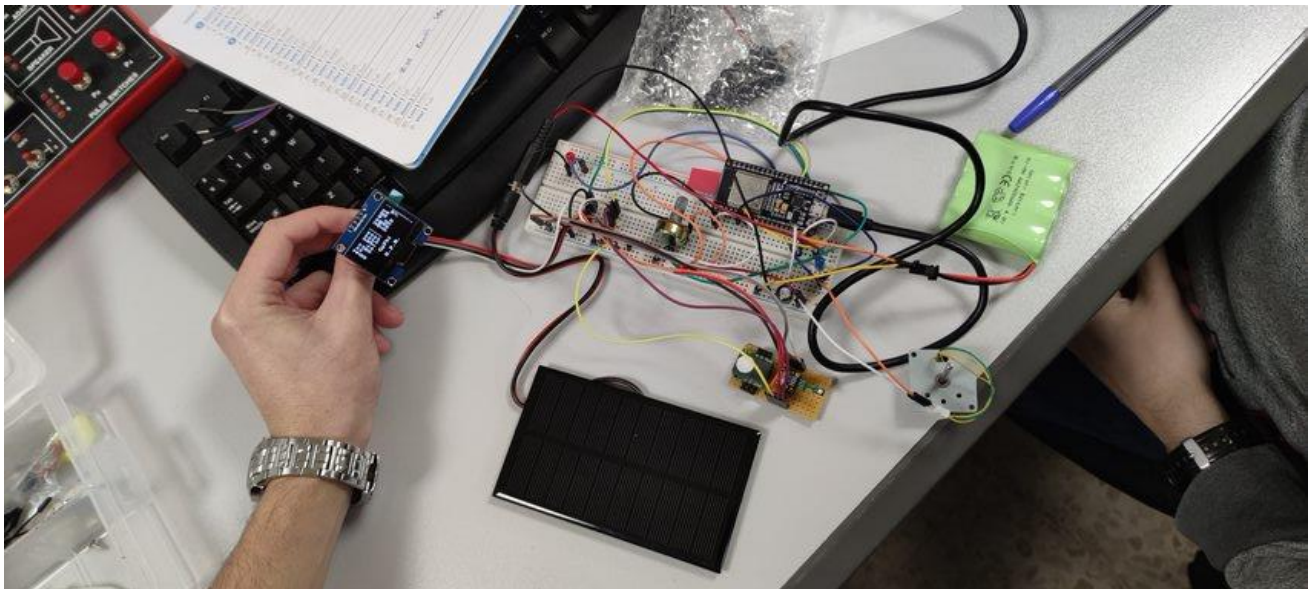


GENERADOR Y FILTRADOR HIDRÁULICO – GYFHI

SBC22M01

<https://upm365.sharepoint.com/sites/SBC22M01>



Miembros del equipo:

Serrano, Alejandro (coordinador)	alejandro.serranol@alumnos.upm.es
Carrasco, Ismael	ismael.carrascol@alumnos.upm.es
Parla, Raúl	raul.parla.mota@alumnos.upm.es
Riñón, Alejandro	alejandro.rinon.reneo@alumnos.upm.es

1.Introducción	4
1.1 Contexto	4
1.2 Trabajo relacionado	4
1.3 Objetivo del proyecto	5
2. Solución implementada: Descripción técnica del proyecto	9
2.1 Software	9
2.1.1 Código y librerías	9
2.1.1 Plataforma IoT	14
2.2 Hardware	21
2.3 Casos de uso	23
3. Demostración: Evaluación con/del sistema	26
4. Discusión y Conclusiones	33
4.1 Limitaciones y dificultades encontradas	33
4.2 Implicaciones, prospectiva y cómo extender el trabajo	33
Referencias	35
ANEXOS	38
Anexo I. Enlaces de documentos de SharePoint	38
Anexo II. Código github	40

Índice de figuras

Figura 1. Previsión de crecimiento de la población mundial («Población mundial», 2015)	4
Figura 2. 1er Boceto sobre la arquitectura esperada (elaboración propia)	6
Figura 3. Croquis sobre una segunda versión del chasis (elaboración propia)	7
Figura 4. Boceto sobre la arquitectura esperada (elaboración propia)	7
Figura 5. Primer nivel del panel (elaboración propia)	15
Figura 6. Segundo nivel del panel, sección A (elaboración propia). Nótese como uno de los widgets solo se usa para verificar que se ha sincronizado.	15
Figura 7. Segundo nivel del panel, sección B (elaboración propia).	16
Figura 8. Segundo nivel del panel, sección C (elaboración propia).	16
Figura 9. OTA Rule Chain (elaboración propia).	17
Figura 10. TeaSpike Rool Chain (elaboración propia).	18
Figura 11. Root Rule Chain (elaboración propia).	18
Figura 12. Respuesta del bot (elaboración propia).	19
Figura 13. Placa sensores utilizados (elaboración propia).	21
Figura 14. Diseño HW físico (elaboración propia)	23
Figura 15. Casos de usos (elaboración propia).	24
Figura 16. Imágenes OTA 1 (elaboración propia).	26
Figura 17. Imágenes OTA 2 (elaboración propia).	27
Figura 18. Imágenes OTA 3 (elaboración propia).	27
Figura 19. Imágenes I2C TempHum 1 (elaboración propia).	28
Figura 20. Imágenes I2C TempHum 2 (elaboración propia).	28
Figura 21. Imagen I2C luminosidad (elaboración propia).	28
Figura 22. Imagen detección ADC (elaboración propia).	29
Figura 23. Imagen detección switch activo en consola (elaboración propia)	29
Figura 24. Imagen display activo con switch a ON (elaboración propia).	30
Figura 25. Imagen detección switch inactivo en consola (elaboración propia)	30
Figura 26. Imagen display a clear con el switch a OFF (elaboración propia)	31

Índice de tablas

Tabla 1: Clasificación de artículos relacionados más relevantes (Elaboración propia).....	5
Tabla 2: Prueba inicialización (Elaboración propia).....	27
Tabla 3: Prueba toxicidad antes del filtro (Elaboración propia).....	28
Tabla 4: Prueba toxicidad tras el filtro (Elaboración propia).....	29
Tabla 5: Prueba sensores ADC básica (Elaboración propia).....	29
Tabla 6: Prueba display OLED (Elaboración propia).....	31
Tabla 7: Prueba display OLED (Elaboración propia).....	32

1.INTRODUCCIÓN

1.1 CONTEXTO

En los últimos años el precio del crudo y otros combustibles fósiles está en alza, y la guerra de Ucrania ha incentivado considerablemente a los gobiernos occidentales a encontrar y mejorar recursos alternativos para generar energía (D. Álvarez, 2022). Otra necesidad existente ha sido emplear una fuente de energía renovable y que no impactase en exceso al ecosistema, ya bastante dañado; y escapar de los abusivos precios de las eléctricas (RTVE.es / AGENCIAS, 2022).

El problema real a largo plazo es, sin embargo, la enorme contaminación medioambiental que la acción humana ha causado y causará, el efecto en la biosfera y todos los que habitan en ella, y la dependencia excesiva en los combustibles fósiles. Entre algunos de los efectos secundarios de nuestra actividad, destaca a nivel terrestre la infiltración de químicos y micropartículas nocivas (entre ellas micro plásticos (IBERDROLA, 2019)) en la hidrosfera, envenenando las tierras y haciendo de los recursos hídricos potables y terrenos libres de químicos un bien tremendamente valioso, detectándose la presencia de estas sustancias incluso en aguas tratadas para el consumo humano (National Geographic, 2022). Parte de este proceso de envenenamiento del suelo se produce cuando las emisiones contaminantes del aire precipitan al suelo durante la lluvia, o cuando se utilizan aguas ya contaminadas para regar (Fundación Aquae, 2021).

Con el alcance de la población humana del planeta por encima de los 8 mil millones (Flores, 2022) y en aumento (ver predicciones de la Figura 1: («Población mundial», 2015)), este último detalle es muy importante para el desarrollo sostenible debido a que la producción creciente de alimentos para consumo humano directo y ganadería se ven en riesgo de presentar diversos tóxicos y concentrarse en el consumidor humano final.



Figura 1. Previsión de crecimiento de la población mundial («Población mundial», 2015)

1.2 TRABAJO RELACIONADO

Actualmente existe un gran número de alternativas para este problema de forma (más) sostenible, destacando la captación de energía solar mediante paneles solares (IEEEExplore, 2022) y presas para generación de energía hidráulica con turbinas de Pelton, muy eficaces (endesa, 2021), y con respecto al aspecto de limpieza medioambiental también existen instrumentos que monitorizan la presencia de sustancias nocivas o agentes patógenos en el agua (IEEEExplore, 2022), así como mecanismos filtradores de estos como el "cubo de basura flotante" que filtra el agua de mar y filtros portables de agua como el LifeSaver (Barrie, 2015) para filtrar aguas contaminadas en países tercermundistas; visibles en la tabla adjunta (Tabla 1). Cabe destacar sin embargo que muchas de ellas tienen la debilidad de ser pasivas y no utilizar un controlador que informe de problemas o regule el mecanismo; o directamente no son portables (como en el caso de las presas).

Proyecto	Descripción	Fortalezas	Debilidades
Paneles Solares (IEEEExplore, 2022)	Los paneles solares captan la energía solar y la transforman en electricidad.	Energía verde del sol.	Su producción necesita de resinas epoxi que son tóxicas. Áreas comúnmente nubladas o poco iluminadas.
Turbinas Pelton de presas (endesa, 2021)	Este tipo de turbinase suele equipar a generadores de corriente alterna para transformar la energía hidráulica en energía eléctrica de forma muy eficiente.	Energía verde de los ríos y lagos	Suelen necesitar de una cantidad enorme de agua para funcionar. Para generar electricidad de forma significativa suelen estar en presas hidroeléctricas, demasiado grandes, de gran impacto medioambiental y paisajístico e inmóviles.
Detectores de toxinas (IEEEExplore, 2022)	Una aplicación por Android que comunica el nivel de pH, sólidos disueltos y temperatura inicial.	Portable, barato, bastante exacta, cubre parte de las necesidades del problema.	Solo detecta, no trata de resolverlo.
Cubo de basura flotante Seabin (EcoInventos, 2022)	Un aparato que recoge plásticos y detergentes del mar y los meter en una papelera.	Gran capacidad de filtrado, inteligente, cubre bien muchas de las necesidades mencionadas.	Solo filtra algunos tóxicos. No informa de lo que detecta.
LifeSaver (Barrie, 2015)	Filtro pasivo supereficiente que permite potabilizar el agua de muchos lugares contaminados.	Portable, barato, gran capacidad de filtrado.	No informa de lo que detecta, es un filtro pasivo. No está diseñado para filtrar grandes cantidades de agua en poco tiempo.

Tabla 1: Clasificación de artículos relacionados más relevantes (Elaboración propia).

1.3 OBJETIVO DEL PROYECTO

Además de la **reducir emisiones y generar energía limpia**, nuestro objetivo también reside en resolver los problemas de **filtración de agua y generación de energía de forma compacta** en un solo dispositivo portable, de tal forma que se pudiera incluir junto a fuentes de agua tales como zonas lluviosas y cascadas con bajo impacto al paisaje, cosas que medios como las presas hidráulicas no pueden, y con reparaciones más sencillas en caso de avería; tratando de demostrar que se pueden realizar este tipo de dispositivos mediante sistemas basados en computador **con poco coste y asequibles para todos**.

Para ello, nuestro equipo de estudiantes de ingeniería de la ETSISI ha decidido presentar el *proyecto GyPhi*, un sistema compuesto por un mecanismo hidráulico, en el que el agua proveniente de la lluvia, ríos o lagos se captaría en un embudo mallado (para prevenir la entrada de detritos) que por unos agujeros de varios tamaños en el fondo redirigiría el flujo hacia un sistema de múltiples turbinas que a su vez convertirían la energía mecánica en eléctrica; además de un panel solar como generador de suplemento. Tras las turbinas ya se pondría un colector que desembocara en un sistema de filtros que eliminaría diversos productos químicos y microplásticos para consumo agrícola y puede que incluso humano (Figura 2, el boceto con 4 turbinas de estrella, y Figura 3, la versión más desarrollada del chasis con turbinas de Pelton).

