Documentación de comandos y mensajes del ESP32

Item generales

- Tiene un servidor web básico que sirve una página HTML configurable.
- Permite manejar un punto de acceso WiFi (AP).
- Soporta la creación y gestión dinámica de endpoints HTTP con datos JSON.
- Procesa comandos recibidos en formato JSON ***vía UART o configurable en SPI.***

Comandos soportados

1. `SCAN`

- Descripción: Escanea las redes WiFi disponibles y envía la lista de SSIDs y su intensidad (RSSI) por UART.

Ejemplo JSON:

```
{ "cmd": "SCAN" }
```

- Mensajes UART:
    - OK: `[ESP32] Escaneando redes WiFi...`
    - Error: `[ESP32] No se encontraron redes WiFi.`

2. `CONNECT`

- Descripción: Conecta el ESP32 a una red WiFi usando SSID y contraseña.
- Parámetros:
    - `"ssid"`: nombre de la red WiFi.
    - `"pass"`: contraseña de la red.

- Ejemplo JSON:

```
{ "cmd": "CONNECT", "ssid": "MiRed", "pass": "12345678" }
```

Mensajes UART:

- OK: `[ESP32] OK|IP: xxx.xxx.xxx.xxx` (IP asignada)
- Error: `[ESP32] Error al conectar a la red: MiRed`
- Error de parámetros: `[ESP32] Falta SSID o PASS en CONNECT.`

3. `DISCONNECT`

- Descripción: Desconecta el ESP32 de WiFi o apaga el punto de acceso.
- Parámetro:
    - `"target": "WiFi"` (default) o `"AP"`

Ejemplo JSON:

```
{ "cmd": "DISCONNECT", "target": "WiFi" }
```

- Mensajes UART:
    - OK WiFi: `[ESP32] WiFi desconectado.`
    - Error WiFi: `[ESP32] No está conectado a ninguna red.`
    - OK AP: `[ESP32] AP detenido`

4. `AP`

- Descripción: Inicia un punto de acceso WiFi con SSID y contraseña.
- Parámetros:
    - `"ssid"`: nombre del AP.

- `"pass"`: contraseña del AP.

Ejemplo JSON:

```
{ "cmd": "AP", "ssid": "ESP32_AP", "pass": "password" }
```

- Mensajes UART:
  - OK: `[ESP32] AP iniciado: ESP32_AP | IP: 192.168.4.1`
  - Error: `[ESP32] Error al iniciar AP`
  - Error de parámetros: `[ESP32] Faltan datos para AP`

5. `WebServer`

- Descripción: Actualiza o crea un endpoint HTTP con respuesta JSON.
- Parámetros:
  - `"label"`: nombre del endpoint (p. ej. `"Temperatura"`).
  - `"data"`: objeto JSON con datos a responder en ese endpoint.

Ejemplo JSON:

```
{
  "cmd": "WebServer",
  "label": "Temperatura",
  "data": { "dato1": 25 }
}
```

- Mensajes UART:
  - OK: `[ESP32] Endpoint /Temperatura actualizado`
  - Error (sin conexión WiFi o AP): `[ESP32] No se puede crear endpoint. No conectado a WiFi ni en modo AP.`

- Error formato: `[ESP32] Comando WebServer mal formado: falta 'label'`

6. GET

- Descripción: Realiza una petición HTTP GET a una URL externa y reporta código de respuesta y contenido.
- Parámetros:
  - `"url"`: URL completa para la petición.

Ejemplo JSON:

```
{ "cmd": "GET", "url": "http://example.com/api" }
```

- Mensajes UART:
  - OK: `[ESP32] GET http://example.com/api -> Código: 200` + contenido JSON (en una línea)
  - Error: `[ESP32] Error en GET -> Código: XXX`
  - Error no conectado: `[ESP32] No conectado a WiFi, no se puede hacer GET.`

7. HTML

- Descripción: Actualiza la página HTML servida por el WebServer.
- Parámetros:
  - `"html"`: string con contenido HTML.

