Computación Ubicua

Sesión 3 – Introducción a NodeMCU

Ana Castillo Martínez

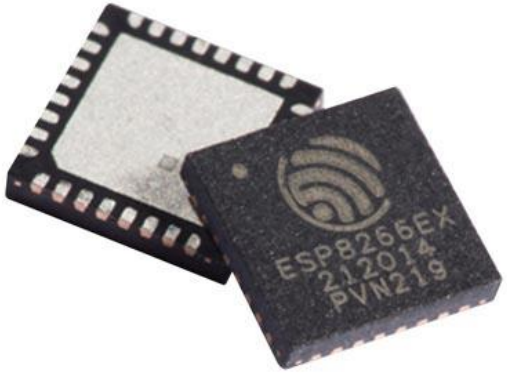
Javier Albert Segui

Universidad de Alcalá

- ESP8266
- ESP32
- NODEMCU
- Entorno de desarrollo
- Conectando al exterior

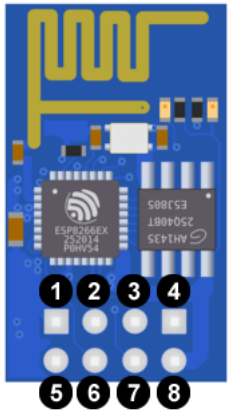
ESP8266

- Es un chip (SoC) de bajo costo con una pila tcp/ip completa y un microcontrolador.
- Utiliza una CPU de 32 bits Tensilica Xtensa LX106 a 80 ó 160 MHZ
- Tiene interfaces GPIO, PWM y ADC y soporta protocolos I2C, SPI, I2S, serie e infrarrojos
- Incorpora un reloj RTC
- Funciona entre 3 - 3.6V y 80 mA
- Necesita de una memoria externa



ESP8266 - II

- Soporta IPv4 y los protocolos TCP/UDP/HTTP/FTP
- No soporta HTTPS en un principio.
- 3 modos de operación:
  - Active mode o modo activo: a pleno rendimiento.
  - Sleep mode o modo dormido
  - Deep sleep o modo en sueño profundo



1 GND
2 GPIO2
3 GPIO0
4 RXD
5 TXD
6 CH_PD
7 RESET
8 Vcc

Usos

- Electrodomésticos conectados.
- Automatización del hogar.
- Casas inteligentes.
- Automatización de la industria.
- Cámaras IP.
- Redes de sensores.
- Wereables.
- IoT (Internet of Things o Internet de las Cosas)
- IIoT (Industrial Internet of Things o Internet de las Cosas para el sector Industrial)

ESP32

- Es un chip (SoC) de bajo costo con una pila tcp/ip completa y un microcontrolador. Mas potente que su hermano menor el ESP8266
- Procesador Xtensa Dual-Core LX6 de 32 bits a 160 ó 240 MHz. Un nucleo para la comunicación y otro nucleo para el resto de procesos
- Bluetooth 4.0 BLE
- Acelerador de encriptación por hardware