El sistema también dispondrá de un sistema de sensores químicos antes y tras los filtros para analizar el nivel de contaminación del agua y dar esta información a los usuarios mediante el ESP-32 (tanto en una página web básica, como por Telegram, como por un Display activable, como se puede ver en la Figura 4; en rojo se indican

los mocks no planeados originalmente en el diseño inicial) para determinar si esa agua puede ser utilizada/consumida o no, además de la energía producida y almacenada. En caso de detectar un tóxico antes del filtro además informará con un aviso luminoso y por el display, y si detecta tras el filtro, indicará con el filtro acústico además de informar por pantalla que el sistema de filtros requiere un recambio. Contribuimos al desarrollo sostenible de nuestro dispositivo imprimiendo en 3D la mayor parte de sus componentes.

El sistema de turbinas, utilizando turbinas de Pelton reducidas, estaría diseñado teniendo en cuenta alternadores, motores paso-a-paso (ya que generan energía con el menor número de revoluciones por minuto posible (George, 2012) o por defecto unas dinamos, y posicionado para maximizar la energía obtenida mediante la fuerza gravitatoria, con las turbinas un poco alejadas de los agujeros y entre sí.

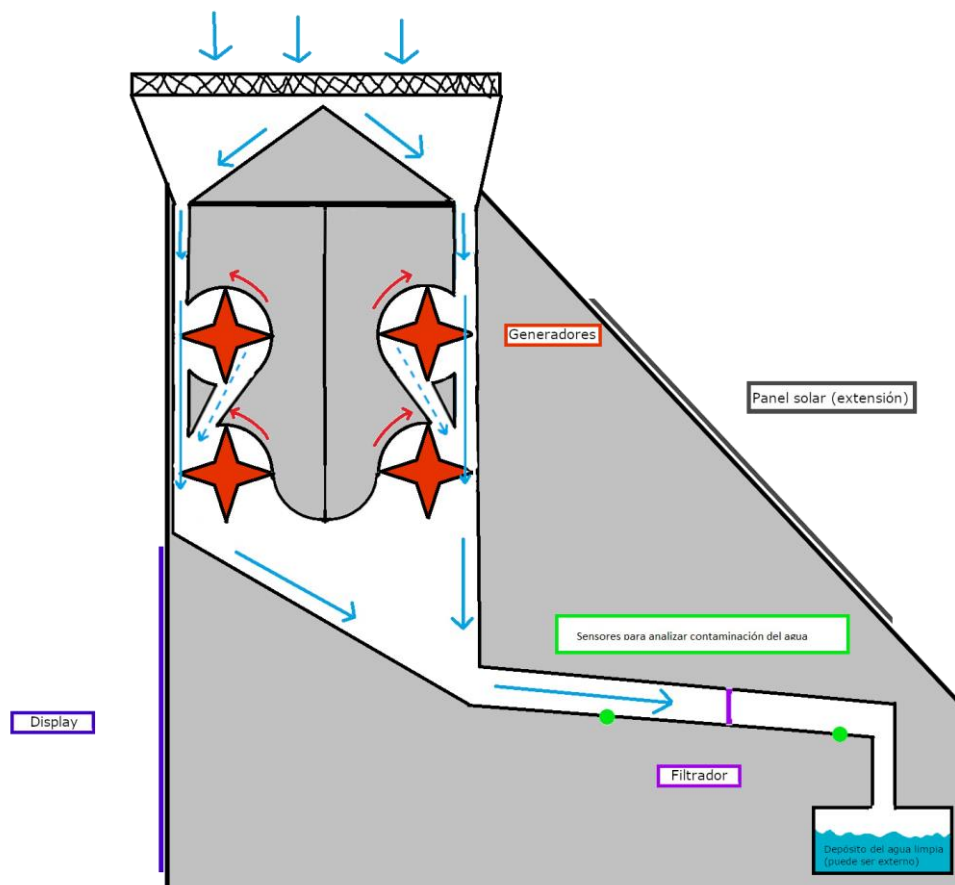


Figura 2. 1er Boceto sobre la arquitectura esperada (elaboración propia)

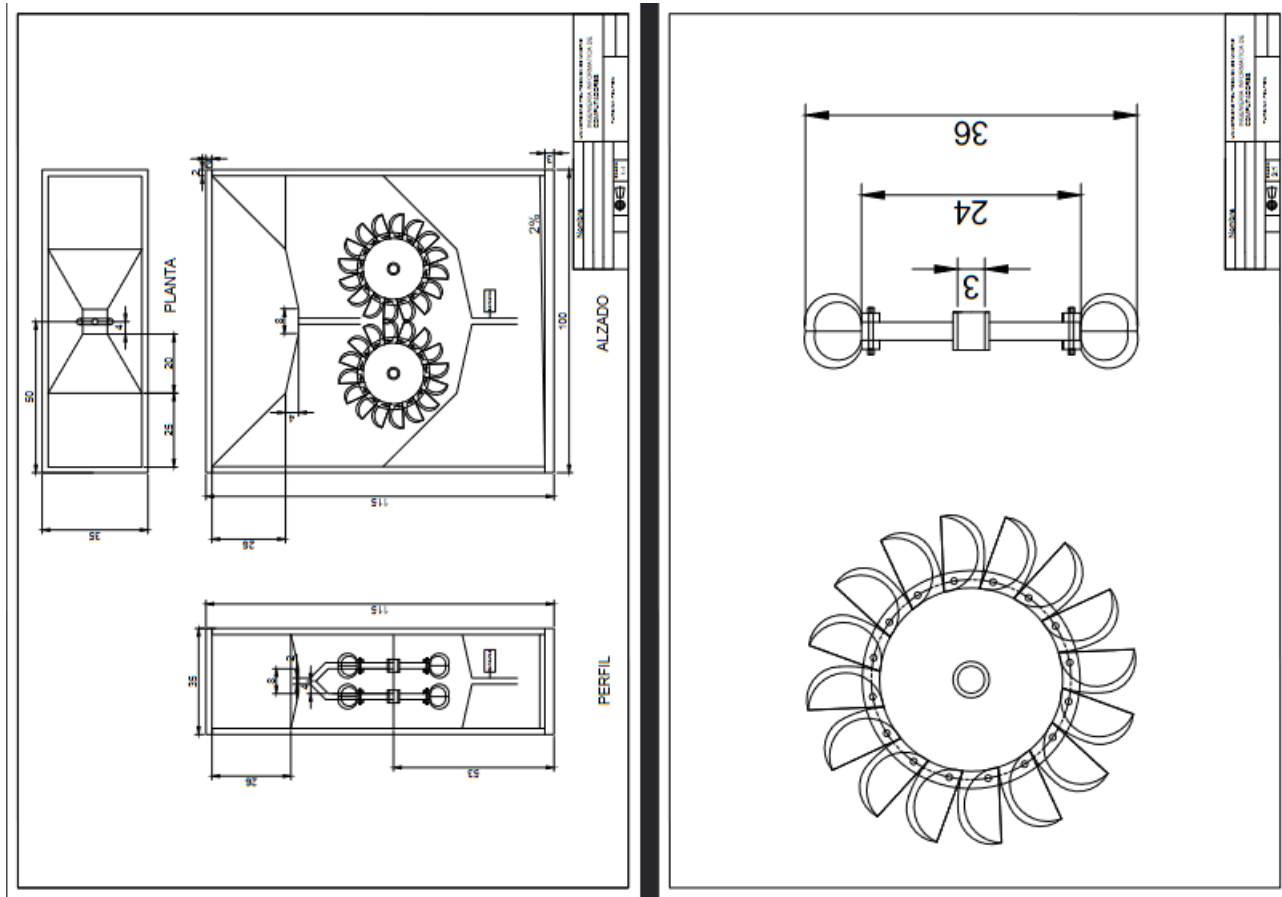


Figura 3. Croquis sobre una segunda versión del chasis (elaboración propia)

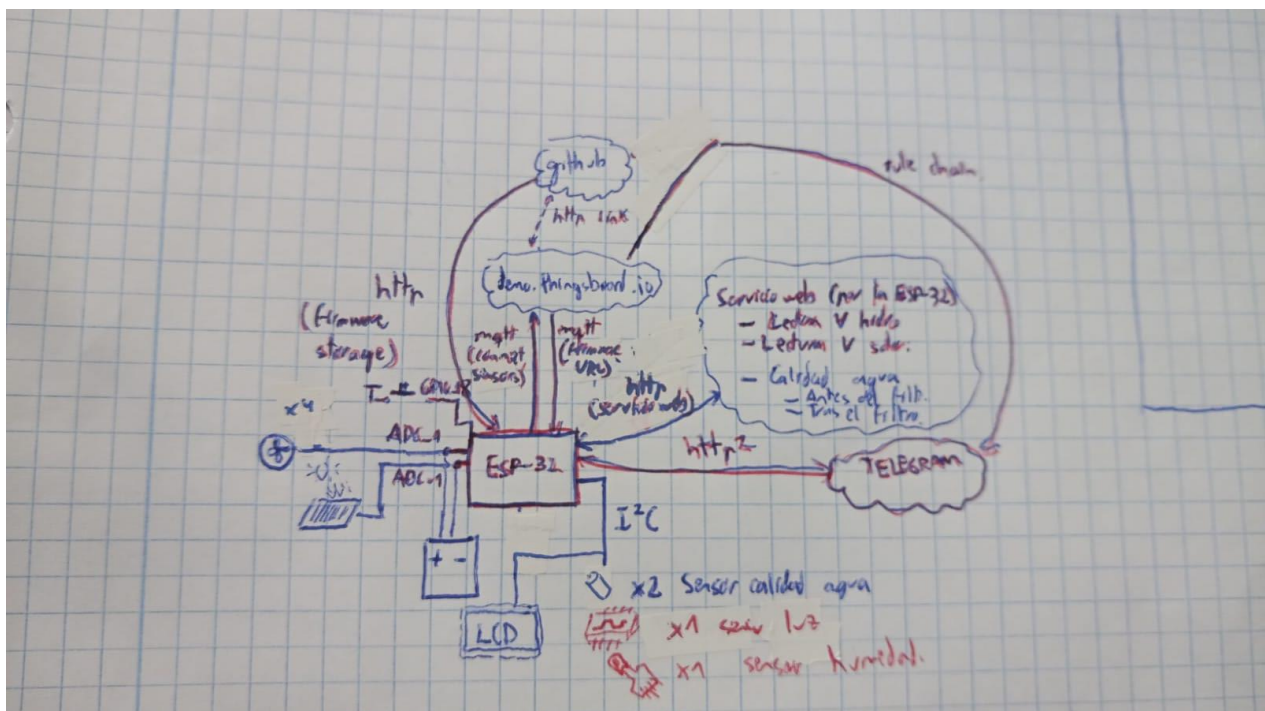


Figura 4. Boceto sobre la arquitectura esperada (elaboración propia)

2. SOLUCIÓN IMPLEMENTADA: DESCRIPCIÓN TÉCNICA DEL PROYECTO

2.1 SOFTWARE

2.1.1 CÓDIGO Y LIBRERÍAS

Se ha empleado exclusivamente C sobre VisualStudio Code para Espressif ESP-32 en todo el proyecto final – aunque es cierto que a veces se han comprobado programas de Arduino para entender su funcionalidad (arquero99, 2022).

