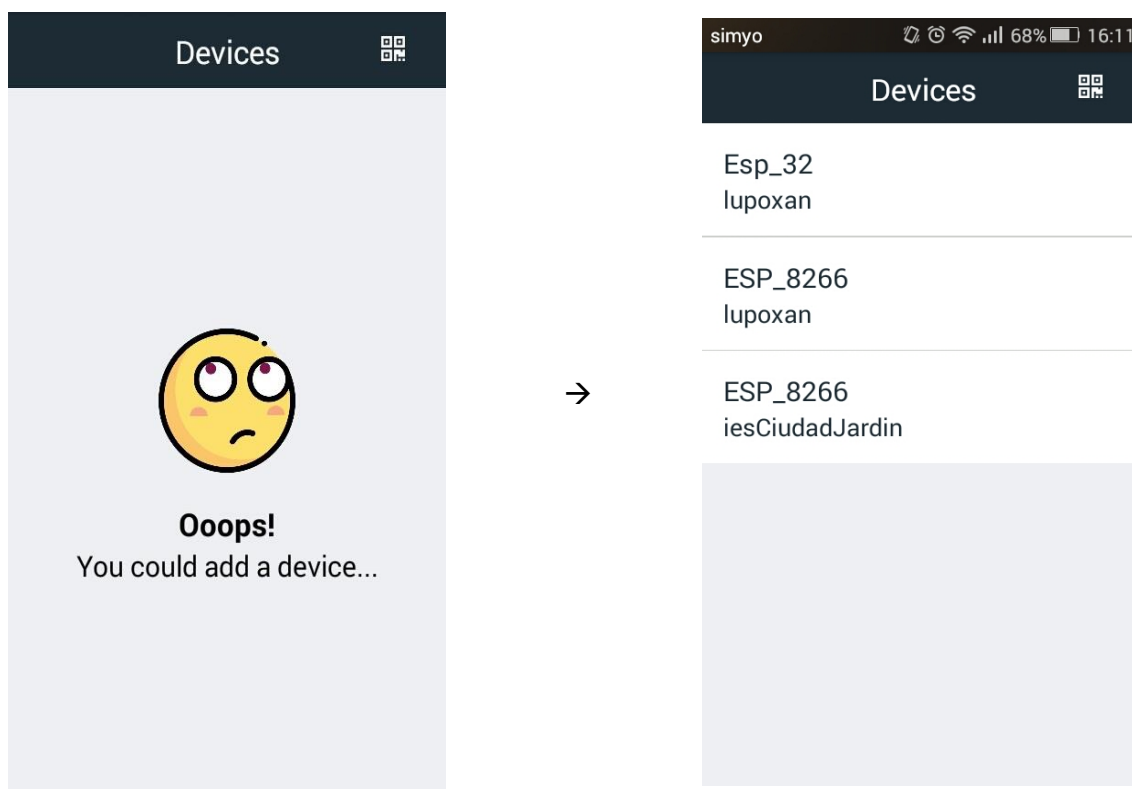
En Play Store, buscaremos la aplicación thinger.io y la instalaremos en nuestro teléfono móvil y cuando ya esté instalada la abriremos. Nos aparecerá la imagen siguiente puesto que no hemos añadido aún ningún dispositivo de nuestra cuenta.

Nota: se pueden añadir dispositivos de distintos usuarios.

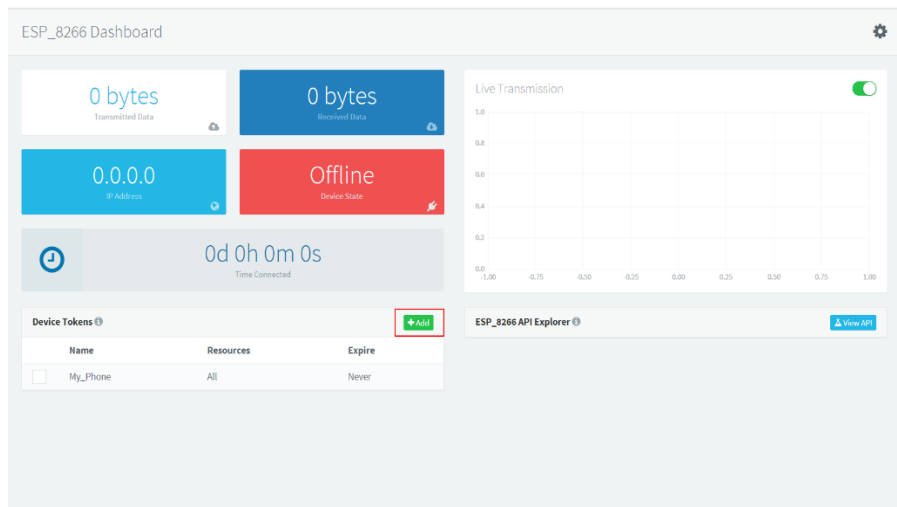
Para añadir un dispositivo nuevo tan solo hemos de clicar en el icono del código QR situado en la esquina superior derecha.

Así escanearemos en la página de Thinger.io dicho código y ya tendremos listo nuestro dispositivo.

En grande, sería el nombre del que nos hemos conectado y en pequeño el nombre del usuario que posee ese dispositivo.



Nota: para poder acceder al código QR antes debemos de generar un permiso en la plataforma de nuestro dispositivo. Para ello, en la zona “devices” clicaremos en el dispositivo que nos interese y en “Devices tokens” daremos en “add”:



Así nos saldrá una pantalla como la siguiente:

Add Device Token

Token Name ⓘ

Token Access ⓘ

☒ Allow accessing all device resources with this token

☐ Limit the resource access for this token

Token Expiration ⓘ

☒ This token will never expire

☐ Limit the token lifetime on some date

Cancel OK

Pondremos un nombre cualquiera, normalmente será el del dispositivo móvil o Tablet que tendrá el acceso a nuestro “device”.

