

Fco. Javier Rodriguez Navarro

ANEXO

De 0 a 100
con
Arduino y ESP32



De 0 a 100

con

Arduino y ESP32

Fco. Javier Rodríguez Navarro

Copyright © 2025 Fco. Javier Rodriguez Navarro,

**Portada, texto, composición e ilustración:
© Fco. Javier Rodriguez Navarro**

**Anexo al libro de ISBN: 9798325177422
Publicación independiente**

Queda prohibida la reproducción de este material por cualquier medio. Para más información puede ponerse en contacto con el autor.

**Web: www.pinguytaz.net
GitHub: <https://github.com/pinguytaz>
Mail: cuentas@pinguytaz.net**

Índice

1	Introducción	1
1.1	Fe de erratas	1
2	Actualización ESP32	5
2.1	WIFI	6
3	Ampliaciones	9
3.1	Entradas digitales	9
3.2	Almacenamiento	9
3.2.1	Generación de partición NVS Partition	9
4	ANEXO “Tablas”	11
5	ANEXO “Figuras”	13

1 INTRODUCCIÓN

Este Anexo es una ampliación del libro “De 0 a 100 con Arduino y ESP32” que se puede comprar en Amazon (<https://amzn.eu/d/32OJJdF>) y contempla la Fe de erratas y las actualizaciones de librerías utilizadas, así como la ampliación de algunos temas y otros nuevos para profundizar más en la programación de Arduino y ESP32.

Por ese motivo muchos de los temas tratados hacen referencia a capítulos e informaciones dadas en el libro, siendo de esta forma una actualización en línea del libro anteriormente nombrado.

1.1 Fe de erratas

1. Página 3

NOTA: El nombre de los **ejemplos** publicados

2. Página 9

tiene varias **mejoras** al tener

3. Página 17

Archivo: Tenemos las opciones de crear un nuevo **programa**

4. Página 18 No estaba en negrita

Herramientas:

5. Página 19

Se habla de los iconos de la parte derecha cuando debería decir **izquierda**

6. Página 20 Se corrigen a texto en castellano

Indica botonera derecha cuando debería decir **izquierda**

Plotter Serie

Monitor Serie

7. Página 25

ultimo botón de la derecha **de** la botonera superior

8. Tabla-B se corrige la descripción del OR “|”

Tabla B: Operadores de bits.

Operador	Descripción
&	AND binario $0b0110 \& 0b1101 = 0b0100$
	OR binario $0b0110 0b1101 = 0b1111$
^	XOR binario $0b0110 \wedge 0b1101 = 0b1011$
~	Negación $\sim 0b0110 = 0b1001$
«	Desplazamiento a la izquierda. Valor » desplaza
»	Desplazamiento a la derecha Valor « desplaza

9. Página 42

al hablar de los parámetros por referencia sobra “una”

.... **es la solución limpia y recome**

Se corrige el comentario del código que dice que retorna entero y debe decir **flotante**

10.El capitulo 11 pasa a llamarse “Programación ESP32” para diferenciarlo del 5.

11.Tabla AM se corrige el voltaje del color verde.

Tabla AM: Valores típicos LEDs

Color	Rango Tensión	Rango Corriente
Rojo	1,8 – 2,2V	10-20 mA
Naranja	2,1 – 2,4V	10-20mA
Amarillo	2,1 – 2,4V	10-20mA
Verde	2,2- 2,5V	10-20mA
Azul	3,5 – 3,8V	20mA
Blanco	3,6V	20mA

12.----

2 ACTUALIZACIÓN ESP32

Todos los ejemplos del libro se realizaron con la versión 2.x(basado en ESP-IDF 4.4) del core Arduino para ESP32, durante la edición del libro apareció la versión 3.x(basada en ESP-IDF 5.1) del core de Arduino.

Tenemos una guía de migración en la página oficial (https://docs.espressif.com/projects/arduino-esp32/en/latest/migration_guides/2.x_to_3.0.html) también hemos detectado algunos cambios en nombres de contantes así que hemos decidido crear capítulos de actualización con esos cambios y ampliación de la información indicando en que parte del libro afecta.

- ADC

Desaparecen las funciones: `analogSetClockDiv`, `adcAttachPin`, `analogSetVRefPin`.

- LEDC

La generación de PWM ha cambiado para facilitar el uso.

Se eliminan: `ledcSetup`, `ledcAttachPin`. Que se ha fusionado con la nueva función **`ledcAttach`**.

```
bool ledcAttach( pin, freq, resolucion);  
bool ledcAttachChannel(pin, freq, resolucion, canal);  
  
ledcChangeFrequency(PIN, frequency, resolution)
```

- Sensor Hall

Desaparece, eliminando el capítulo 10.2 en el que ya comentábamos que no era mejor usar un dispositivo HAL externo.

- BLE

Cambios de retorno de APIs y tipo de parámetro de `std::string` se pasa al estilo Arduino String.

Cambio tipo de datos UUID de `uint16_t` a la clase BLEUUID.