- Aparte de las librerías del esp-32 nativas, solo se emplearon la librería del componente "sh2lib.h", que se emplea en el código de ejemplos del http2 de Espressif para simplificar el uso de la API del nhttp2; y el especial "<wifi.h>" para la esp-32 ota, que estaba también en los ejemplos para Thingsboard que simplificaba el guardar en y recuperar de la memoria flash la configuración del wifi; y el componente "ssd1306_i2c", que nos simplifica con creces el uso del display mediante I2C y SPI (aunque al final no empleamos el SPI y tuvimos que comentar el i2c_master_init por conflictos con el de otros dispositivos). También nos basamos en los widgets de esp32 del tutorial y los widgets y rule chains de teaSpike y las modificamos (las originales pueden encontrarse en el gitHub) para que nos permitieran actualizar el firmware y para poder enviar mensajes al Telegram y alertas.

main.c (simple)

```
• #include <esp_wifi.h>
• #include <esp_event.h>
• #include <esp_log.h>
• #include <esp_system.h>
• #include <nvs_flash.h>
• #include <sys/param.h>
• #include "esp_netif.h"
• #include "esp_eth.h"
• #include "protocol_examples_common.h"
•
• #include <esp_https_server.h>
• #include "esp_tls.h"
•
• // Del LED
• #include <stdio.h>
• #include "driver/gpio.h"
• #include "led_strip.h"
• #include "sdkconfig.h"
• // Fin de includes del LED
•
• /*Del ADC específicamente*/
• #include "driver/adc.h"
• #include "esp_adc_cal.h"
• // Fin de la seccion ADC
•
```

```
• // Del reseteo de fabrica
• #include "esp_partition.h"
• #include "esp_https_ota.h"
• #include "esp_ota_ops.h"
• #include "esp_log.h"
•
• // MQTT
•
• #include "lwip/sockets.h"
• #include "lwip/dns.h"
• #include "lwip/netdb.h"
•
• #include "mqtt_client.h"
•
• #include <stdint.h>
• #include <stddef.h>
• #include "esp_wifi.h"
• #include "esp_system.h"
• #include "nvs_flash.h"
• #include "esp_event.h"
•
• #include "freertos/FreeRTOS.h"
• #include "freertos/task.h"
• #include "freertos/semphr.h"
• #include "freertos/queue.h"
•
• // Aniadir libreria cJSON
• #include "cJSON.h"
•
• // Para I2C
• #include "driver/i2c.h"
• // Para i2c test
• // #include "cmd_i2ctools.h"
•
• // Para mensajes genéricos
• #include <string.h>
•
• // Para Telegram
• #include <stdlib.h>
• #include <sys/time.h>
• #include "lwip/apps/sntp.h"
•
• #include "sh2lib.h"
•
• // Para los sleep
• #include <time.h>
• #include "soc/soc_caps.h"
```

```

• #include "esp_sleep.h"
• #include "driver/rtc_io.h"
• #include "soc/rtc.h"
• #include "esp32/ulp.h"
• #include "driver/uart.h"
•

```

sh2lib.c y sh2lib.h

```

• #include <stdlib.h>
• #include <stdint.h>
• #include <stddef.h>
• #include <stdio.h>
• #include <string.h>
• #include <unistd.h>
• #include <ctype.h>
• #include <netdb.h>
• #include <esp_log.h>
• #include <http_parser.h>
•
• #include "esp_tls.h"
• #include <nghttp2/nghttp2.h>

```

wifi.h (esp-32 ota)

```

• #include <string.h>
• #include <sys/param.h>
•
• #include "wifi.h"
• #include "main.h"
•
• #include "esp_event_loop.h"
• #include "esp_log.h"
• #include "esp_system.h"
• #include "esp_wifi.h"

```

Componente ssd1306 (librerías adicionales)

```

• #include "driver/spi_master.h" // No se usa
• #include "font8x8_basic.h" // Incluye como escribir texto en el display

```

- Código del proyecto: se incluyen (obviamente no se mencionan los sdkconfig aunque sean necesarios y de ahí se saquen algunas configuraciones del programa principal; pues se ajustan a las opciones de la placa, en nuestro caso 4MB con particiones ota):
 - Firmware que realiza las funciones reales del proyecto (carpeta “simple”). Aquí se incluyen
 - Componente sh2lib. Indicado anteriormente.
 - Componente ssd1306_i2c; para el display mencionado anteriormente. Modificado para no utilizar su i2c_master_init, sino el nuestro.
 - Componente main.c: el programa principal. Aunque lo hemos organizado para tener los include primero, las variables después, los cuerpos de las subrutinas y tareas en tercer lugar y luego el main en sí, con los certificados para el servidor http y el http2

de Telegram por separado en la subcarpeta certs, y algunas configuraciones por defecto establecidas en el archivo Kconfig.projbuild, el programa se puede dividir en ciertas partes importantes para cada código (indicándose en el anexo II):

- **Secciones LED-switch y comunes:** La usamos principalmente para debug, pero nos sirve también para activar el display y contiene tanto variables que diferentes secciones utilizan (como el voltajeHidro) como el programa principal que lo inicia todo. Este llama primero llama a los inits del sistema de sleep, pines básicos input/output e i2c y la calibración del adc, luego al wifi, posteriormente al cliente mqtt, y los servidores http2 y http con diferentes tasks; y por último un bucle infinito que por polling comprueba el estado del switch del display (y lo enciende en consecuencia), toma lecturas de las medidas analógicas de generación de energía hidráulica y solar y de los tóxicos del I2C y procede a enviar los datos por mqtt a Thingsboard. Tras repetir esto varias veces entra en light sleep por un minuto, del que solo puede despertar antes de tiempo si se activa el botón del display (cabe notar sin embargo que en la versión final por decisión del equipo los sleep quedan desactivados).
- **Secciones I2C básicas:** configuran los sensores I2C básicos como nuestros mock de los sensores de toxinas antes (sensor de temperatura y humedad) y tras (sensor de luminosidad) el filtro, así como los comandos básicos que le pedimos nos responda del Telegram, como el que se usa para devolver los datos pedidos, que en algunos casos es más complejo de lo esperado y resulta útil introducir en una función propia. También se incluye el código de un sensor de CO2 sencillo que, aunque correcto nuestro ESP32 no soporta ya que no tolera bien la función de clock stretching y siempre acaba en timeout (error 0x107).
- **Secciones ADC:** simplificamos el uso del único módulo ADC que podemos tener disponible a la vez que el wifi. En los anexos se indican las secciones específicas que guardan las variables interesadas, así como la calibración. Sin embargo la lectura en sí como ya fue mencionado se realiza en el bucle del programa principal.
- **Secciones MQTT:** su papel es comunicarse de y hacia el servidor demo.thingsboard, aunque la función de recepción no se usa en este programa sino en el de la esp32-ota. Aparte del init, la función llamada cerca del final del bucle del programa principal transforma los datos de las lecturas de los sensores en un json que envía al panel de Thingsboard.
- **Secciones http del servidor básico:** estos soportan una página web básica accesible por red de área local (en este caso la red SBC) que nos muestra los mismos datos que se envían al Thingsboard; refrescándose cada 10 segundos; y también proporciona acceso a un botón para reiniciar la ESP32 a la partición de fábrica y otro para comprobar físicamente si responde (que nos permite encender un LED de debug).

- **Secciones http2 de Telegram:** se explicarán más adelante en la sección del sistema IoT, pero en resumen mediante la biblioteca sh2lib ya mencionada tras iniciar la conexión se realiza un bucle en el que hacen peticiones GET a la página getUpdates del robot (con una ID que hemos decidido no incluir en el código final) con un cierto offset y al finalizar la petición se toma el dato, se desglosa en formato JSON, se actualiza el offset y se toma el contenido de los mensajes del chat autorizado para ejecutar comandos (la referencia a ellos se puede ver en los anexos). Esos comandos llaman a otra función GET (al investigar el SW Arduino de arquero99, descubrimos que para peticiones simples el POST no era necesario) que espera a que se reciba otra respuesta antes de cerrar y volver a preguntar unos segundos después. El bucle tan corto se hizo para garantizar una respuesta rápida a los mensajes del Telegram, y el offset se hizo por eficiencia para que solo tuviera que cargar todos los mensajes en la bandeja de entrada 1 vez. La parte de emisión de alertas se deja en el panel del Thingsboard y sus reglas.
- **Secciones de retorno de fábrica:** su papel es retornar a la sección que tiene el software de la ota, para que busque actualizaciones
- **Secciones de sleep:** su papel es regular el tipo de sleep que nuestro SW tiene, lo hemos establecido para ser un light sleep tras un período de tiempo o si se activan algunos de nuestros periféricos (si fuera profundo el offset del http2 de Telegram se perdería y en nuestro caso queremos mantenerlo simple sin tener que leer de muchos ficheros internos y sin que accidentalmente el valor guardado fuese demasiado alto como para obtener nada).
- SW que utilizamos como ota (carpeta “ota”). Como este código es muy similar al código del ejemplo del Tutorial del Thingsboard (excepto en dos switches de mqtt donde faltaban unas opciones) no se destacará nada importante en las líneas de código, solo que toma la versión del firmware y la url según el token de dispositivo grabado en la memoria de la placa, pidiéndole por mqtt la url de descarga y la versión que tiene el Thingsboard, y si la versión de la placa está desactualizada, procede a iniciar mediante opciones de wifi también indicadas en la memoria de la placa una comunicación http para descargar el nuevo firmware.
- Link al Panel de Thingsboard.
- Widgets/Rule chains importados de terceros originales.
- Una copia de este documento en formato .docx

2.1.1 PLATAFORMA IOT

Persistencia

A nivel del esp32 ota, utilizamos el device “ESP32 v2” (con token de acceso “YSRNEFDXnyIGhX9OayIG”) que cargará del Thingsboard la versión almacenada en la url que el widget de la ota almacena siempre y cuando su versión del firmware esté desactualizada (a comparación con la del widget, es un número menor). Esta conexión requiere de tener el certificado de la url de la que descarguemos la ota (en nuestro caso github).

Una vez ha cargado el firmware, ese mismo “ESP32 v2” genera un json que almacena los datos de energiaSolar (en mV), energiaHidraulica (en mV) desde el ADC1 ambos de 0V a 4V, toxicidad antes del filtro (como “co2I2C”), toxicidad tras el filtro (como “luzI2C”), ambas en un mock de partes por millón (ppm) hasta 20000 y por I2C; y si el switch de activación del display está activo “botonDisplay” (que no tiene unidades, simplemente manda 1 o 0 pues es entrada digital). Esta conexión se realiza mediante mqtt y requiere de tener el token del dispositivo del thingsboard para autenticarse.

Las mediciones se realizan cada 5 segundos mínimo para prevenir interferencias y dar tiempo a que se actualize, y porque realmente no necesitamos de actualizar los datos con alta frecuencia.

Dashboards o paneles de visualización de datos

El sistema de paneles tiene 2 niveles. El primero muestra la lista de acceso al dispositivo público de la ota, indicando si el dispositivo se encuentra conectado y sincronizado, respectivamente (Figura 5).

El segundo nivel indica detalles de la ota y de los sensores. En primer lugar tenemos la sección A, que indica versión del firmware y la url de descarga, así como de nuevo el estado de conexión y sincronización (Figura 6). Posteriormente tenemos un diagrama de tiempos que indican los resultados a lo largo del tiempo, así como los niveles de energía producida en un instante dado (el indicador analógico es el hidráulico y la barra digital el manual) (Figura 7). Ya al final se tienen un indicador del estado del switch del display, y un widget de alertas que responde con la rule chain TeaSpike. La sección A usa los widgets del tutorial del esp-32, mientras que el resto fueron creación nuestra puramente.