Posee dos opciones:

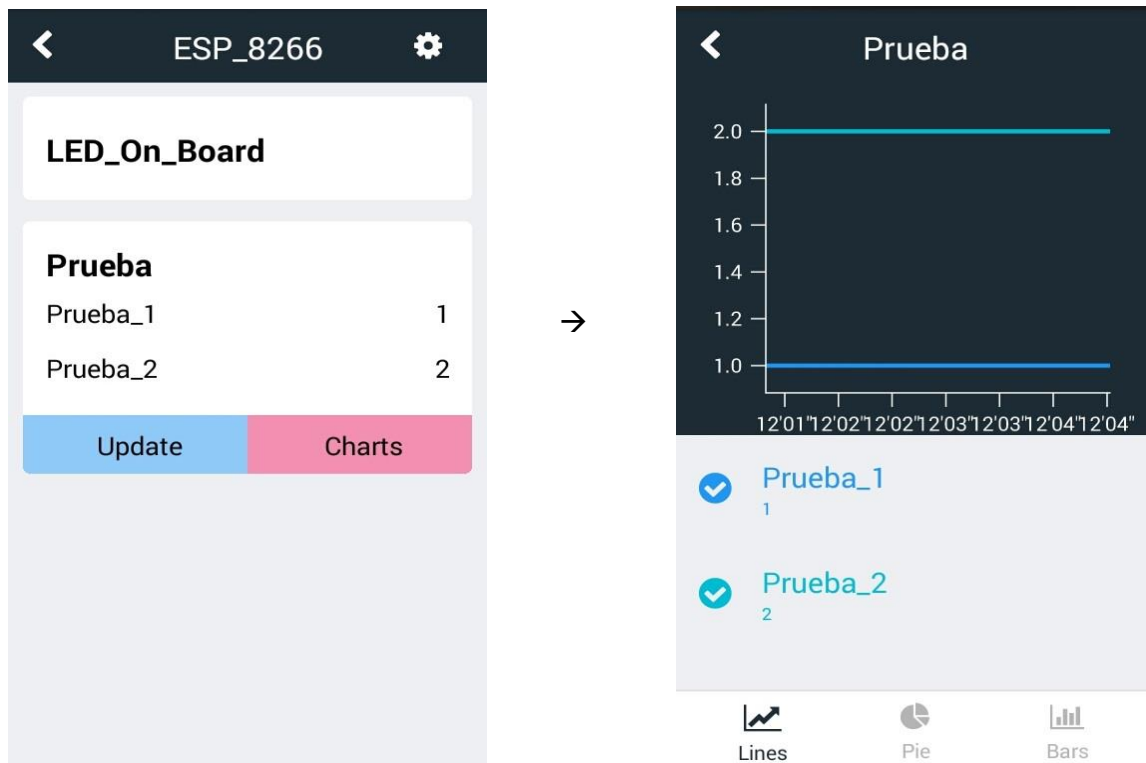
1. **Token Access:** el cual permite al móvil o a la Tablet acceder a las distintas variables que tenga nuestro dispositivo. Podemos limitar el acceso a todas las variables o a por ejemplo una sola.
2. **Token Expiration:** permite controlar el tiempo que tiene acceso nuestro móvil o Tablet al dispositivo conectado. Podemos configurarlo para que nunca expire y siempre tenga acceso, pero podemos poner una fecha y una hora en la que termina ese permiso.

Así ya daremos en ok y nos aparecerá el “device token” con una contraseña muy larga, esto es para desarrolladores. Nosotros clicaremos en “Show QR Code” y lo escanearemos como anteriormente vimos.

Y ya podremos cerrar y tendremos en nuestro móvil o Tablet el dispositivo que queremos visualizar sin necesidad de abrir un navegador web en el ordenador.

5.2 Probando la aplicación

Cuando ya tengamos nuestro dispositivo en el móvil o Tablet, tendremos acceso a él para visualizar todas las “resources” que tenga. Para ello tan solo debemos clicar en él:



Si pinchamos en “update” se refrescarán las variables que tengamos. En este caso, Prueba_1 y Prueba_2.

Pero lo interesante es verlo en una gráfica que nos muestre en tiempo real el transcurso de estas variables. Para ello tan solo tendremos que pinchar en “charts” y se abrirá otra pantalla en la que aparecerá una gráfica a tiempo real.

Se estará refrescando cada segundo.

Podremos elegir también qué variables visualizar al mismo tiempo tan solo pinchando en el checkbox que aparece al lado del nombre de cada variable.

En la parte inferior nos da opción al modo de visualización, es decir, en modo líneas, a modo de tarta o en modo de barras.

Nota: Si nuestro dispositivo está desconectado, cuando pinchemos en él, nos mostrará un mensaje de error o si nuestro acceso ha sido restringido.

Podemos ver la configuración de nuestro dispositivo en el móvil o en la Tablet pinchando en la rueda de “Settings”. Ahí nos aparecerá el nombre del dispositivo, el usuario, el servidor al que está conectado (<https://api.thinger.io>), el Token QR, la fecha de creación del acceso y la fecha de expiración del acceso.

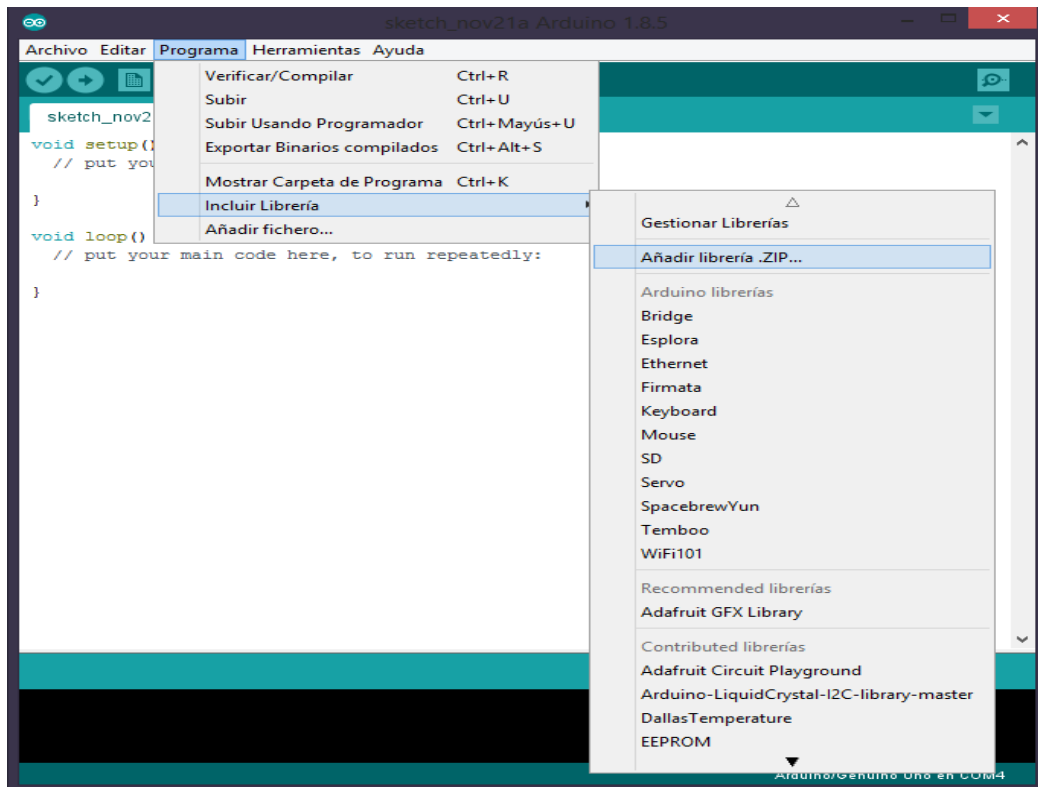
También podemos borrar el dispositivo en la configuración (El botón rojo “remove”).