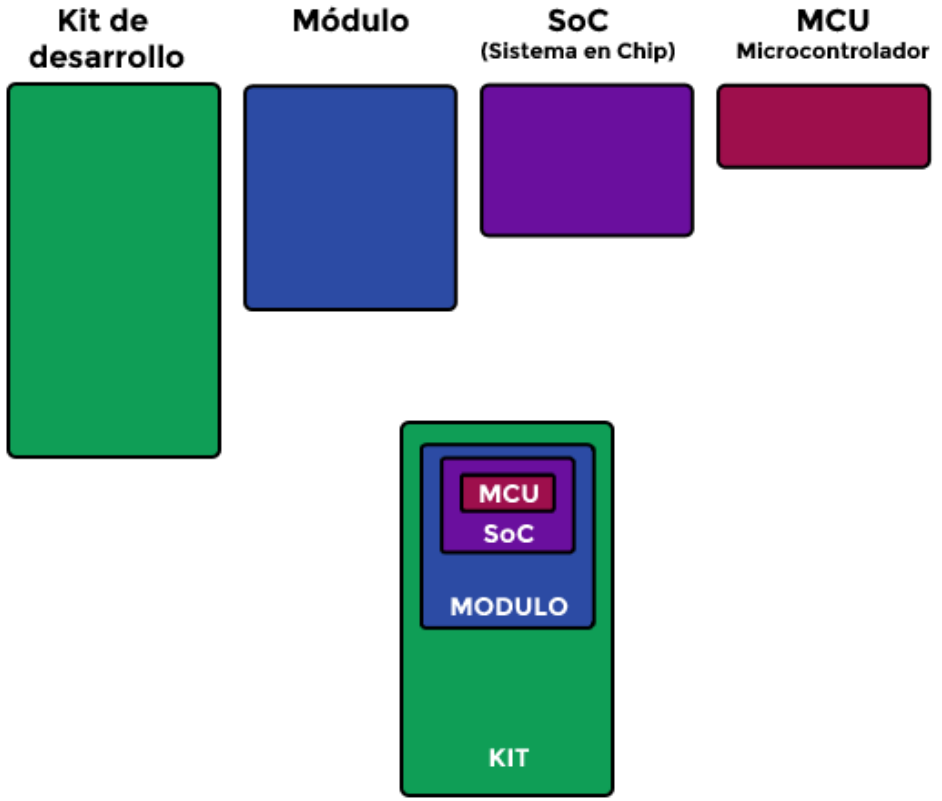
Comparativa ESP8266 – ESP32

Característica	ESP8266	ESP32
Procesador	Tensilica LX106 32 bit a 80 MHz (hasta 160 MHz)	Tensilica Xtensa LX6 32 bit Dual-Core a 160 MHz (hasta 240 MHz)
Memoria RAM	80 kB (40 kB disponibles)	520 kB
Memoria Flash	Hasta 4 MB	Hasta 16 MB
ROM	No	448 kB
Alimentación	3.0 a 3.6 V	2.2 a 3.6 V
Rango de temperaturas	-40°C a 125°C	-40°C a 125°C
Consumo de corriente	80 mA (promedio). 225 mA máximo	80 mA (promedio). 225 mA máximo
Consumo en modo sueño profundo	20 uA (RTC + memoria RTC)	2.5 uA (10 uA RTC + memoria RTC)
Coprocesador de bajo consumo	No	Sí. Consumo inferior a 150 uA
WiFi	802.11 b/g/n (hasta +20 dBm) WEP, WPA	802.11 b/g/n (hasta +20 dBm) WEP, WPA
Soft-AP	Sí	Sí