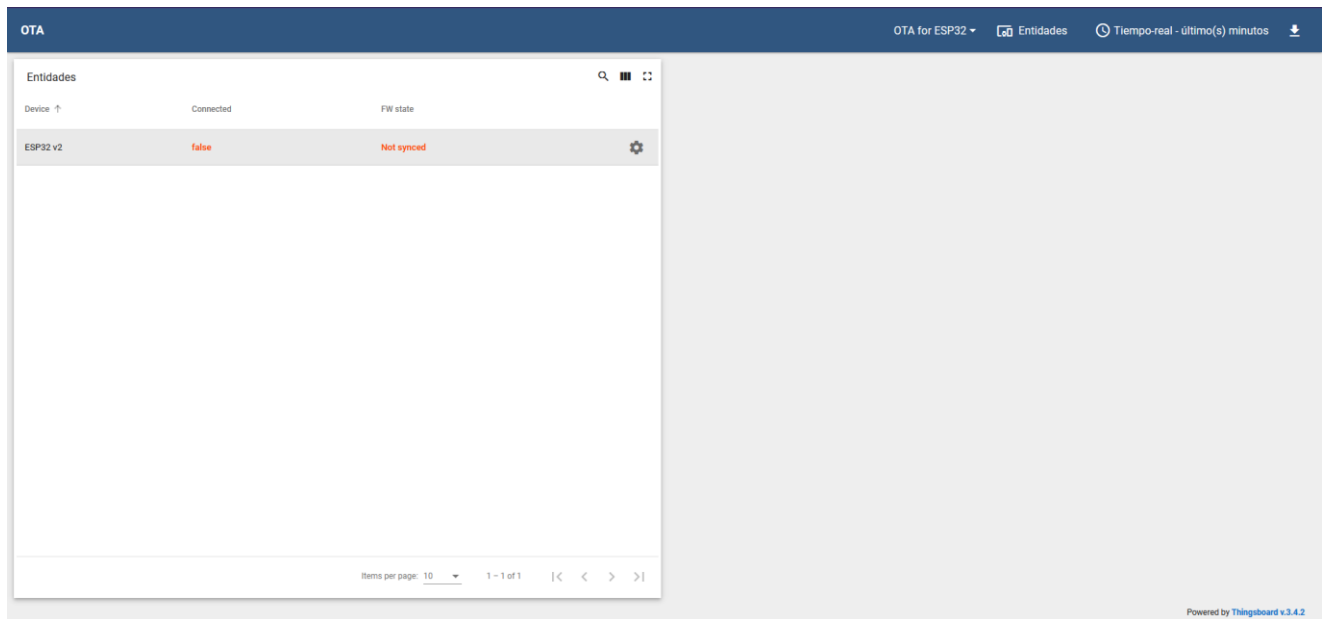


Figura 5. Primer nivel del panel (elaboración propia)

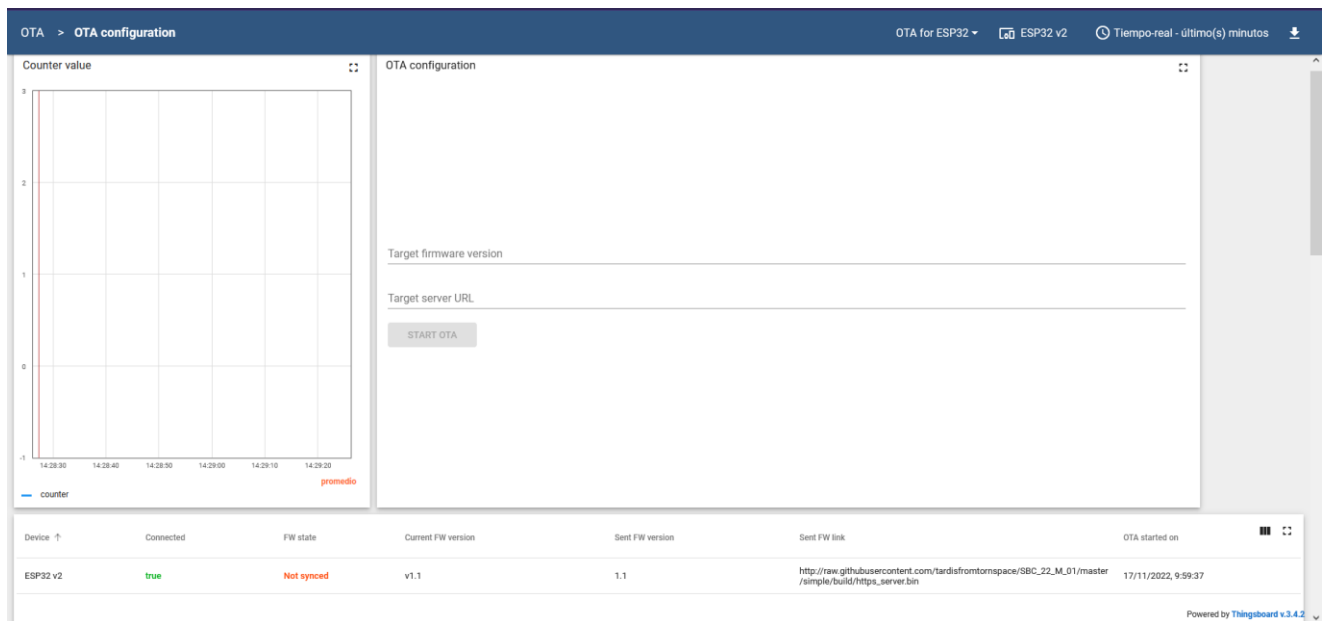


Figura 6. Segundo nivel del panel, sección A (elaboración propia). Nótese como uno de los widgets solo se usa para verificar que se ha sincronizado.



Figura 7. Segundo nivel del panel, sección B (elaboración propia).

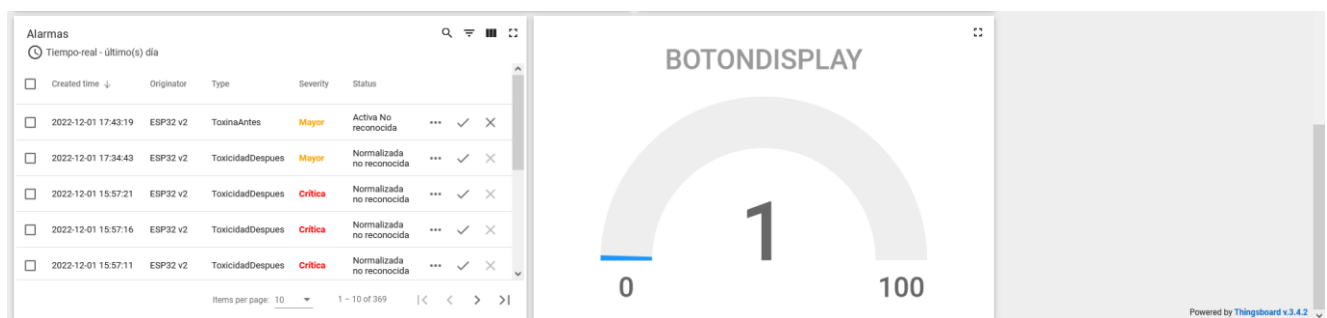


Figura 8. Segundo nivel del panel, sección C (elaboración propia).

Reglas

Se han implementado 2 cadenas de reglas que se insertan en la Root Rule Chain. Cabe notar que nosotros no introducimos una comprobación de nombre en la versión final ya que solo disponemos de un dispositivo en el perfil:

-ESP32 is synced: esta simplemente hace que se pase a la ESP32 por mqtt la url hacia la que descargar por https, y posteriormente comprueba que tienen la misma versión del firmware instalada (Figura 9).

-TeaSpike: acá discriminamos las señales recibidas de toxinas antes y después de filtro, el voltaje generado en total por las turbinas y la placa solar y el estado del switch del display (al fin y al cabo que este switch esté activo indica que alguien está manipulando el dispositivo *in situ*, escenario 0), de tal forma que de un aviso de si el filtro necesita reemplazarse o está roto (escenario 1), así como diversas alertas de mayor prioridad si se detectan niveles altos de toxinas (escenario 2 para antes del filtro,

escenario 3 tras el filtro, naturalmente si es tras el filtro se da una alerta mayor, ya que nos esperamos que el agua antes del filtro no esté tratada), y si de media se genera insuficiente energía (por debajo de los 3.3V, caso de uso 4) o demasiada energía (por encima de los 6V, caso de uso 5) (Figura 10). En caso de una alerta se comunica mediante un bot de Telegram previamente creado, para lo que requerimos de una id del bot y dos scripts, el primero en azul crea el mensaje y el segundo en naranja utiliza la API para enviarlo. En caso contrario limpia las alertas.

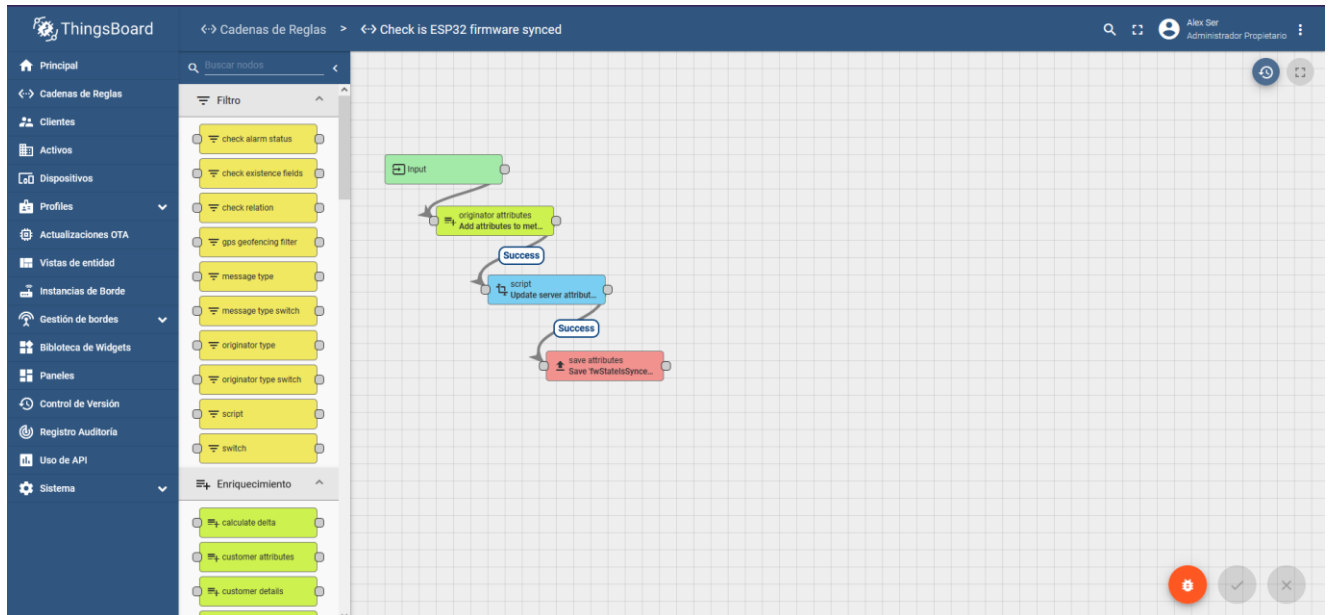


Figura 9. OTA Rule Chain (elaboración propia).

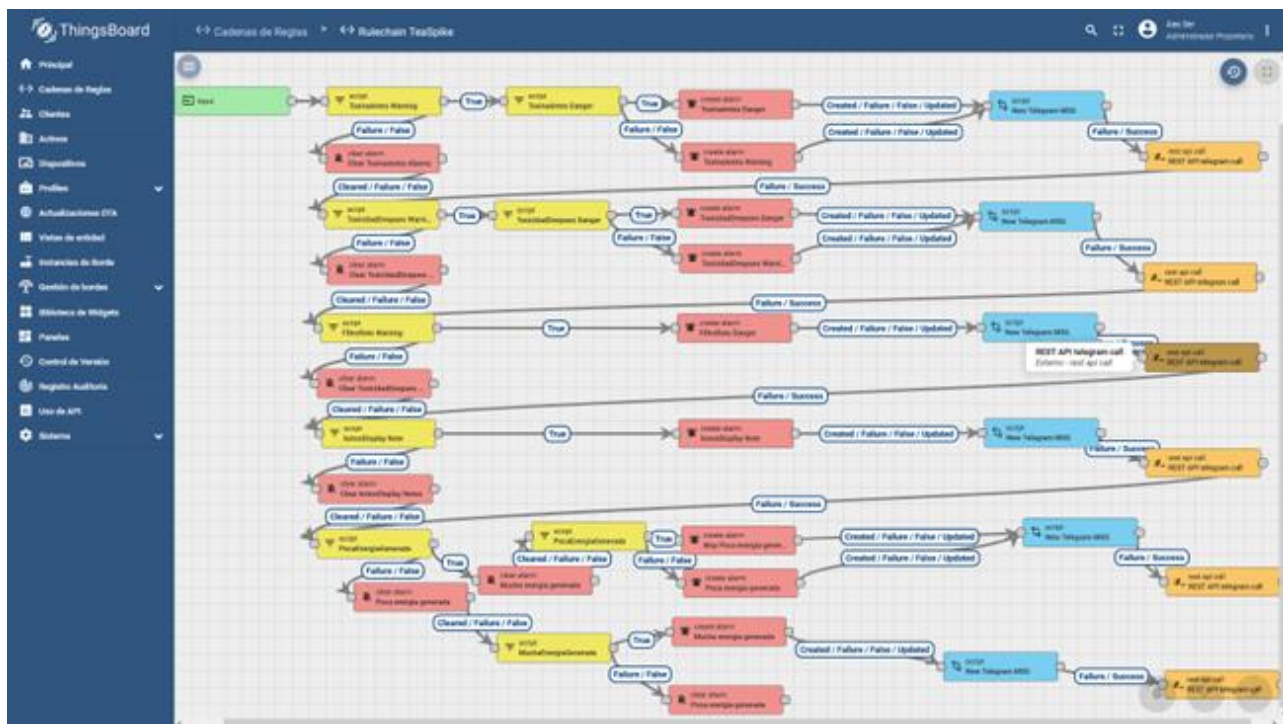


Figura 10. TeaSpike Rool Chain (elaboración propia).