El tipo de retorno de los métodos `BLEScan::start` y `BLEScan::getResults` a cambiado de `BLEScanResults` a `BLEScanResults*`.

- WIFI
 - **flush()** está destinado a enviar el contenido del buffer de transmisión, ahora en **WiFiClient**, **WiFiClientSecure** y **WiFiUDP** se utilizara el método **clear()**
 - **WiFiServer** tiene la función **accept()** y **available()** con la misma funciabilidad, pero pondremos en desuso **available()**

2.1 WIFI

Tratamos de las actualizaciones del capítulo “10.5 WIFI”

Se actualizan algunas tablas de eventos ya que estos han cambiado algo la nomenclatura pasando a cambiar el prefijo “SYSTEM” por “ARDUINO” y ademas añade el modulo al que afecta el evento en este caso es “WIFI”

Tabla A: “Tabla AD Eventos WIFI en modo AP”

Evento	Descripción
ARDUINO_EVENT_WIFI_AP_START	AP iniciado
ARDUINO_EVENT_WIFI_AP_STOP	AP parado
ARDUINO_EVENT_WIFI_AP_STACONNECTED	Se conecta una estación al AP
ARDUINO_EVENT_WIFI_AP_STADISCONNECTED	Se desconecta una estación.
ARDUINO_EVENT_WIFI_AP_STAIPASSIGNED	Se asigna una IP
ARDUINO_EVENT_WIFI_AP_PROBEREQRECVED	Llega un paquete de sondeo.
ARDUINO_EVENT_WIFI_AP_GOT_IP6	Preferencia IP6


Tabla B: “Tabla AF Eventos WIFI en modo STA”

Evento	Descripción
ARDUINO_EVENT_WIFI_READY	Interfáz WIFI preparado
ARDUINO_EVENT_WIFI_SCAN_DONE	Escaneo completado
ARDUINO_EVENT_WIFI_STA_START	Estación iniciada
ARDUINO_EVENT_WIFI_STA_STOP	Estación parada
ARDUINO_EVENT_WIFI_STA_CONNECTED	Estación conectada
ARDUINO_EVENT_WIFI_STA_DISCONNECTED	Desconexión de la estación
ARDUINO_EVENT_WIFI_STA_AUTHMODE_CHANGE	Modo autorización cambiado
ARDUINO_EVENT_WIFI_STA_GOT_IP	Obtiene IP
ARDUINO_EVENT_WIFI_STA_LOST_IP	Estación ha perdido la IP

3 AMPLIACIONES

En este capítulo están las ampliaciones de nuevos temas que complementan el libro.


3.1 Entradas digitales

Se añade un ejemplo para complementar mejor el apartado 7.1.4 el ejemplo es 

3.2 Almacenamiento

Ampliamos el capítulo “**8.2.1 Preference.h en ESP32**” para poder actualizar los datos de “*Preference*” sin necesidad de realizar un programa que los grave como se define en el ejemplo


 y después leer estos datos con el ejemplo

 esto se aleja del IDE de Arduino al tener que instalar las aplicaciones de *ESP-IDF* pero creemos que es de interés, esto se puede ver en 3.2.1 Generación de partición NVS Partition .

3.2.1 Generación de partición NVS Partition

Los datos tipo “Clave-Valor” que se explican en “**8.2.1 Preference.h en ESP32**” se almacenan en la partición tipo “NVS” que como se ve en el capítulo “**8.3 Flash ESP32 (Múltiples sistemas de archivo)**” el ESP32 puede tener un sistema de archivos con diferentes particiones con diferentes usos, en este caso es la datos nvs que se utiliza para almacenar datos en plan EEPROM o la de nuestro interés *Preference* y vamos a explicar como añadir los

datos a esta partición, desde línea de comandos, para su posterior lectura en nuestro programa.

Tenemos un nuevo ejemplo con Script en Almacenamiento  en el que el programa *NVS.ino* los leerá.

Deberemos instalar las herramientas ESPTOOLS ya que algunos ejecutables necesarios no se instalan en el entorno Arduino.

Lo primero será generar un fichero CVS que tendrá los namespace y claves-valor que tengamos en ellos.

```
key,type,encoding,value                                CABECERA
prueba,namespace,,                                   NOMBRE DEL NAMESPACE
Campo1,data,string,UncampoqualquieraCAMPOS CON LOS DATOS
Campo2,data,string,Valordelaclave
Autor,data,string,elNombre
```

Tipos: file, data, namespace.

Codificación: u8, i8, u16, i16, u32, i32, u64, i64, string, hex2bin, base64, y binary.

Luego crearemos la imagen de la partición y la subiremos.

```
nvs_partition_gen.py generate <CSV> <IMG> <TAMANO>
```

```
esptool.py --chip esp32 write_flash -z <DIRECCION> <IMG>
```

4 ANEXO “TABLAS”

Tabla A: “Tabla AD Eventos WIFI en modo AP”	6
Tabla B: “Tabla AF Eventos WIFI en modo STA”	7

5 ANEXO “FIGURAS”