Nota: el servidor al que está conectado puede ser editado pinchando en él, pero es para desarrolladores y nosotros lo dejaremos tal y como está.

6. Código Arduino para Thinger.io

6.1 Biblioteca de funciones para Thinger.io:

- En esta página encontraremos la información necesaria para descargar la biblioteca.
- <http://docs.thinger.io/arduino/#installation-manual-import>
- Clicamos en “Download library”.
- En nuestro IDE de Arduino, importaremos dicha biblioteca:



- Reiniciamos nuestro IDE de Arduino y veremos que se ha incluido en nuestras bibliotecas como “thinger.io”
- Sólo nos interesará las relacionadas con el “esp8266” y el “esp32”.

Nota: para el dispositivo esp8266 es necesaria la biblioteca “ESP8266WiFi.h” y para el esp32 solo es necesaria “WiFi.h”

Antes de programar nada:

1- Incluir biblioteca:

a. ESP8266:

- #include<ESP8266WiFi.h>
- #include <ThingerESP8266.h>
- Más las que necesitemos (RTCLIB, Wire, etc).

b. ESP32:

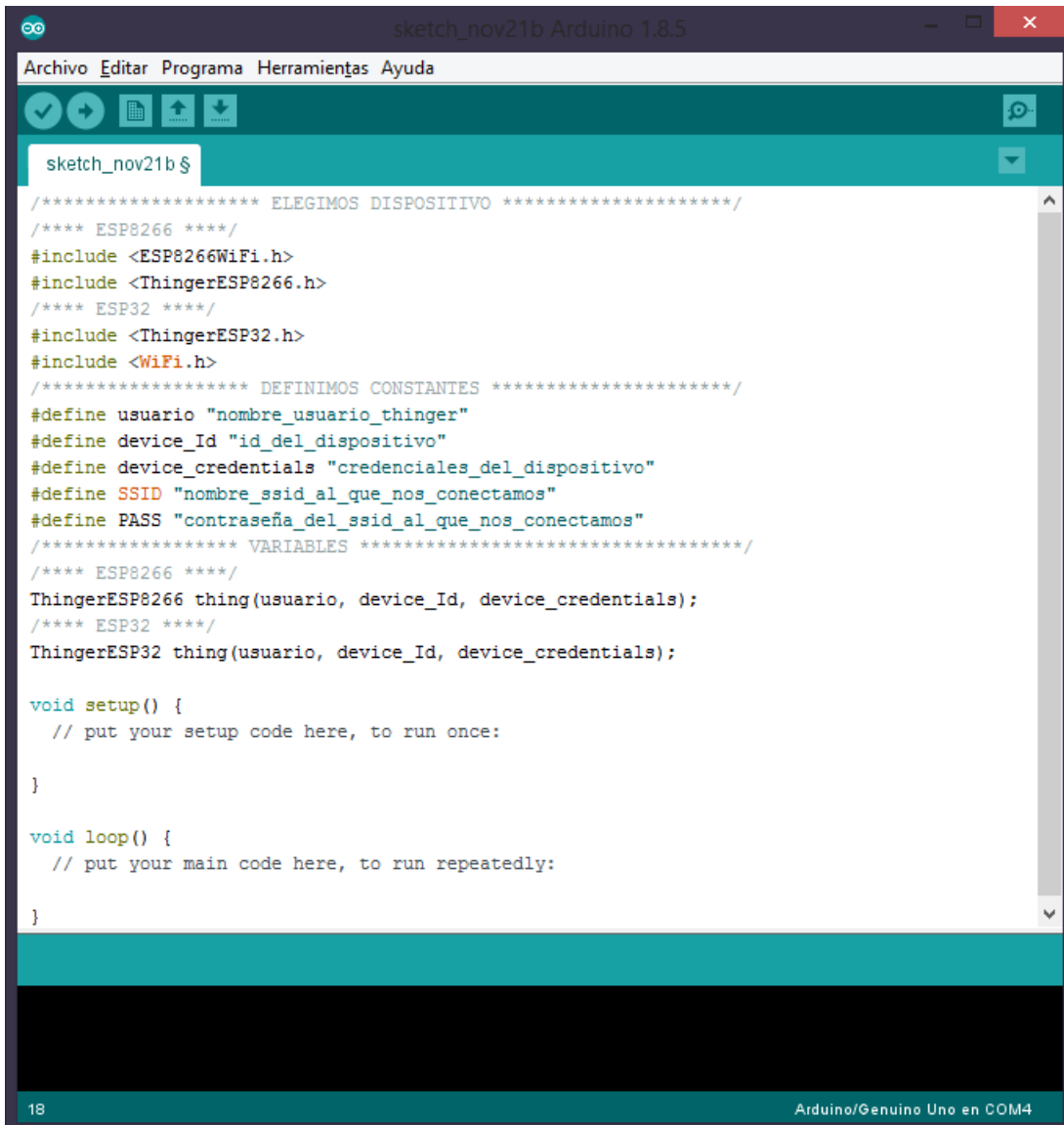
- #include <WiFi.h>
- #include <ThingerESP32.h>
- Más las que necesitemos (RTCLIB, Wire, etc).

2- Definir constantes:

- a. #define usuario "nombre_usuario_thinger"
- b. #define device_Id "id_del_dispositivo"
- c. #define device_credentials "credenciales_del_dispositivo"
- d. #define SSID "nombre_ssid_al_que_nos_conectamos"
- e. #define PASS "contraseña_del_ssid_al_que_nos_conectamos"
- f. Más las que necesitemos.