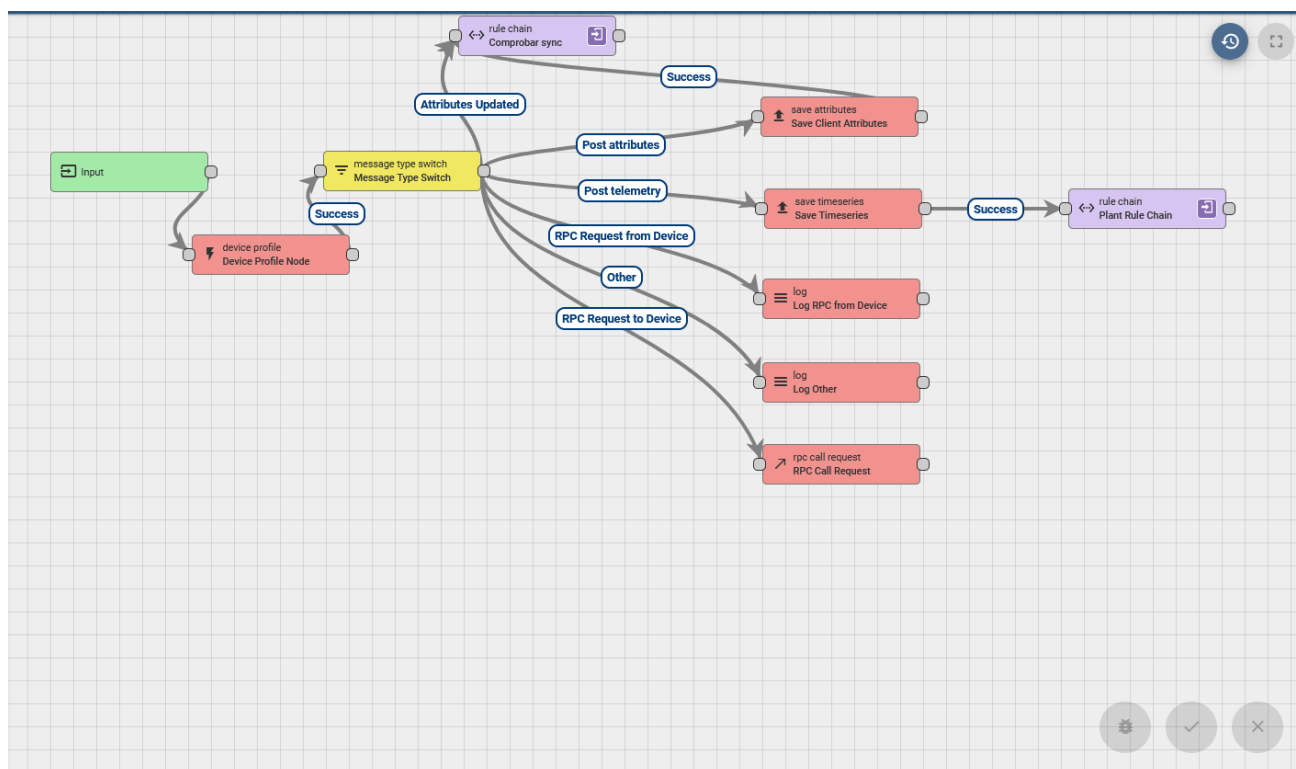


Figura 11. Root Rule Chain (elaboración propia).

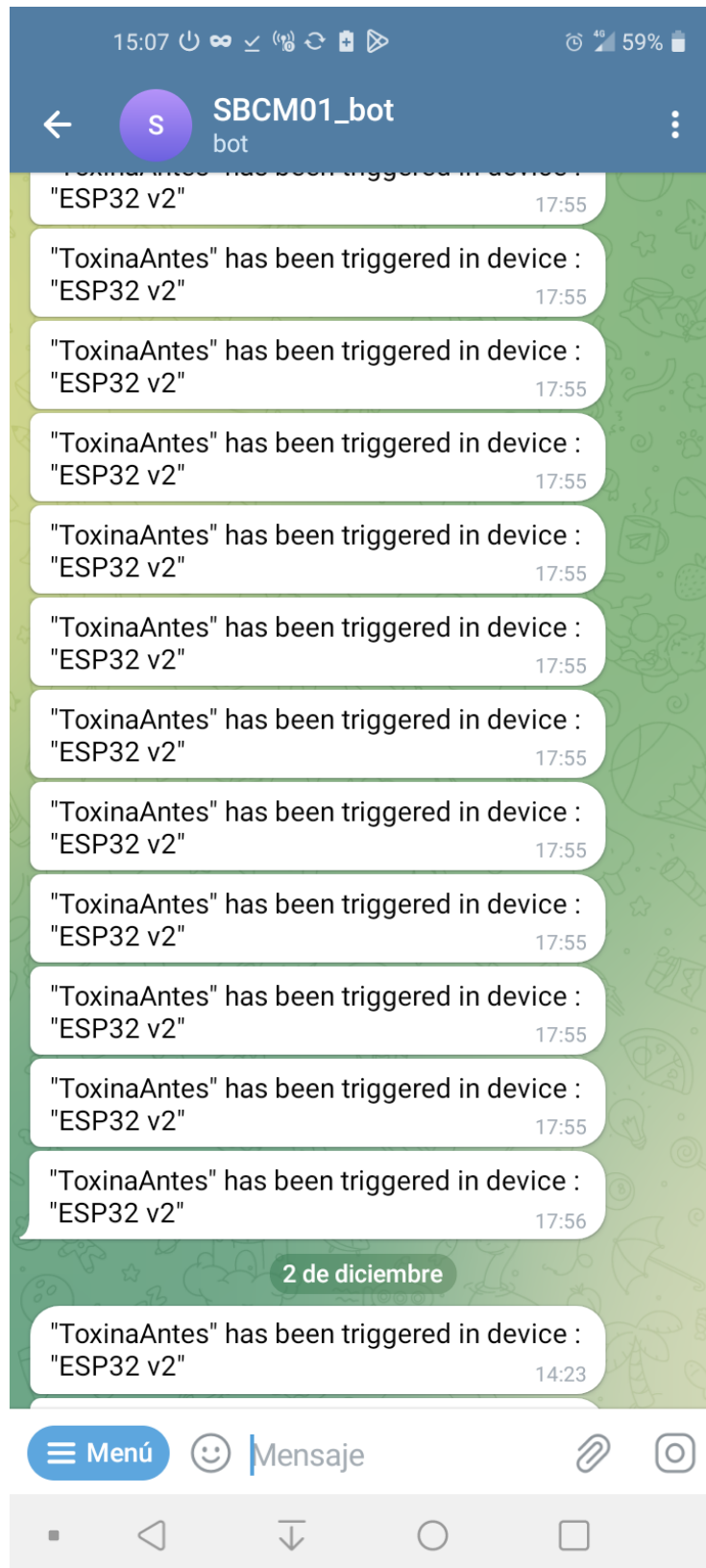


Figura 12. Respuesta del bot (elaboración propia).

Adicionalmente, mediante la librería del ejemplo http2 de Espressif "sh2lib.h", se simplifica el envío y recepción de mensajes del Telegram. Mediante el código de dicho ejemplo, se controla el manejador que se realiza tras recibir una respuesta para transformar los datos en JSON, se divide en mensajes, se seleccionan los mensajes con contenido y se emitirá una respuesta con respecto a dicho contenido. Entre los comandos (disponibles en el anexo II) están:

- /saluda – responde al chat con Hola Mundo.
- /myId – manda la id del que pregunta.
- /restartPlaca – hace que la placa vuelva a versión de fábrica tras un tiempo.
- /datos – devuelve al chat la energía solar e hidráulica generada, así como los tóxicos antes y tras el filtro y si el botón de encender el display está activo.
- Comandos de preguntas de plantas (por petición del profesor).

2.2 HARDWARE

Esta sección se refiere a los componentes puramente HW. Para ver las piezas del chasis y turbinas impresas en 3D, consultar Anexo I.

Entre los dispositivos físicos conectados a la ESP32, tenemos un interruptor, un sensor I2C de Humedad y temperatura (que simula las toxinas post-filtro), un sensor I2C de luminosidad (que simula las toxinas antes del filtro), dos generadores de electricidad (en nuestro caso, dos dinamos del sistema hidráulico y una placa solar) conectadas a una batería y a dos pines de módulo ADC1, un Display OLED por I2C para mostrar resultados cuando el interruptor mencionado se encuentra activo, un LED de debug y el módulo wifi que la ESP32 tiene por defecto para comunicarse por internet.

1.-Configuración:

En nuestro proyecto podemos encontrar diferentes sensores situados en una misma placa:

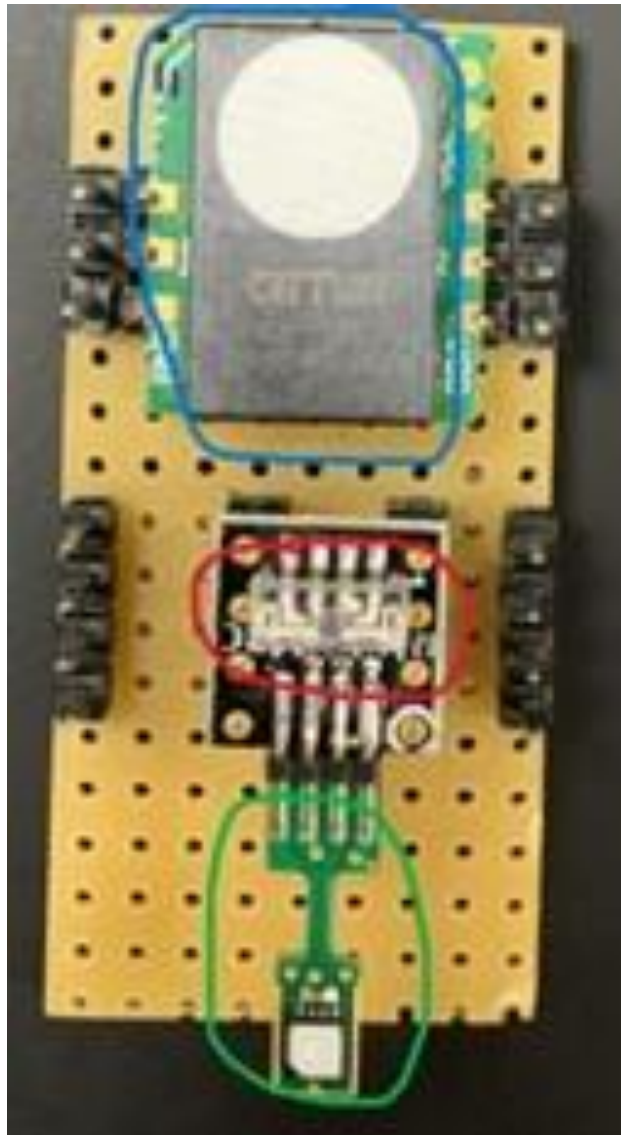


Figura 13. Placa sensores utilizados (elaboración propia).

Marcado por colores encontramos los siguientes sensores:

-(AZUL) Sensor de CO2: IAQ-Core C (rs-online, 2022).

-(ROJO) Sensor de luminosidad: VEML7700-TR (rs-online, 2022).

-(VERDE) Sensor humedad y temperatura: Sensirion SHT85 (sensirion.com, 2022).

De estos tres sensores utilizamos el rojo y verde que nos recogen diferentes datos y valores que son mostrados por nuestros actuadores, entre los cuáles está el Display OLED SH1106 (Az-delivery, 2022) que también nos muestra los valores de energía generada por los motores de nuestro prototipo.