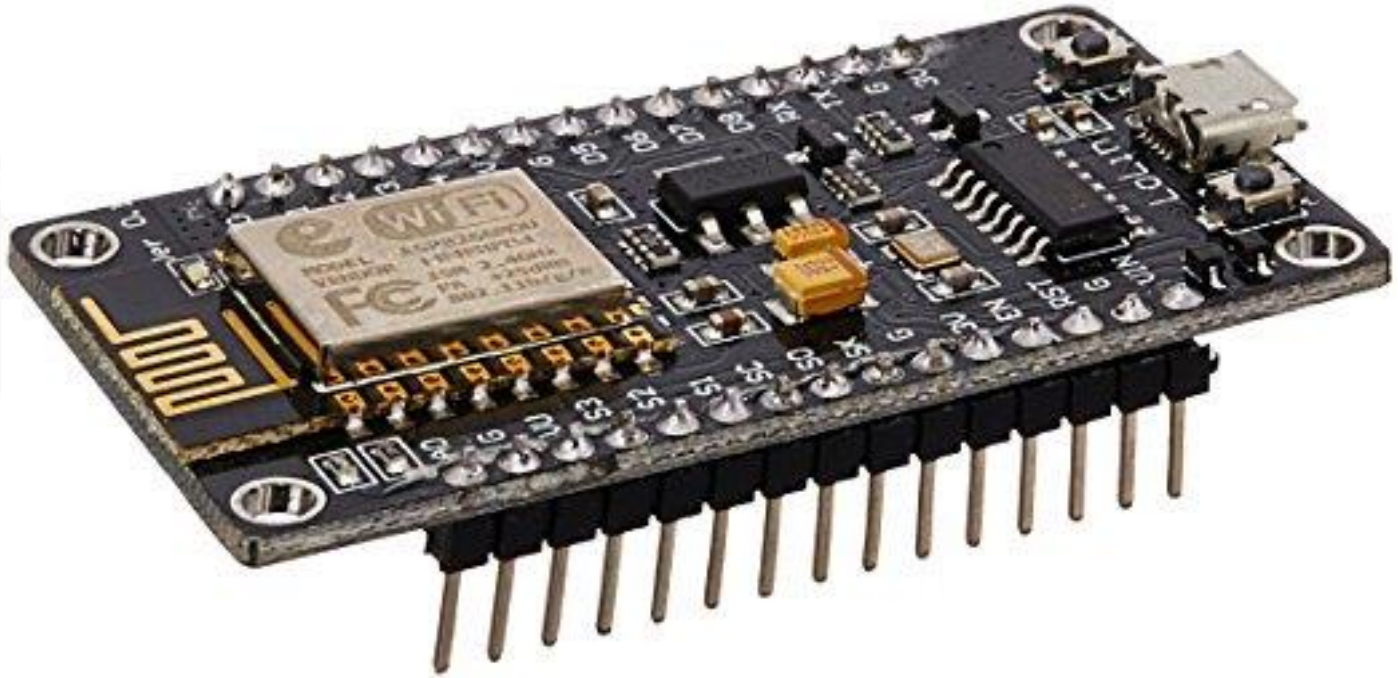
Bluetooth	No	v4.2 BR/EDR y BLE
UART	2* (En una de ellas solo puede utilizarse el pin Tx)	3
I2C	1	2
SPI	2	4
GPIO (utilizables)	32	11
PWM	8	16
ADC	1 (10 bit)	18 (12 bit)
ADC con preamplificador	No	Sí (Bajo ruido) Hasta 60 dB
DAC	No	2 (8 bit)
1-Wire	Implementado por software	Implementado por software
I2S	1	2
CAN bus	No	1 x 2.0
Ethernet	No	10/100 Mbps MAC
Sensor de temperatura	No	Sí
Sensor efecto HALL	No	Sí