3- Definir variables:

- a. **ESP8266:**
 - i. ThingerESP8266 thing(usuario, device_Id, device_credentials);
- b. **ESP32:**
 - i. ThingerESP32 thing(usuario, device_Id, device_credentials);



```
sketch_nov21b Arduino 1.8.5
Archivo Editar Programa Herramientas Ayuda
sketch_nov21b $
/***** ELEGIMOS DISPOSITIVO *****/
/**** ESP8266 ****/
#include <ESP8266WiFi.h>
#include <ThingerESP8266.h>
/**** ESP32 ****/
#include <ThingerESP32.h>
#include <WiFi.h>
/***** DEFINIMOS CONSTANTES *****/
#define usuario "nombre_usuario_thinger"
#define device_Id "id_del_dispositivo"
#define device_credentials "credenciales_del_dispositivo"
#define SSID "nombre_ssid_al_que_nos_conectamos"
#define PASS "contraseña_del_ssid_al_que_nos_conectamos"
/***** VARIABLES *****/
/**** ESP8266 ****/
ThingerESP8266 thing(usuario, device_Id, device_credentials);
/**** ESP32 ****/
ThingerESP32 thing(usuario, device_Id, device_credentials);

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

18 Arduino/Genuino Uno en COM4
```

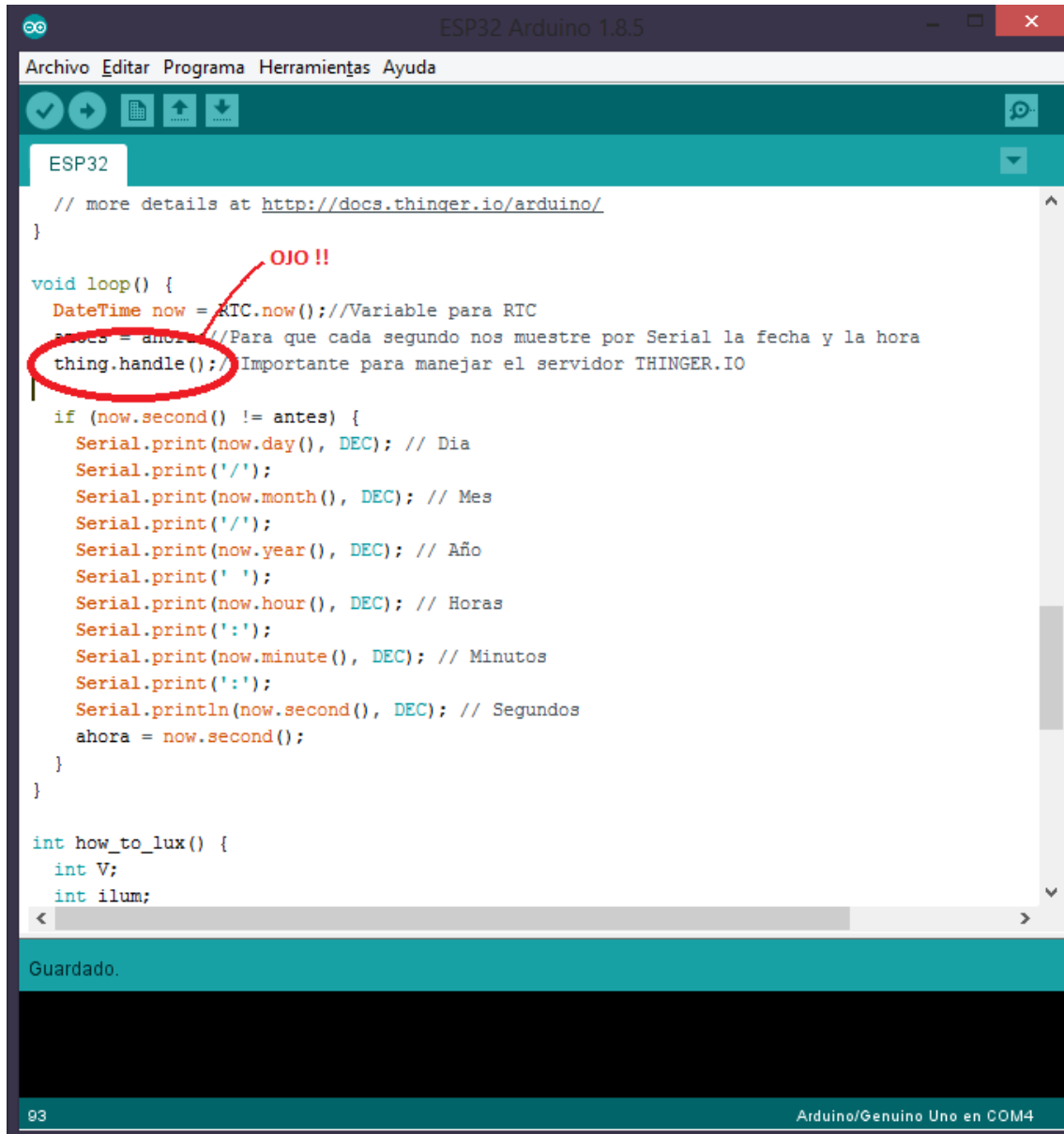
Así pues llegamos a las estructuras básicas:**1- Void setup():**

- Utilizaremos una función para que se conecte a nuestro servicio de WiFi:
 - `thing.add_wifi(SSID, PASS);`
- Será donde declararemos los “resources” o elementos clave que queramos enviar o visualizar en Thingier.io
- Estos elementos claves podrán ser de entrada, salida o ambos. Para ello:
 - **Elementos clave sencillos:**
 - **Control de un pin digital:**
 - `thing[“nombre”] << digitalPin(PIN_DIGITAL);`
 - `thing[“nombre”] << invertedDigitalPin(PIN_DIGITAL);`
 - **Elementos de salida (valor de un sensor):**
 - `Thing[“nombre”] >> outputValue(PIN_ANALOGICO);`
 - **Modificar variables globales, locales, etc:**
 - `Thing[“nombre”] << inputValue(nombre_variable);`
 - `Thing[“nombre”] << inputValue (nombre_variable, { /*Ejecutar alguna función cuando cambia el valor de la variable*/ });`
 - **Control de un servo:**
 - `Thing[“servo”] << servo (myservoInstance);`
 - **Elementos clave avanzados:**
 - **Elementos de entrada:**
 - `Thing[“nombre”] << [] (pson& in){ /*Bloque de instrucciones*/};`
 - **Elementos de salida:**
 - `Thing[“nombre”] >> [] (pson& out){//Instrucciones
Out = función();
};`
 - **Varios elementos de salida a la vez:**
 - `Thing[“nombre”] >> [] (pson& out){//Instrucciones
Out[“nombre_salida”] = función();
Out[“nombre_salida2”] = funcion2();
};`
 - **Modificar variables:**
 - `Thing[“nombre”] << [] (pson& in){ //Bloque de funciones
nombre_variable = in;
};`
 - **Varios elementos de entrada a la vez:**
 - `Thing[“nombre”] << [] (pson& in){//Bloque de funciones
int nombre_variable = in[“nombre_a_visualizar”];
Int nombre_variable2 = in[“nombre_a_visualizar2”];
};`
 - **Entrada y salida a la vez:**
 - `Thing[“nombre”] = [] (pson& in, pson& out){
Out[“nombre”] = in[“nombre_variable”];
};`