Por otro lado, cabe destacar que el sensor azul, sensor de CO2, al comunicarse por I2C como el resto de los sensores del montaje de la imagen anterior, logramos que funcionara, pero nos dio problemas ya que siempre obteníamos un valor 0x00 de este. De primeras pensamos que fue por el tiempo de refresco que este necesitaba, ya que hasta que mostraba datos normales debían de pasar unos 5 minutos aproximadamente, pero tras esperar este tiempo el sensor nos seguía devolviendo el mismo valor descrito anteriormente, por tanto y tras probar otro sensor del mismo tipo para comprobar si era defectuoso (que también fallaba), descubrimos que nuestra ESP32 no soporta clock stretching adecuadamente y decidimos cambiar de sensor usando el sensor de humedad y temperatura del montaje que tenemos.

Tras la consulta de los diferentes datasheets de los sensores y las comprobaciones con el multímetro logramos averiguar que pines correspondían a cada sensor. El sensor de CO2 tiene los 6 pines que le rodean, dejando dos de ellos sin conectar por recomendación del datasheet. Al sensor de luminosidad (rojo) le corresponden los pines de la derecha con el orden de abajo a arriba (1. SCL 2. VDD 3. GND 4. SDA). Por último, al sensor de humedad/temperatura (verde) le corresponden los pines de la izquierda ordenados de arriba a abajo (1. SCL 2. VDD 3. GND 4. SDA).

2.-Esquema físico: como se ve, hemos empleado los pines 34 y 32 para las conexiones analógicas de los voltajes de los dos generadores solar e hidráulico; no escogimos el 35 porque por algún extraño motivo emplear dicho pin podía causar que las lecturas analógicas se distorsionaran sin motivo, además se conectan a la batería con un par de diodos para evitar que la batería alimentase a los generadores, y con un condensador de 125 uF (tras consultar el manual técnico y el datasheet del ESP32, calculamos suponiendo que a los 100 ms queremos que el voltaje esté por encima del mínimo de 1.8V y 0.5A (Espressif, 2022), en nuestro caso 2V y 0.5A). Al final utilizamos el pin 2 para el switch del display, con la funcionalidad adicional de que al mantenerlo activo podamos prevenir que se actualizase la memoria flash en físico. Además, utilizamos 2 de los 4 posibles pines recomendados por el manual técnico del ESP32 para el uso del bus I2C, G4 para SCL y G15 para SDA (Espressif, 2022), con resistencias de pull-up de 4K7 ohmios por recomendación de los fabricantes de los sensores de CO2 (AMS.com, 2022), luminosidad (VISHAY, 2022) y humedad y temperatura (Sensirion, 2022).

Originalmente habíamos tratado de utilizar el sensor de CO2 para simular el sensor adecuado, pero como ya mencionamos anteriormente, descubrimos que siempre hacía time-out. Eso era porque dicho sensor requería de clock stretching, una función que al parecer la ESP32 que empleamos no soporta adecuadamente. Procedimos a utilizar el sensor de humedad y temperatura en su lugar.

Por otra parte, la configuración del display OLED no tuvo mucho problema ya que funciona por I2C y la implementación de esto ya la teníamos resuelta gracias a los otros sensores, lo único que tuvimos que hacer

fue incorporar a nuestro código SW las librerías y componentes propios de este display encontrados en “ssd1306.h” y “font8x8_basic.h”, configurándolos a nuestra manera ya que teníamos pines diferentes, y hacer las conexiones físicas para probarlo y ver que funcionaba correctamente.

Por último, cabe destacar que hemos añadido dos diodos conectados a panel solar y dinamo para que las energías generadas por cada elemento no se compartan entre sí y sean independientes, de este modo evitaríamos errores de malas lecturas.

Tras las aclaraciones anteriores nuestro diseño físico queda de esta manera:

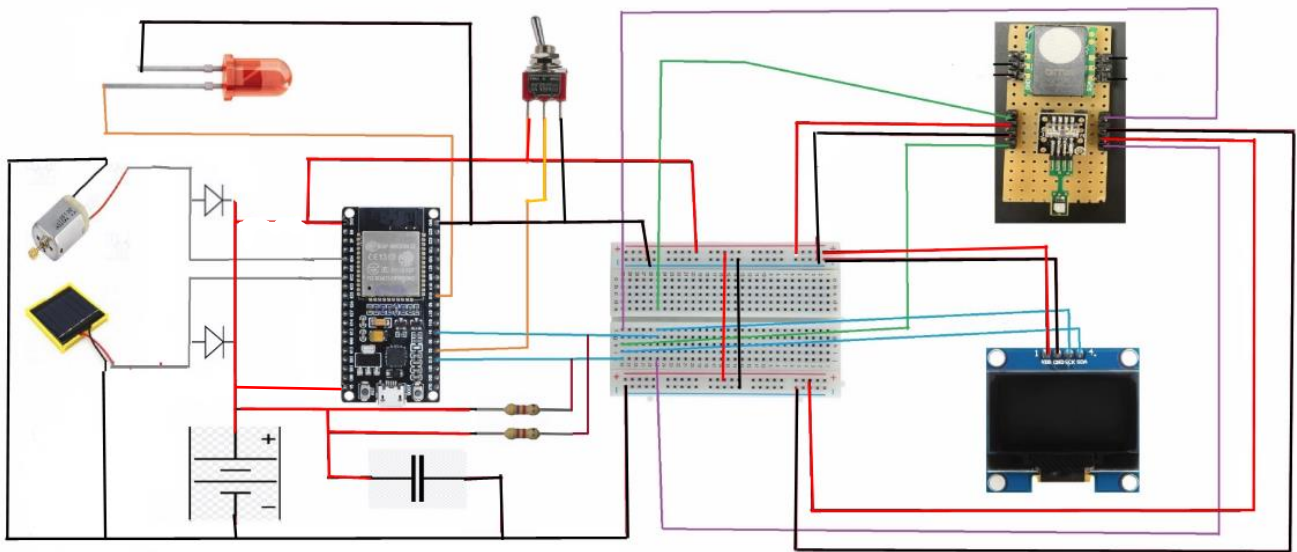


Figura 14. Diseño HW físico (elaboración propia)

2.3 CASOS DE USO

Operadores y usuarios:

- cliente Telegram: tiene rol de privilegio medio, y tiene ciertos privilegios también
- cliente http: el de la página web
- cliente físico: es el que toca el botón del display para chequear, solo puede manipular la ESP32 físicamente en l ode activar/desactivar switch, tiene pocos privilegios.
- Usuario Thingsboard.

Caso de uso 0: se enciende el switch del display.

Caso de uso 1: filtro roto.

Caso de uso 2: alta toxicidad antes del filtro.

Caso de uso 3: alta toxicidad tras el filtro.

Caso de uso 4: poca energía generada.

Caso de uso 5: mucha energía generada.

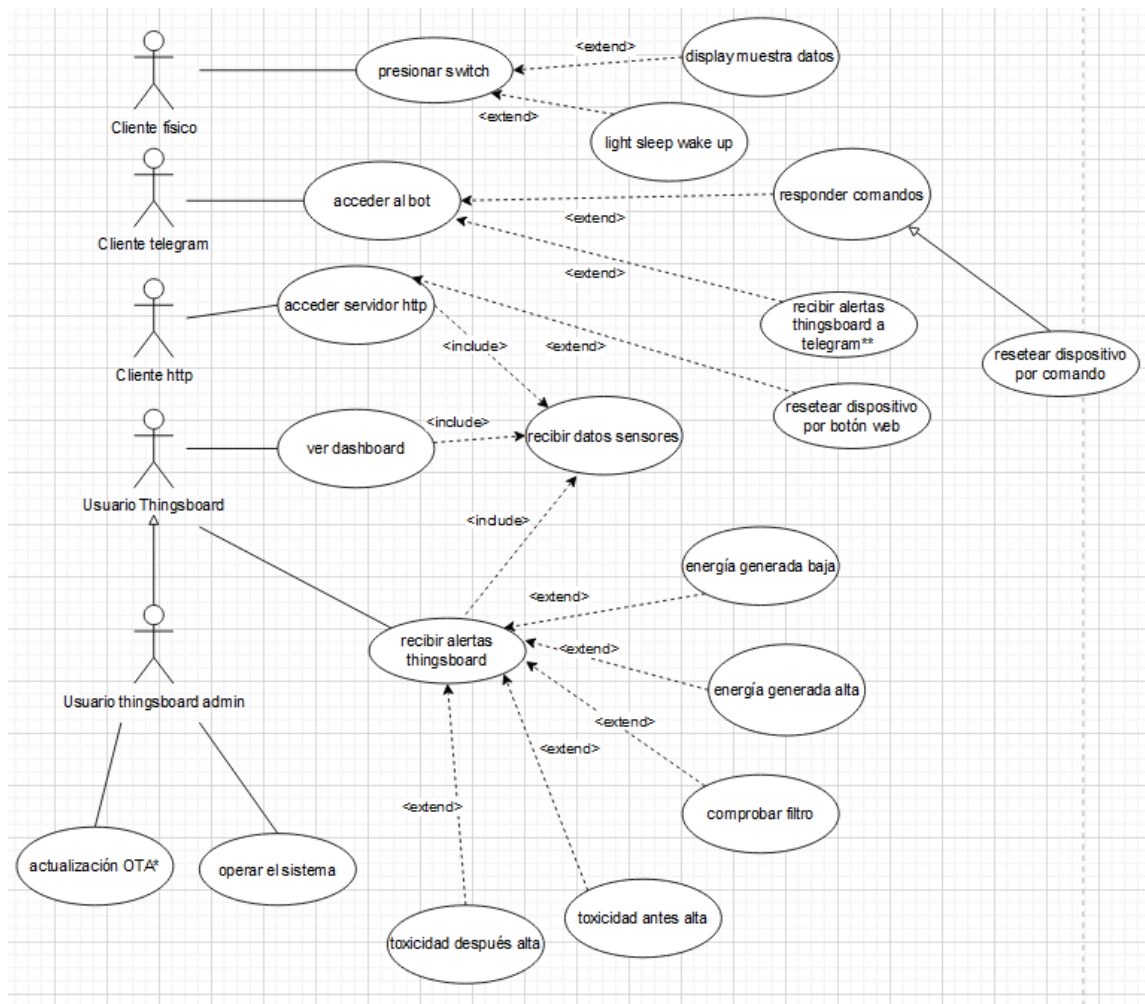


Figura 15. Casos de usos (elaboración propia).

Explicación casos de uso:

El cliente físico tiene la función de presionar el switch, si enciende este dará el acceso para que el display muestre los datos generados por los sensores de nuestro proyecto y se sale del light sleep.

El cliente de telegram tiene varias funciones, este se encargará de acceder al bot creado en telegram y dentro de este podrá consultar los comandos programados explicados anteriormente en nuestra memoria.

El cliente http mediante el servidor web creado podrá consultar los datos de los sensores y además podrá reiniciar la ESP32 mediante el botón web.

El usuario de thingsboard recibirá las alertas a partir de los datos generados por los sensores.

Por último, el usuario de thingsboard admin será capaz de actualizar la OTA y además podrá operar el sistema pudiendo modificar el panel (dashboard) del demo thingsboard.

Manual de usuario

El uso del sistema se ha descrito parcialmente en las secciones anteriores, pero queda bien recogerlos acá.