IR	Sí	Sí
Temporizadores	3	4 (64 bits)
Encriptación por hardware	No (TLS 1.2 por software)	Sí (AES, SHA, RSA, ECC)
Gen. de núm. aleatorios	No	Sí
Encriptación de la flash	No	Sí
Arranque seguro	No	Sí

NodeMCU

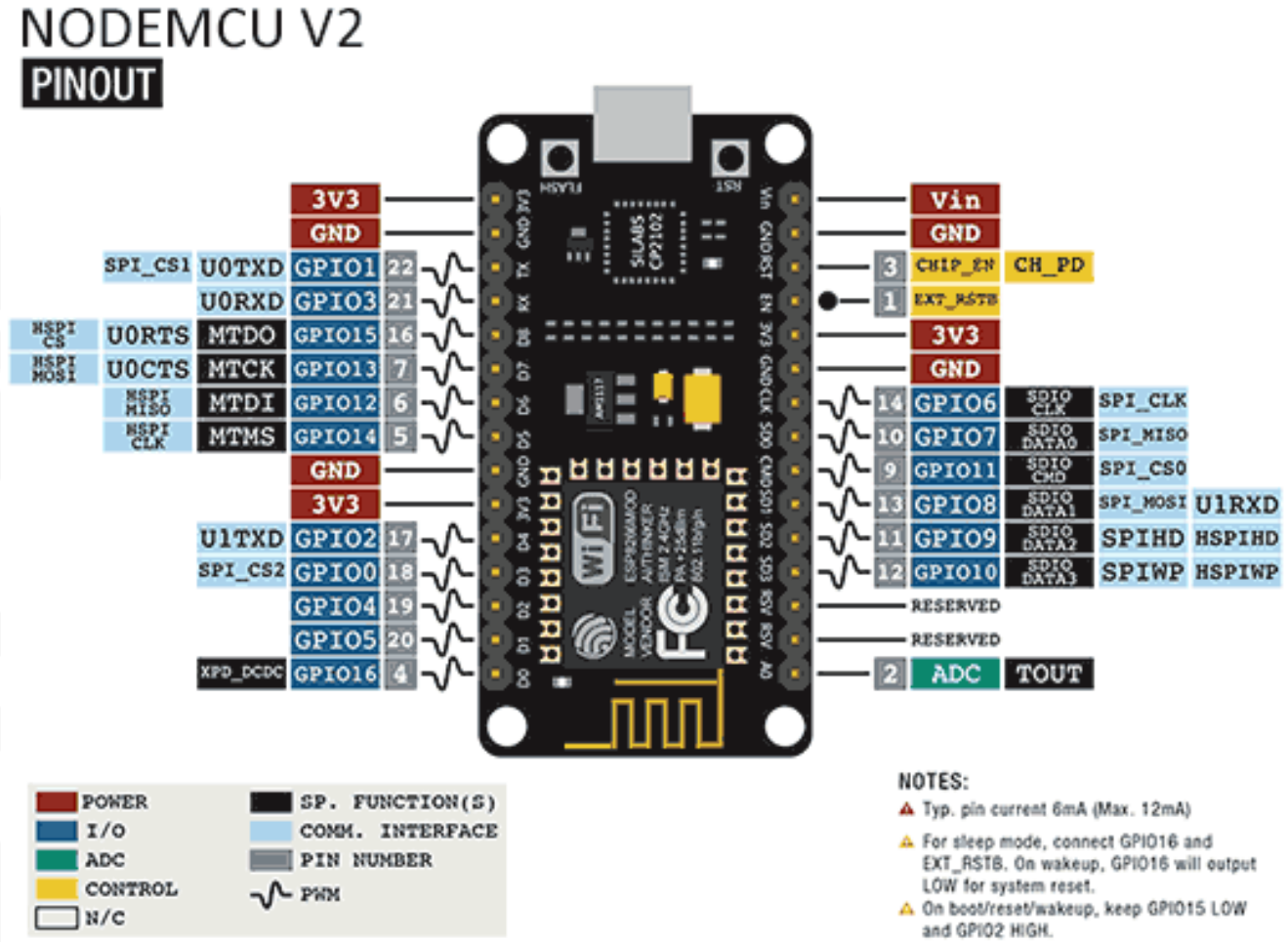
- Es una plataforma IoT de código abierto.
- NodeMCU **NO** es un microcontrolador es una placa o kit de desarrollo.
- Incluye un firmware que se ejecuta en el SoC ESP8266 / ESP32.
- Tiene 11 pines GPIO