2- Void loop():

- Aquí sólo debemos de escribir “thing.handle ()” es la función que se encarga de iniciar la comunicación y de mantenerla abierta con el servidor.
- También podremos manipular los elementos clave aquí con la función “thing.stream (thing[“nombre”])” vista anteriormente en el “manual de usuario para thinger.io”.
- Se podrá llamar a otros dispositivos en esta sección, definiendo un elemento clave en el “A” y utilizando la función thing.call_device(“A”, “Elemento_clave”); , así se podrán comunicar dentro de la misma cuenta distintos dispositivos.

Nota: la función thing.handle() es la más importante, no olvidarla nunca.



```
// more details at http://docs.thinger.io/arduino/
}

void loop() {
  DateTime now = RTC.now(); //Variable para RTC
  antes = ahora; //Para que cada segundo nos muestre por Serial la fecha y la hora
  thing.handle(); // Importante para manejar el servidor THINGER.IO

  if (now.second() != antes) {
    Serial.print(now.day(), DEC); // Dia
    Serial.print('/');
    Serial.print(now.month(), DEC); // Mes
    Serial.print('/');
    Serial.print(now.year(), DEC); // Año
    Serial.print(' ');
    Serial.print(now.hour(), DEC); // Horas
    Serial.print(':');
    Serial.print(now.minute(), DEC); // Minutos
    Serial.print(':');
    Serial.println(now.second(), DEC); // Segundos
    ahora = now.second();
  }
}

int how_to_lux() {
  int V;
  int ilum;
```

Nota: Una vez cargado, configuramos nuestro Data Bucket (solo los “resources” que hemos declarado como “out”). Nuestro dispositivo saldrá como “connected”.

Para más información: <http://docs.thinger.io/arduino/#coding>

7. Instalando el dispositivo

A la hora de instalarlo y conectarlo debemos de tener especial cuidado con las tensiones que vamos a medir puesto que como máximo se admiten 50V en corriente continua, si quisiéramos aumentar la tensión que queremos medir debemos de cambiar los valores de las resistencias en el divisor de tensión.

Nota: es importante que la tensión de alimentación y la tensión que queremos medir no sean de la misma fuente o si lo son deben estar separadas entre si.

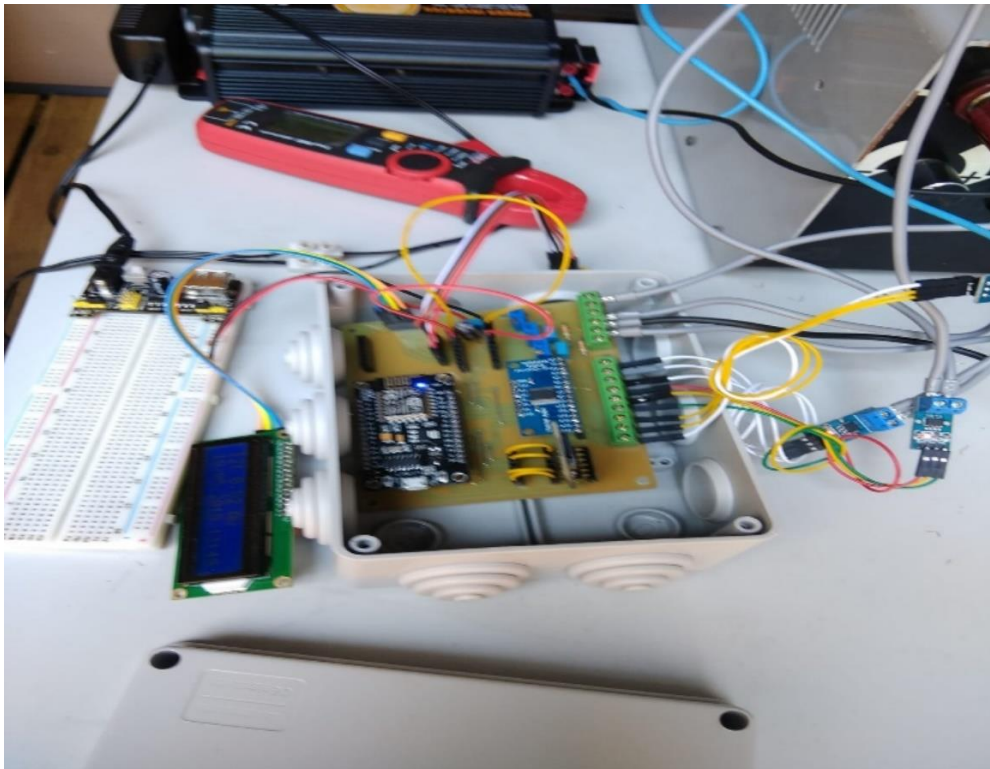


En nuestro caso hemos optado por intercalar un inversor puesto que aísla las tensiones y de ahí conectarlo a un transformador que reduce la tensión de 230 AC a 12 DC. Así ya quedan separadas las tensiones de alimentación y de medición.

Esta necesidad de separar las tensiones de alimentación y de medición es debido a que la tensión de referencia en el regulador instalado no es el negativo o GND si no que es el positivo. Si se juntaran estas tensiones generaría una sobretensión en el ESP8266 y éste se quemaría, provocando daños irreparables.

Nota: como el positivo es la tensión de referencia no hace falta conectar de los tres circuitos, con uno solo ya es suficiente, aunque se pueden conectar los tres.