- Para el cliente Telegram, una vez se ha configurado el Panel y el programa del bot a tu ID de chat (individual o de grupo) ejecutas el comando /start para iniciar una conversación. El resto de los comandos ya se han mencionado en la sección 2.1.1. plataforma IoT, pero cabe mencionar que, si se hace en un chat de grupo, por políticas del Telegram el bot única y exclusivamente recibe comandos que empiecen con “/”, el resto de los mensajes de texto con contenido solo quedan atendidos si se hace chat individual directamente al bot. Por claridad todas las respuestas del bot siguen la estructura “<Mensaje al que responde>:<Nombre del bot>:<Mensaje de respuesta>”. En caso de que se le envíe un comando incorrecto, el mensaje de respuesta es “No entiendo”. Si el mensaje recibido proviene del Thingsboard, la trama de respuesta tiene la estructura “<Alerta>” has been triggered in device “<Nombre del dispositivo>”. Los mensajes de Thingsboard se siguen enviando hasta que la alerta termina (ver Figura 12) y se enviarán al usuario que el Administrador del Thingsboard haya configurado de antemano (ver doble asterisco (**) de la Figura 15)
- Para el cliente http, el sistema es muy simple, tras acceder a la IP local del ESP32 (debe consultarse con el servidor, pues no es fija) únicamente tiene tres opciones: encender/apagar el LED, y activar una cuenta atrás para reiniciar la ota a la partición de fábrica y buscar actualizaciones, por botones en la pantalla del móvil u ordenador. Una vez se activa la cuenta atrás la página no permite alterar ningún valor.
- Cliente físico: solo tiene la opción de activar o desactivar el display (que solo muestra el voltaje hidráulico y solar, así como las toxinas pre- y post filtro y el acrónimo del proyecto y la Universidad Politécnica de Madrid) mediante un switch – este botón también permite despertar al sistema de un sueño ligero, así como previene el acceso a la flash si se enciende, por lo que se puede utilizar para prevenir actualizaciones. Si quieres que alguna de estas dos funciones ocurra, por favor ponga el switch a off.
- Usuario Thingsboard: acá en realidad tienes el modo usuario normal o no registrado, y el del administrador. El primer modo solo puede ver el panel (explicado en la sección 2.1.1 plataforma IoT), mientras que con el segundo es posible alterar el panel e incluir nuevos dispositivos al perfil de la forma indicada en los múltiples tutoriales de Thingsboard. De momento solo un usuario (alejandro.serranol@alumnos.upm.es) es el administrador del Panel de Thingsboard; pero en este caso para permitir al profesorado el uso del Panel con todas las funcionalidades, hemos esperado que el usuario promedio no sea malicioso y desee alterar el firmware del dispositivo (ver asterisco simple (*) de la Figura 15) – esto es algo que para futuras versiones sería recomendable parchear.

3. DEMOSTRACIÓN: EVALUACIÓN CON/DEL SISTEMA

Inicialización desde la ota	
Descripción	Nuestro software por petición del profesorado requiere ser actualizado a distancia, esto supone emplear la ota de la ESP32 para guardar una partición de fábrica y otra dónde se almacena el firmware.
Hipótesis	La ota descarga el enlace y número de versión del Thingsboard, para luego descargar el firmware de dicho enlace (si su versión es más antigua) e iniciarlo.
Demostración	<p>Enlace al video de demostración en SharePoint (Ver anexo I)</p>  <p>The screenshot displays the boot logs of an ESP32 device running ESP-IDF v4.4.2-dirty. It shows the initialization of the bootloader, the partition table (including nvs, phy_init, and factory partitions), and the loading of the application from the factory partition. The logs indicate a successful OTA update process, with the application starting up and initializing the heap and flash.</p>

Figura 16. Imágenes OTA 1 (elaboración propia).

	<pre> I (0) cpu_start: Starting scheduler on APP CPU. I (605) ServidorSimple: Example configured to blink GPIO LED! I (615) gpio: GPIO[18] InputEn: 0 OutputEn: 0 OpenDrain: 0 Pullup: 1 Pulldown: 0 Intr:0 I (625) ServidorSimple: Example configured to switch GPIO LED! I (625) gpio: GPIO[2] InputEn: 0 OutputEn: 0 OpenDrain: 0 Pullup: 1 Pulldown: 0 Intr:0 I (635) ServidorSimple: Configuring analog pins I (645) gpio: GPIO[34] InputEn: 0 OutputEn: 0 OpenDrain: 0 Pullup: 1 Pulldown: 0 Intr:0 I (655) gpio: GPIO[32] InputEn: 0 OutputEn: 0 OpenDrain: 0 Pullup: 1 Pulldown: 0 Intr:0 I (665) gpio: GPIO[0] InputEn: 0 OutputEn: 0 OpenDrain: 0 Pullup: 1 Pulldown: 0 Intr:0 I (715) ServidorSimple: Inicialización Sensor Lumen A exitoso I (715) ServidorSimple: Inicialización Sensor Lumen B exitoso I (715) ServidorSimple: Inicialización Sensor Lumen C exitoso I (715) ServidorSimple: I2C initialized successfully I (725) ServidorSimple: Panel is 128x64 I (735) SSD1306: OLED configured successfully I (735) ServidorSimple: Estamos en ESP-32 This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 1, 4MB external flash Minimum free heap size: 273304 bytes I (805) wifi:wifi driver task: 3ffc0990, prio:23, stack:6656, core=0 I (805) system_api: Base MAC address is not set I (805) system_api: read default base MAC address from EFUSE I (835) wifi:wifi firmware version: eea27d I (835) wifi:wifi certification version: v7.0 I (835) wifi:config NVS flash: enabled I (835) wifi:config nano formatting: disabled I (835) wifi:Init data frame dynamic rx buffer num: 32 I (845) wifi:Init management frame dynamic rx buffer num: 32 I (845) wifi:Init management short buffer num: 32 I (845) wifi:Init dynamic tx buffer num: 32 I (855) wifi:Init static rx buffer size: 1600 I (855) wifi:Init static rx buffer num: 10 I (865) wifi:Init dynamic rx buffer num: 32 I (865) wifi_init: rx ba win: 6 I (865) wifi_init: tcpip mbox: 32 I (875) wifi_init: udp mbox: 6 I (875) wifi_init: tcp mbox: 6 I (885) wifi_init: tcp tx win: 5744 I (885) wifi_init: tcp rx win: 5744 I (885) wifi_init: tcp mss: 1440 I (895) wifi_init: WiFi IRAM OP enabled I (895) wifi_init: WiFi RX IRAM OP enabled I (905) example_connect: Connecting to SBC... I (905) phy_init: phy_version 4670,719f9f6, Feb 18 2021,17:07:07 I (1015) wifi:mode : sta (24:6f:28:7c:1f:b8) I (1015) wifi:enable tsf I (1025) example_connect: Waiting for IP(s) I (3425) wifi:new:<6,0>, old:<1,0>, ap:<255,255>, sta:<6,0>, prof:1 I (4175) wifi:state: init -> auth (b0) I (4185) wifi:state: auth -> assoc (0) I (4185) wifi:state: assoc -> run (10) </pre> <p>Figura 17. Imágenes OTA 2 (elaboración propia).</p> <pre> w (4205) wifi:cb:aa:dd:idx:0 (ifx:0, 70:4d:7b:8c:a3:b8), tid:0, ssn:0, winSize:64 I (4215) wifi:connected with SBC, aid = 2, channel 6, Bw20, bssid = 70:4d:7b:8c:a3:b8 I (4215) wifi:security: WPA2-PSK, phy: bgn, rssi: -54 I (4215) wifi:pm start, type: 1 I (4245) wifi:AP's beacon interval = 102400 us, DTIM period = 3 I (5795) example_connect: Got IPv6 event: Interface "example_connect: sta" address: fe80:0000:0000:0000:26ff:fe7c:1fb8, type: ESP_IP6_ADDR_IS_LINK_LOCAL I (6295) ServidorSimple: Starting server I (6295) esp_https_server: Starting server I (6295) esp_https_server: Server listening on port 443 I (6295) ServidorSimple: Registering URI handlers I (6305) esp_netif_handlers: example_connect: sta ip: 192.168.1.177, mask: 255.255.255.0, gw: 192.168.1.1 I (6315) example_connect: Got IPv4 event: Interface "example_connect: sta" address: 192.168.1.177 I (6325) example_connect: Connected to example_connect: sta I (6325) example_connect: - IPv4 address: 192.168.1.177 I (6335) example_connect: - IPv6 address: fe80:0000:0000:0000:26ff:fe7c:1fb8, type: ESP_IP6_ADDR_IS_LINK_LOCAL Connecting to server </pre> <p>Figura 18. Imágenes OTA 3 (elaboración propia).</p>
Datos	<p>Ninguno en sí, aparte de decir versión del Thingsboard (1.9) y enlace URL</p> <p>http://raw.githubusercontent.com/tardisfromtornspace/SBC_22_M_01/master/simple/build/https_server.bin</p>
Interpretación	<p>El sistema funciona correctamente</p>

Tabla 2: Prueba inicialización (Elaboración propia).

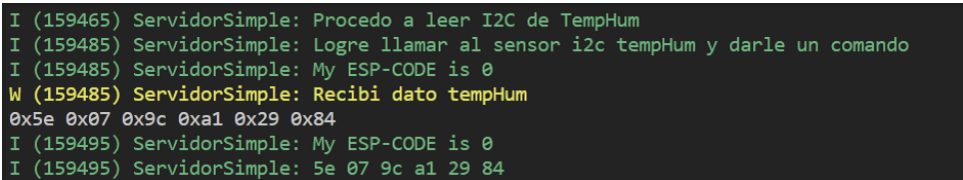
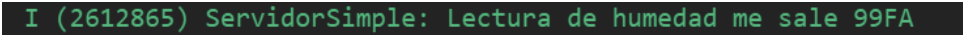
Prueba del mock de toxicidad antes del filtro (TempHum)	
Descripción	Debido a las limitaciones por presupuesto, chasis, mantener líquidos contenidos en un proyecto móvil y de no poder mantener dos sensores idénticos a la ESP32 sin activar un segundo bus I2C, lo que no es posible con el Wi-fi activado, decidimos utilizar un sensor alternativo.
Hipótesis	Al poner algo húmedo cerca del sensor, el valor de humedad debería aumentar.
Demostración	<p>Enlace al video de demostración en SharePoint (Ver anexo I)</p>  <p>Figura 19. Imágenes I2C TempHum 1 (elaboración propia).</p>  <p>Figura 20. Imágenes I2C TempHum 2 (elaboración propia).</p>
Datos	Visibles en el vídeo en Thingsboard.
Interpretación	El sistema funciona correctamente.

Tabla 3: Prueba toxicidad antes del filtro (Elaboración propia).

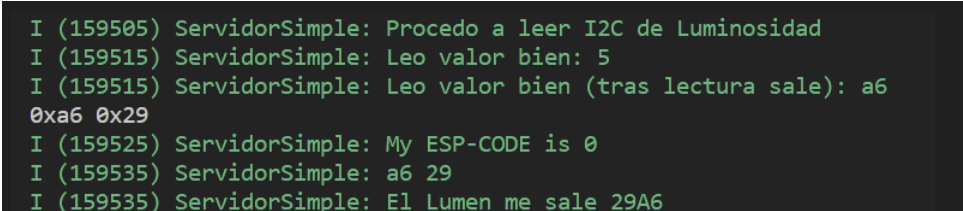
Prueba del mock de toxicidad tras el filtro (Luminosidad)	
Descripción	Debido a las limitaciones por presupuesto, chasis, mantener líquidos contenidos en un proyecto móvil y de no poder mantener dos sensores idénticos a la ESP32 sin activar un segundo bus I2C, lo que no es posible con el Wi-fi activado, decidimos utilizar un sensor alternativo.
Hipótesis	Al tapar el sensor de luz, debería reducirse el valor de luminosidad.
Demostración	<p>Enlace al video de demostración en SharePoint (Ver anexo I)</p>  <p>Figura 21. Imagen I2C luminosidad (elaboración propia).</p>
Datos	Visibles en el vídeo en Thingsboard.
Interpretación	El sistema funciona correctamente.

Tabla 4: Prueba toxicidad tras el filtro (Elaboración propia).

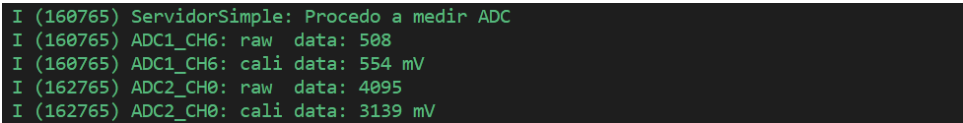
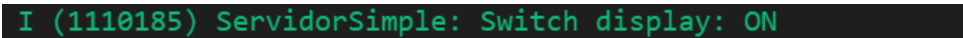
Prueba de detección del ADC	
Descripción	El sensor analógico digital permite medir los voltajes generados por las turbinas y panel solar (que alimentan la batería) por separado gracias a su gran impedancia que evita el flujo de voltaje. Se utiliza un único ADC multiplexado porque el módulo wi-fi emplea el segundo.
Hipótesis	Al girar la turbina se generará electricidad, y el voltaje recibido en su pin ADC correspondiente se reflejará, lo mismo para la placa solar
Demostración	<p>Enlace al video de demostración en SharePoint (Ver anexo I)</p>  <p>Figura 22. Imagen detección ADC (elaboración propia).</p>
Datos	Visibles en el vídeo en Thingsboard.
Interpretación	El sistema funciona correctamente.