Placa nodeMCU



NodeMCU Pinout

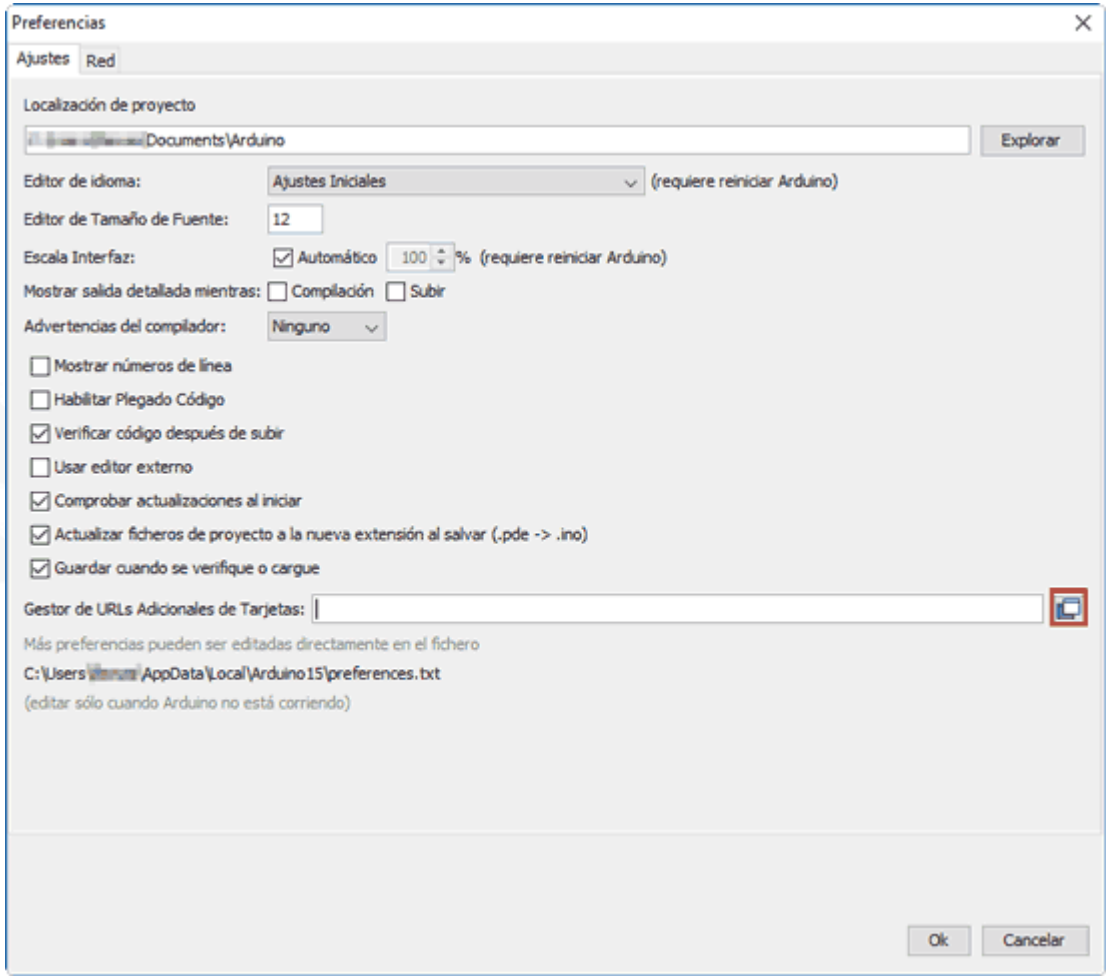


Puesta en marcha

- NodeMCU
- Arduino IDE
- Configuración del IDE

Configurar el IDE

- Añadir en preferencias: http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Esto importará lo necesario para poder añadir el paquete de desarrollo de ESP8266