Ejemplo JSON:

```
{ "cmd": "HTML", "html": "<h1>Nuevo título</h1>" }
```

- Mensajes UART:
    - OK: `[ESP32] HTML actualizado`
    - Error: `[ESP32] No se encontró el campo 'html'.`
    - Error: `[ESP32] Campo 'html' vacío.`

Mensajes generales

- Procesando comando recibido:
  `[ESP32] Procesando comando: <comando JSON recibido>`
- Error parseando JSON:
  `[ESP32] Error al parsear JSON: <mensaje error>`
- Error máximo endpoints alcanzado:
  `[ESP32] No hay espacio para nuevos endpoints`

Notas adicionales

- Para reiniciar el ESP32 se usa una señal en el pin `RESET_SIGNAL_PIN` que llama a `esp_restart()` o boton reset.
- La comunicación por UART imprime los mensajes con prefijo `[ESP32]` para distinguir mensajes del dispositivo.
- El servidor HTTP atiende endpoints dinámicos según los datos cargados con el comando `WebServer`.
- Los comandos pueden enviarse en texto plano o en formato JSON (el código actual maneja principalmente JSON).

Código de Prueba:

```python
import time
import board
import busio
import digitalio
import random
import json

class ESP32UART:
    def __init__(self, tx_pin, rx_pin, ready_pin, baudrate=115200, timeout=0.1):
        self.uart = busio.UART(tx=tx_pin, rx=rx_pin, baudrate=baudrate, timeout=timeout)
        self.ready_pin = digitalio.DigitalInOut(ready_pin)
        self.ready_pin.direction = digitalio.Direction.INPUT
        self.ready_pin.pull = digitalio.Pull.DOWN

    def esperar_ready(self, timeout=5):
        inicio = time.monotonic()
        while time.monotonic() - inicio < timeout:
            if self.ready_pin.value:
                return True
            time.sleep(0.01)
        return False

    def solicitar_comando(self, comando_dict):
        mensaje = json.dumps(comando_dict) + "\n"
        self.uart.write(mensaje.encode())

    def leer_respuesta(self, timeout=5):
        buffer = b""
        inicio = time.monotonic()
        while time.monotonic() - inicio < timeout:
            data = self.uart.read(64)
            if data:
                buffer += data
                while b'\n' in buffer:
                    linea, buffer = buffer.split(b'\n', 1)
                    try:
                        texto = linea.decode().strip()
                    except Exception:
                        texto = "<error decoding>"
                    if texto:
                        print(texto)
        if buffer:
            try:
                texto = buffer.decode().strip()
            except Exception:
                texto = "<error decoding>"
            if texto:
                print("Respuesta parcial:", texto)

def main():
    # --- Reset del ESP32 usando GP11 como pin temporal ---
    reset_pin = digitalio.DigitalInOut(board.GP11)
    reset_pin.direction = digitalio.Direction.OUTPUT
    print("Reiniciando ESP32-S3...")
    reset_pin.value = True
    time.sleep(0.1)
    reset_pin.value = False
    time.sleep(0.1)
    reset_pin.value = True
    time.sleep(1)
    reset_pin.deinit()
```

```
esp = ESP32UART(tx_pin=board.GP12, rx_pin=board.GP13, ready_pin=board.GP10)

html = """
<html><head><style>
body { background-color: #222; color: #fff; text-align: center; font-family: sans-serif; }
h1 { color: #4CAF50; }
</style></head><body><h1>Hola desde RP2040</h1></body></html>
"""

if not esp.esperar_ready(timeout=10):
    print("ESP32 no está listo.")
    return

# === PRUEBA DE COMANDOS ===

print("\n--- TEST: SCAN ---")
esp.solicitar_comando({"cmd": "SCAN"})
esp.leer_respuesta(timeout=15)

print("\n--- TEST: CONNECT ---")
esp.solicitar_comando({"cmd": "CONNECT", "ssid": "xxxx", "pass": "xxxx"})
esp.leer_respuesta(timeout=15)

print("\n--- TEST: HTML ---")
esp.solicitar_comando({"cmd": "HTML", "html": html})
esp.leer_respuesta(timeout=5)

print("\n--- TEST: GET (api) ---")
esp.solicitar_comando({
    "cmd": "GET",
    "url": "http://wifitest.adafruit.com/testwifi/index.html"
})
esp.leer_respuesta(timeout=10)
print("\n--- TEST: GET (api) ---")
esp.solicitar_comando({
    "cmd": "GET",
    "url": "http://jsonplaceholder.typicode.com/todos/1"
})
esp.leer_respuesta(timeout=10)

print("\n--- TEST: AP (crear punto de acceso) ---")
esp.solicitar_comando({
    "cmd": "AP",
    "ssid": "RP2040_AP",
    "pass": "clave1234"
})
esp.leer_respuesta(timeout=5)

print("\n--- TEST: DISCONNECT (WiFi) ---")
esp.solicitar_comando({"cmd": "DISCONNECT", "target": "WiFi"})
esp.leer_respuesta(timeout=3)

print("\n--- TEST: DISCONNECT (AP) ---")
esp.solicitar_comando({"cmd": "DISCONNECT", "target": "AP"})
esp.leer_respuesta(timeout=3)
print("\n--- TEST: CONNECT ---")
esp.solicitar_comando({"cmd": "CONNECT", "ssid": "xxx", "pass": "xxx"})
esp.leer_respuesta(timeout=10)

print("\n--- INICIANDO ENVÍO PERIÓDICO ---")
time.sleep(3)
```

```python
    # --- LOOP PERIÓDICO CON TEMPERATURA ---
    while True:
        if esp.esperar_ready(timeout=1):
            temperatura = round(random.uniform(20, 30), 1)
            esp.solicitar_comando({
                "cmd": "WebServer",
                "label": "Temperatura",
                "data": {"dato1": temperatura}
            })
            esp.leer_respuesta(timeout=2)
        else:
            print("ESP32 no listo.")
        time.sleep(5)

if __name__ == "__main__":
    main()
```

