

Fco. Javier Rodríguez Navarro

ANEXO

De 0 a 100
con
Arduino y ESP32



De 0 a 100

con

Arduino y ESP32

Fco. Javier Rodríguez Navarro

Copyright © 2024 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.pinguylaz.net

GitHub: <https://github.com/pinguylaz>

Mail: cuentas@pinguytaz.net

Índice

1 Introducción	1
2 Actualización ESP32	3
2.1 WIFI	4
3 Ampliaciones	5
3.1 Almacenamiento	5
3.1.1 Generación de partición NVS Partition	5
4 ANEXO “Tablas”	7
5 ANEXO “Figuras”	9

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 linea del libro anteriormente nombrado.

Tabla A: Fe de erratas

Lugar	Pone	Debe poner
Pag. 32 Tabla-B	en el operador pone AND Binario	Debe ser OR binario

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 qué parte del libro afecta.

- WIFI
 - **flush()** está destinado a enviar el contenido del buffer de transmisión, ahora en **WiFiClient**, **WiFiClientSecure** y **WiFiUDP** se utilizará el método **clear()**
 - **WiFiServer** tiene la función **accept()** y **available()** con la misma funcionalidad, pero pondremos en desuso **available()**
- ADC
- BLE
- Sensor Hall
- I2C
- LEDC
- Timer

- UART

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

Evento	Descripción
ARDUINO_EVENT_WIFI_READY	Interfaz 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 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 grabe como se define en el ejemplo **Gpreferencias** y después leer estos datos con el ejemplo **Preferencias** 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 3.1.1 Generación de partición NVS Partition .

3.1.1 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 linea de comandos, para su posterior lectura en nuestro programa.

Tenemos un nuevo ejemplo con Script en Almacenamiento **NVS** 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 sera generar un fichero CVS que tendrá los namespace y claves-valor que tengamos en ellos.

```
key, type, encoding, value  
prueba, namespace,, CABECERA  
Campo1,data,string,UncampocualquierA NOMBRE DEL NAMESPACE  
CAMPOS CON LOS DATOS  
Campo2,data,string,Valor dela clave  
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: <i>Fe de erratas</i>	1
Tabla B: “Tabla AD Eventos WIFI en modo AP”	4
Tabla C: “Tabla AF Eventos WIFI en modo STA”	4

5 ANEXO “FIGURAS”