Detalle del inversor



Detalle de montaje

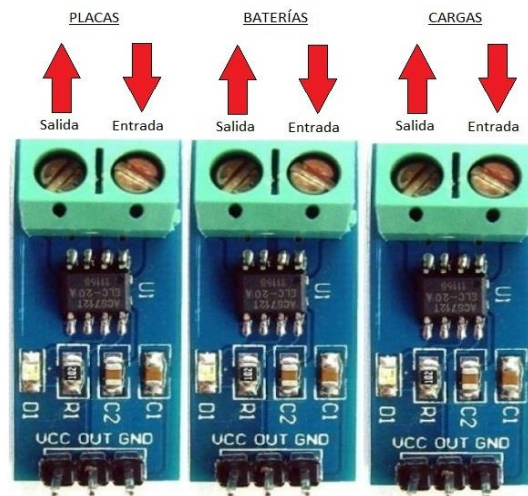
7.1 Conexión de los sensores

Para conectar los sensores de corriente de efecto hall tan sólo hay que tener en cuenta el sentido de la corriente puesto que, en el caso de las baterías, si se están cargando será negativa y si entregan potencia será positiva.

Estos sensores se conectarán de la siguiente manera:

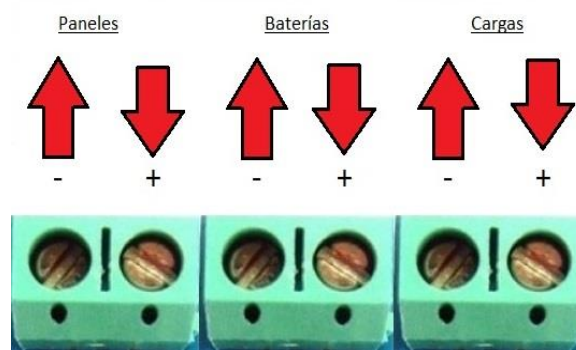
1. Placas solares → Canal 0 del multiplexor
2. Baterías → Canal 1 del multiplexor
3. Circuito de utilización → Canal 2 del multiplexor

Vista de frente y de derecha a izquierda se conectarán 1,2 y 3. Y serán primero negativo, luego la señal y por último el positivo (Gnd, Out, Vcc).



Conexión de corrientes

Vista de frente y de derecha a izquierda se conectarán 1,2 y 3. Se tendrá en cuenta la polaridad la cual será primero positivo y después negativo.



Conexión de tensiones

Para conectar los sensores de tensión se conectará:

1. Placas solares → Canal 3 del multiplexor
2. Baterías → Canal 4 del multiplexor
3. Circuito de utilización → Canal 5 del multiplexor

8. Soluciones a problemas generales

Aquí hablaremos de los problemas generales que se pueden producir en la placa y los problemas que hemos tenido en la fabricación de la misma.

8.1 Problema con pantalla LCD al iniciar

Normalmente cuando conectamos por primera vez la placa, la pantalla no se inicia correctamente o muestra caracteres raros.

Esto es debido a que el bus I2C no se ha inicializado correctamente.

La solución es pulsar el botón de “reset” del ESP8266 hasta que muestre bien los valores en la pantalla.

Si esto no es efectivo, primero desconectar la placa, esperar unos segundos y después reconectar la tensión.

Si vuelve a no ser efectivo, recomendamos o subir de nuevo el código fuente o simplemente conectándolo a un PC a través del puerto microUSB.

8.2 Problema con tarjeta SD

Si al iniciar la tarjeta SD, ocurre un “error al iniciar”, primero asegúrese de que la tarjeta está insertada, puesto que si no hay SD, no podrá iniciar nada.

Si por el contrario está insertada y sigue dando errores, lo más normal es que el bus SPI tenga las líneas cambiadas, es decir, el MISO por el MOSI y viceversa. Este cambio también genera un error al subir un nuevo sketch.

Puede ser que el zócalo esté quemado y/o averiado, repóngalo inmediatamente.

Si lo anterior no lo soluciona, formatee la microSD a FAT32, puesto que el sistema de archivos NTFS no se reconoce.

8.3 Problema entradas analógicas

El ESP8266 tan solo posee una entrada analógica (A0) y puesto que se necesitan mínimo seis entradas, optamos por intercalar un multiplexor de 16 canales.

Este multiplexor representaba una gran ventaja que es la opción de utilizar sus entradas como analógicas. No como la mayoría de multiplexores que suelen ser digitales.

Además no presenta problemas al alimentarlo a 3.3V en vez de a sus 5V que es lo más óptimo.

Su velocidad de cambio de canales tampoco representó un gran problema puesto que lo realiza en nanosegundos.

8.4 Problema con regulador ISFV

El regulador de carga que utiliza la instalación donde incorporaremos nuestro dispositivo IOT posee una inapreciable pero problemática diferencia, puesto que su punto común o de referencia es el positivo y no el negativo como cabe esperar.

En primera instancia optamos por un par de amplificadores operacionales que invirtieran la tensión y así las lecturas se harían de forma convencional. Pero esto requiere un circuito más aparatoso.

La opción más sencilla era adaptarse al regulador y medir tensiones negativas estableciendo el punto común como el positivo y la medición como el negativo.

Así el ESP8266 lee de la siguiente manera:

1. Establece los 0V cuando le llegan 3.3V
2. Establece la máxima tensión cuando le llegan 0V.

8.5 Problema dashboard vacío

Si al entrar en un dashboard configurado, está vacío o se puede modificar sin estar activada la configuración de dashboard, tan solo hay que recargar la página de thinger.

Para más sencillez presionar ctrl+r o F5.

Esto es debido a que no carga bien la base de datos de la plataforma Thinger.io, así al recargar la página, volverá a acceder a nuestra base de datos recuperando toda la información configurada.

8.6 Problema con tensiones de alimentación

Las tensiones de alimentación y de medición deben de ser de distintas fuentes o al menos estar aisladas una de la otra.

Esto es debido a la necesidad de tener una tensión flotante a las entradas de los divisores resistivos. Si fueran las mismas o tuvieran un punto en común se provocaría una sobretensión **quemando** el dispositivo ESP8266 y estropeándose otros como la pantalla LCD o el reloj RTC.

8.7 Problema con RTC

Uno de los problemas más típicos de la RTC (Real Time Clock) son las patillas, puesto que al conectarlo puede producirse una confusión con la línea SDA con la SCL.

Otro problema que puede haber es el modelo, es decir, el código está preparado para un solo modelo que es el “DS3231”. Si se quisiera cambiar el modelo de reloj, habrá que asegurarse que en el código se cambia también el modelo.

El otro modelo más conocido es el “DS1307”.

Si por cualquier motivo la alimentación falla y al restablecerse el reloj se ha desconfigurado, habrá que cambiar la pila que alimenta a la RTC puesto que puede no ser recargable.

El modelo de pila es tipo botón recargable, el código que aparece en la pila que indica que es de ese tipo es “CR2032”.