Consola:

```
--- TEST: SCAN ---
[ESP32] Escaneando redes WiFi...
SSID: NS_NWSN | RSSI: -67
SSID: NS_IOT | RSSI: -68
SSID: NS-Guest-USH | RSSI: -68
SSID: NS-Mobile | RSSI: -68
SSID: AT_401_RAC_056905_WW_62ad | RSSI: -69
SSID: DIRECT-A2-HP DesignJet T650 | RSSI: -73
SSID: Trackers | RSSI: -74
SSID: Othila_reefers | RSSI: -83
SSID: [LG_Wall-Mount A/C]5244 | RSSI: -85
SSID: ar-01 | RSSI: -87

--- TEST: CONNECT ---
[ESP32] OK|IP: 192.168.68.72

--- TEST: HTML ---
[ESP32] HTML actualizado

--- TEST: GET (api) ---
[ESP32] GET http://wifitest.adafruit.com/testwifi/index.html -> Código: 200
This is a test of Adafruit WiFi! If you can read this, its working :)

--- TEST: GET (api) ---
[ESP32] GET http://jsonplaceholder.typicode.com/todos/1 -> Código: 200
{   "userId": 1,   "id": 1,   "title": "delectus aut autem",   "completed": false }

--- TEST: AP (crear punto de acceso) ---
[ESP32] AP iniciado: RP2040_AP | IP: 192.168.4.1

--- TEST: DISCONNECT (WiFi) ---
[ESP32] WiFi desconectado.

--- TEST: DISCONNECT (AP) ---
[ESP32] AP detenido

--- TEST: CONNECT ---
[ESP32] OK|IP: 192.168.68.72

--- INICIANDO ENVÍO PERIÓDICO ---
[ESP32] Endpoint /Temperatura actualizado
[ESP32] Endpoint /Temperatura actualizado
```