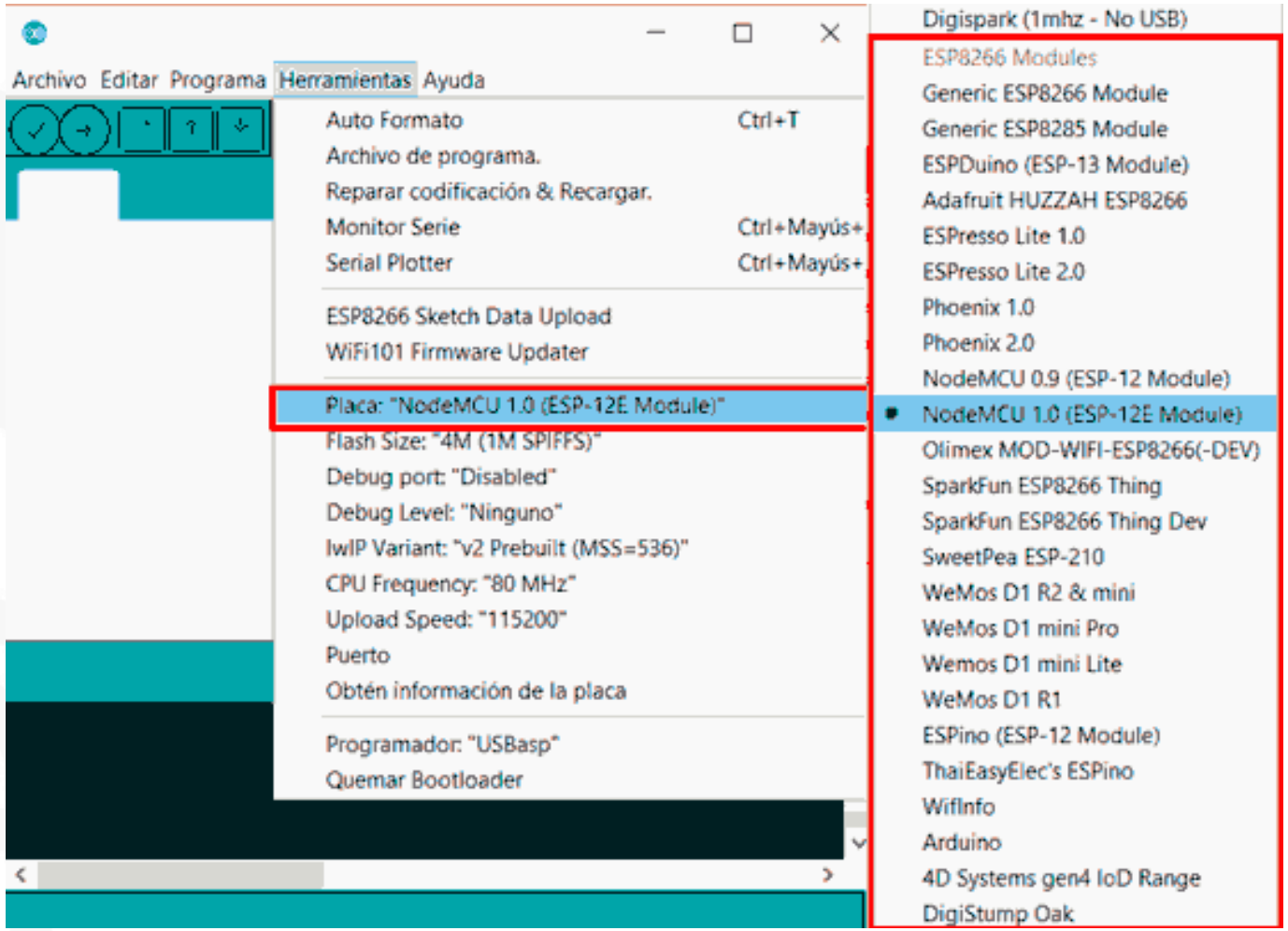


Configurar el IDE - 2

- Después en Herramientas → Placa → Gestor de Tarjetas
- Buscamos los paquetes de ESP8266 y/o ESP32 dependiendo del chip que incluya nuestra nodeMCU

Configurar el IDE - 3

- Seleccionar la placa que vamos a usar



Conectando al exterior

```
// REGION: Librerias

#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <Wire.h>

//REGION: Configuración wireless
const char* ssid="AAAAAAAAAA";
const char* password = "LaClav3d3laWiFi";
const char* namehost="NODE1";
//REGION: Conectividad
WiFiClient espClient;

void setup() {
  /* Enviamos mensajes via puerto serie */
  initSerial();
  /* Conectamos el nodeMCU a la red Wifi */
  initWifi();
}

void initSerial()
{
  Serial.begin(115200);
  Serial.setTimeout(5000);

  // Inicialización del puerto serie
  while(!Serial) { }

  //Mensaje de bienvenida
  Serial.println("Booting ..... \n");
  Serial.println("\tDispositivo en marcha");
}

void initWifi()
{
  Serial.println();
  Serial.print("Wifi conectando a: ");
  Serial.println( ssid );
  WiFi.hostname(namehost);
  WiFi.begin(ssid,password);

  while( WiFi.status() != WL_CONNECTED ){
      delay(500);
      Serial.print(".");
  }
  Serial.println("Wifi conectada!");
  Serial.print("NodeMCU IP Address : ");
  Serial.println(WiFi.localIP() );
}
```

Over The Air (OTA)

```
#include <ArduinoOTA.h>
void initOTA()
{
  /* configure dimmers, and OTA server events */
  analogWriteRange(1000);
  analogWrite(led_pin, 990);

  for (int i = 0; i < N_DIMMERS; i++) {
    pinMode(dimmer_pin[i], OUTPUT);
    analogWrite(dimmer_pin[i], 50);
  }
  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH) {
      type = "sketch";
    } else { // U_SPIFFS
      type = "filesystem";
    }
    for (int i = 0; i < N_DIMMERS; i++) {
      analogWrite(dimmer_pin[i], 0);
    }
    analogWrite(led_pin, 0);

    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
     for (int i = 0; i < 30; i++) {
       analogWrite(led_pin, (i * 100) % 1001);
       delay(50);
     }
     Serial.println("\nEnd");
  });
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
     Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
     Serial.printf("Error[%u]: ", error);
     if (error == OTA_AUTH_ERROR) {
       Serial.println("Auth Failed");
     } else if (error == OTA_BEGIN_ERROR) {
       Serial.println("Begin Failed");
     } else if (error == OTA_CONNECT_ERROR) {
       Serial.println("Connect Failed");
     } else if (error == OTA_RECEIVE_ERROR) {
       Serial.println("Receive Failed");
     } else if (error == OTA_END_ERROR) {
       Serial.println("End Failed");
     }
     ESP.restart();
  });
  ArduinoOTA.begin();
  Serial.println("Ready OTA UPDATE");
```