8.8 Problema con el inversor de tensión

Hay que tener especial cuidado con el tipo de inversor que se esté utilizando puesto que los de onda cuadrada o PWM no actúan de la misma manera que los de onda senoidal pura.

Cuando se utilice un inversor de onda cuadrada, en la puesta en marcha, esperaremos unos segundos antes de inicializar el ESP8266.

9. Posibles mejoras

Como todas las cosas, este dispositivo de monitorización puede ser mejorado tanto para la seguridad de las personas como para la propia del dispositivo.

9.1 Sensor de temperatura

Al estar instalado en una zona como es Extremadura, en verano se alcanzan temperaturas nocivas para el dispositivo que podrían dañar la integridad del ESP8266.

Añadir un sensor LM35 o incluso un DHT11 que no solo mida temperatura sino también humedad, para el control de la misma es una mejora que ayudaría a mantener una seguridad notable en la placa.

Este sensor activaría una alarma en el caso de llegar a la temperatura configurada y se apagaría en caso de superar una segunda consigna de temperatura.

También podría añadirse un ventilador para la refrigeración y no tener que estar pendiente de si salta la alarma o no.

9.2 Añadir Relés de control

En algunas instalaciones, podría darse el caso del desconocimiento por parte del usuario si conectara más carga de la soportada al circuito de utilización.

Estos relés se activarían al superar una cierta consigna configurada por el usuario o por el técnico competente. Podrán ser activados o desactivados según criterio, pero automáticamente, se desconectarán por exceso de consumo.

9.3 Led's indicadores

Se podrían añadir led's que indicaran la conexión a Internet, la transmisión de datos a la misma o el nivel de señal WiFi que llega al dispositivo.

También pueden añadirse led's para indicar el nivel de tensión y corriente mínimo y máximo que se está produciendo.

9.4 Código vía OTA (Over The Air)

A veces es difícil acceder al lugar donde está instalado el dispositivo, por ello, los desarrolladores crearon lo que se denomina programación vía OTA.

Esta programación consiste en subir en primera instancia un código al ESP8266 y vía WiFi, a través de cualquier ordenador, poder modificar el sketch que anteriormente se subió.

El problema con esta mejora es que no puede ser integrado con Thinger.io todavía.

9.5 Utilizar una nueva versión del ESP8266 → El ESP32

El dispositivo ESP8266 está ya obsoleto, posee un convertidor de 10bits y muchas limitaciones a la hora de la programación.

El ESP32 es mucho más rápido, posee un convertidor AD de 12 bits, mayor capacidad de almacenamiento y mayor rapidez de conexión WiFi. Además, incorpora Bluetooth.

9.6 Utilizar un SAI

La utilización de un sistema de alimentación ininterrumpido surge de la necesidad de mantener siempre encendido el dispositivo para que siempre esté monitorizando todo lo que ocurre en la instalación en la que se ha conectado.

Así si se desconectara la instalación por reparaciones o por cualquier problema, seguiríamos teniendo acceso al dispositivo y podríamos controlarlo.

9.7 Modificación de código

El código, aunque funciona como se espera, debiera de ser depurado y mejorado con creces.

Estos dispositivos se caracterizan por consumir muy poca energía, pero se podría reducir aún más actualizando el código para que entrara en modo DEEPSLEEP. Así se conseguiría consumir aún menos y sería más eficiente.

Por otra parte, se podrían aumentar las muestras por ciclo que toma para las mediciones, puesto que cuantas más muestras consiga, más exacta es la medición. Pero habrá de andarse con cuidado puesto que esto reduce el tiempo de ejecución.

Y puestos a mejorar la programación, cambiar totalmente la forma de pensar en texto estructurado a código orientado a objetos.

9.8 Utilizar RaspberryPi3

La idea de esta mejora no es más que poder crear tu propio servidor Thinger.io en una Raspberry.

No tendríamos la necesidad de acceder a la plataforma Thinger.io mediante un navegador de internet o de arrancar un ordenador.

El problema es que cuesta unos 10€. Aunque una vez pagados no hay que volver a pagarlos y ya es tuyo para siempre.

Puede encontrarse en el siguiente enlace:

<https://thinger.io/product/raspberry-pi3-image-thinger-io-server/>

9.9 Modificación para corriente alterna

Al medir sólo corriente continua, queda muy limitado puesto que la mayoría de los dispositivos que utilizamos normalmente son de corriente alterna y si quisiéramos instalarlo en un circuito que no sea de corriente continua, no podremos.

Para esta modificación se necesitaría cambiarlo todo por completo pero eso es otra historia que será contada en otro proyecto.

9.10 Midiendo la eficiencia

Con un sensor de radiación se mediría la energía que le llega a las placas solares para saber cuánto produce con esa radiación. Es decir, medir la eficiencia de las placas solares.

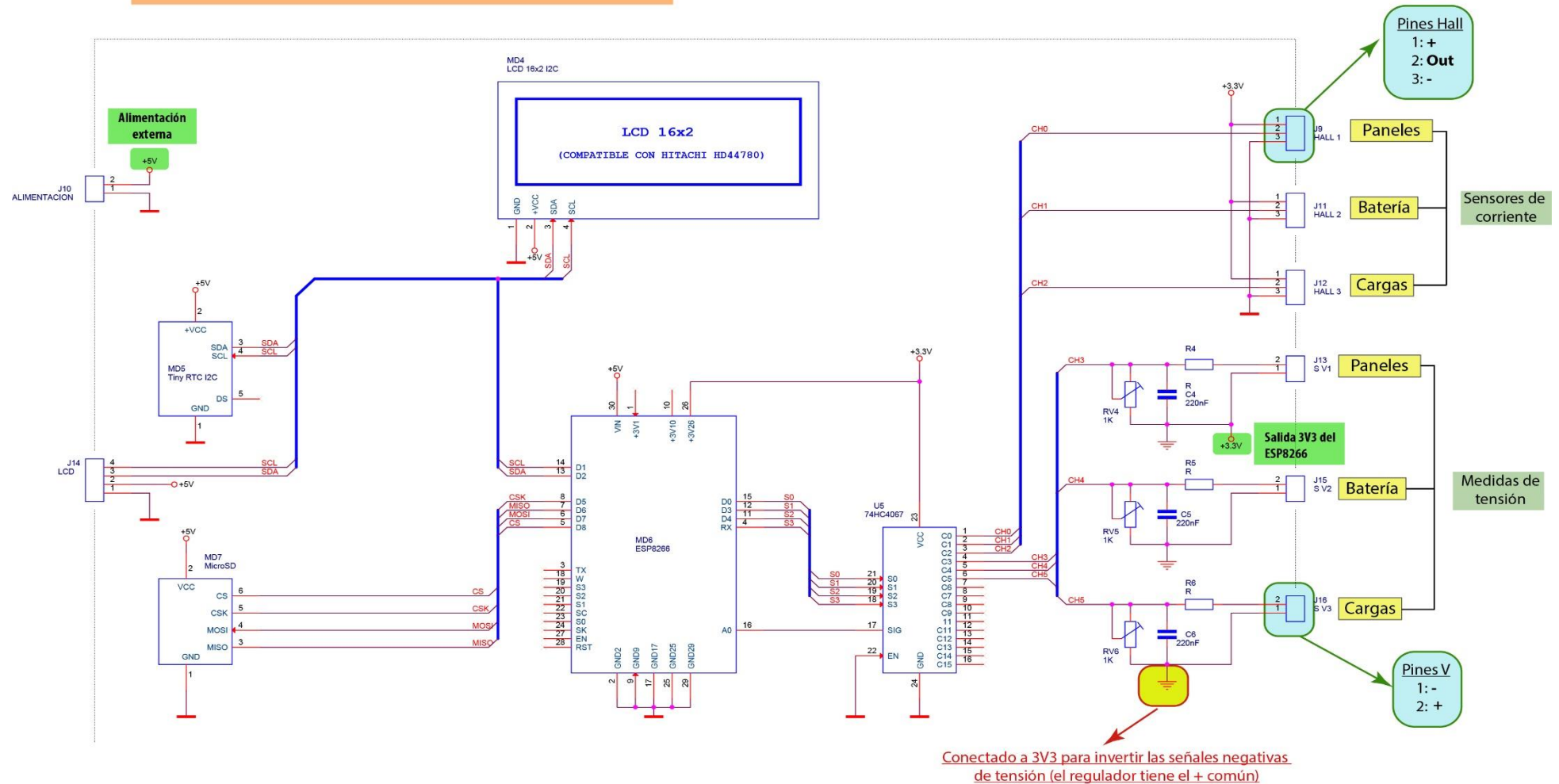
También mediríamos la presión atmosférica y la velocidad del viento, puesto que son datos relevantes a la hora de calcular la eficiencia de dichas placas solares.



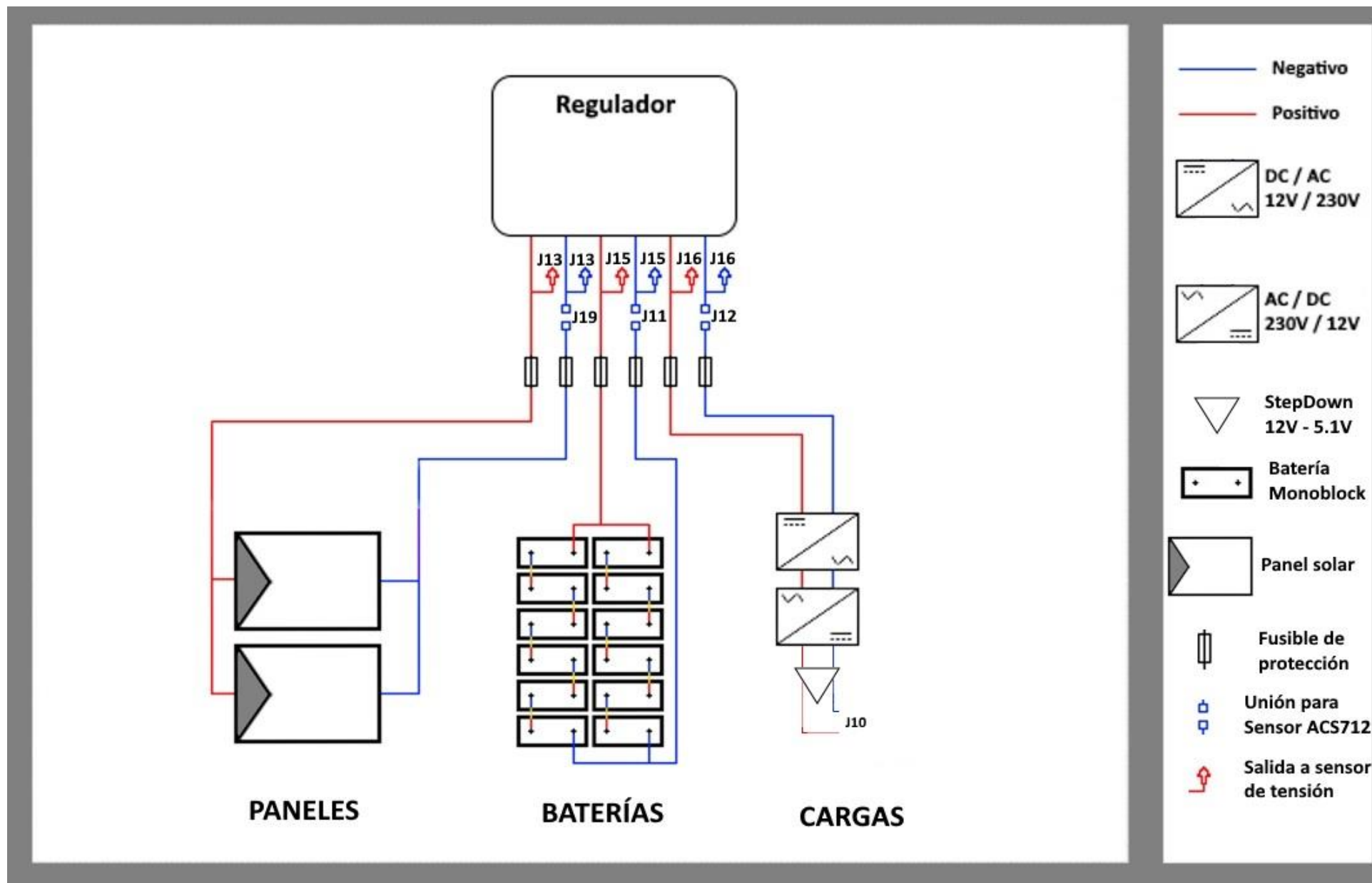
10. Esquemas con especificaciones

Alimentación	- Externa (5V)	- ESP8266 - Pantalla LCD - RTC - Zócalo microSD
	- Salida ESP8266 (3V3)	- Divisor R (medida V) - Hall (medida I)

¡Ojo! La alimentación externa tiene que estar aislada de las tensiones de entrada porque tienen el + común



Esquema PCB



Esquema de conexiones