Tabla 5: Prueba sensores ADC básica (Elaboración propia).

Prueba de respuesta del display y el switch a lecturas	
Descripción	El display es una forma de poder observar los datos sin tener que emplear el WI-fi <i>in situ</i> , por ejemplo, para áreas remotas sin muchos equipos con acceso a Wi-fi disponibles.
Hipótesis	Al apagar el switch el display se apaga. Con el display encendido los datos se actualizan cada pocos segundos.
Demostración	<p>Enlace al video de demostración en SharePoint (Ver anexo I)</p> <p>Prueba con Switch ON, potenciómetro al máximo:</p>  <p>Figura 23. Imagen detección switch activo en consola (elaboración propia)</p>

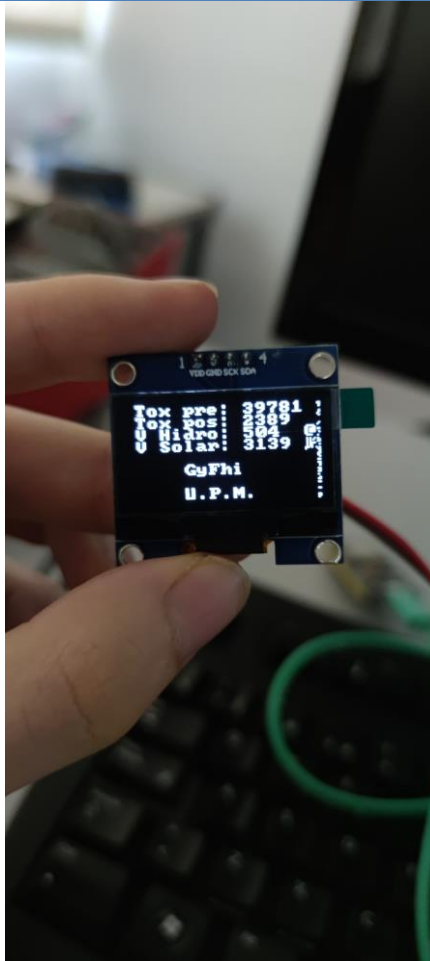


Figura 24. Imagen display activo con switch a ON (elaboración propia).

Prueba con Switch OFF, potenciómetro al mínimo:

```
I (1747935) ServidorSimple: Switch display: OFF
```

Figura 25. Imagen detección switch inactivo en consola (elaboración propia)

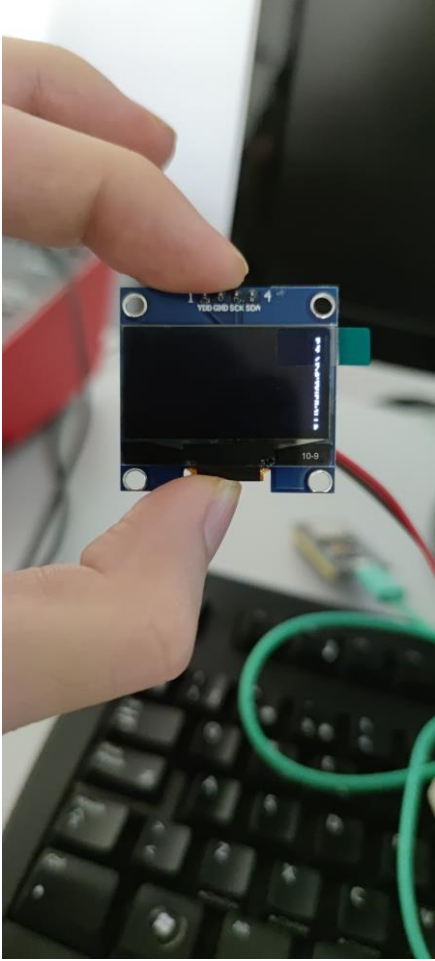
	 <p>Figura 26. Imagen display a clear con el switch a OFF (elaboración propia)</p>
Datos	Visibles en el vídeo en Thingsboard.
Interpretación	El sistema funciona correctamente.

Tabla 6: Prueba display OLED (Elaboración propia).

Prueba proyecto final	
Descripción	Las funciones vitales de detección del sistema deben poder aplicarse correctamente a una versión del proyecto aplicable al mundo real
Hipótesis	El sistema debe de funcionar con los componentes reales que podamos aplicar y sigue funcionando plenamente al mantenerse alimentado con su propia batería.
Demostración	Enlace a los vídeos de demostración en SharePoint (Ver anexo I). Los dos primeros son con alimentación externa. El último con alimentación portátil propia de batería y condensador.

Datos	Visibles en los vídeos en Thingsboard.
Interpretación	El sistema funciona adecuadamente y es capaz de resistir con una batería que el propio sistema carga y mantener los servicios Wi-fi incluida la comunicación con el Thingsboard, aunque no se puede demostrar correctamente hasta que dispongamos del chasis adecuado para realizar las pruebas.

Tabla 7: Prueba display OLED (Elaboración propia).

4. DISCUSIÓN Y CONCLUSIONES

4.1 LIMITACIONES Y DIFICULTADES ENCONTRADAS

Descubrimos que la ESP32 no tolera clock stretching bien, ni tampoco permite usar una segunda unidad de ADC ni I2C si se utiliza el módulo Wi-Fi. Esto nos ha prevenido de usar algunos sensores, así como de usar más de 1 vez el mismo sensor (ya que tienen misma dirección y la única forma de llamarlos sería utilizar un segundo bus I2C).

Además por falta de presupuesto y equipo especializado no pudimos probar los sistemas interactuando con el agua en sí, sino utilizando el aire como fluido para simular la generación de electricidad, y dos sensores no relacionados para simular la medida de toxinas antes y después de un filtro (que tampoco se pudo probar). Además por contratiempos con las impresoras 3D (primero que las impresoras estaban ocupadas, luego problemas con la calibración de la cama, posteriormente que durante la noche la impresión se estropeó y que luego al intentarlo en casa a la impresora 3D de Ismael se le atascó el extrusor) el chasis y piezas impresas no se pudieron realizar a tiempo.

También nos enfrentamos a un problema con la función `light_sleep()` que causaba que se redujera la velocidad de escritura antes de ser llamada y luego entrase en un sleep perpetuo sin que se despertase – cosa que con `deep_sleep()` sí realizaba. Posteriormente descubrimos que era porque la función `light_sleep()` no espera, y tuvimos que hacer uso de la librería `uart` para que esperase. Una vez hecho esto sí funcionaba, pero el problema es que desactivar el wi-fi supone que el resto de la infraestructura wireless colapsaba también, por lo que tuvimos que encontrar un equilibrio entre comunicación y consumo de energía - se decidió que los sleep convencionales tuvieran un período máximo de 60 segundos, y luego para dejar tiempo a re-comunicarse se realizaban 15 mediciones. Aún así, esto supone un gran inconveniente para aquellos que utilizaran el servidor web `http`, puesto que existía el riesgo de la IP de acceso cambiara. Y aún así, no siempre se garantizaba un buen funcionamiento con el `light sleep` al causar que los módulos de wifi se quedaran buscando red y algunas `Tasks` asociadas se cerraran. Y no deberíamos usar el `deep sleep` con la opción de escribir el offset del `http2` de telegram en la flash puesto que constantes escrituras del offset en la flash reducirían su vida útil considerablemente. Por lo tanto al final viendo que necesitábamos que el wi-fi se mantuviera activo para comunicar de alertas de alta toxicidad en el agua (lo cual puede ser crítico), decidimos descartar la función de dormir ya que todas las funciones de sleep desactivan el wifi (aunque la dejamos en el código para atestiguar que la habíamos realizado).

4.2 IMPLICACIONES, PROSPECTIVA Y CÓMO EXTENDER EL TRABAJO

Este sistema con pequeñas modificaciones sería capaz de realizar la tarea dada de generar electricidad. También podríamos modificar el programa para permitir saber el nivel de carga de la batería.

Los filtros del sistema podrían mejorarse para permitir el consumo humano.

Con respecto a la seguridad, sería recomendable restringir el uso del widget de actualización de la URL del firmware de la OTA a los administradores u otros usuarios privilegiados.

Sería posible utilizar tecnologías como la Torre Wardenclyffe de Tesla para transmitir la energía por el aire y así permitir cargas sin conexión física y reducir el uso de cables y sus emisiones de (micro)plásticos del aislante y de cobre en la naturaleza. Es una tecnología existente, aunque no muy extendida (Sector Electricidad, 2014).

Por último, para reducir la energía que el sistema electrónico consumiría, sería posible combinar los sistemas de radio de galena (ieee.org, 2010) (Giordano, 2010), el ladrón de julios (oscar, 2016) o el sistema del *Smart Necklace* (Montalbano, 2022) para usar fuente de energía según radiación electromagnética.

REFERENCIAS

- (s.f.). Recuperado el 27 de 04 de 2018, de Imagen predeterminado de Microsoft.: <http://microsoft.com>
- AMS.com. (6 de 12 de 2022). *pdf1.alldatasheet.com*. Obtenido de pdf1.alldatasheet.com: <https://pdf1.alldatasheet.com/datasheet-pdf/view/861959/AMSCO/IAQ-CORE.html>
- arquero99. (2 de 12 de 2022). *github*. Obtenido de github: <https://github.com/arquero99/E-TEASPILS>
- Az-delivery. (6 de 12 de 2022). *www.az-delivery.de*. Obtenido de *www.az-delivery.de*: <https://www.az-delivery.de/en/products/1-3zoll-i2c-oled-display>
- Barrie, A. (18 de 5 de 2015). *www.foxnews.com*. Obtenido de Fox News: <https://www.foxnews.com/story/lifesaver-bottle-purifies-water-in-seconds>
- D. Álvarez, M.-C. M. (28 de 11 de 2022). *rtve.es*. Obtenido de *rtve.es*: <https://www.rtve.es/noticias/20221128/guerra-ucrania-directo-ultima-hora-noticia/2410208.shtml>
- EcoInventos. (2 de 9 de 2022). *ecoinventos.com*. Obtenido de *ecoinventos.com*: <https://ecoinventos.com/seabin-cubo-basura-flotante-limpiar-oceano/>
- endesa. (17 de 12 de 2021). *www.endesa.com*. Obtenido de *endesa.com*: <https://www.endesa.com/es/la-cara-e/energias-renovables/energia-hidraulica>
- Espressif. (6 de 12 de 2022). *www.espressif.com*. Obtenido de *www.espressif.com*: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf#section.4
- Espressif. (1 de 10 de 2022). *www.espressif.com*. Obtenido de *www.espressif.com*: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf#d
- Espressif Inc. (1 de 9 de 2019). *www.alldatasheet.com*. Obtenido de *alldatasheet.com*: <https://www.alldatasheet.com/datasheet-pdf/pdf/1179101/ESPRESSIF/ESP-WROOM-32.html>
- Flores, D. (15 de 11 de 2022). *https://rtve.es*. Obtenido de *rtve.es*: <https://www.rtve.es/noticias/20221115/poblacion-mundial-ocho-mil-millones-personas/2408893.shtml>
- Fundación Aquae. (22 de 9 de 2021). *www.fundacionaquae.org*. Obtenido de *fundacionaquae.org*: <https://www.fundacionaquae.org/agua-y-contaminacion/>
- George. (25 de 1 de 2012). *www.circuitspecialists.com*. Obtenido de *circuitspecialists.com*: <https://www.circuitspecialists.com/blog/generate-power-with-a-stepper-motor/>

- Giordano, J. L. (2 de 9 de 2010). <http://www.profisica.cl>. Obtenido de profisica.cl:
<http://www.profisica.cl/index.php/component/content/article/113-fisica-cotidiana/como-funcionan-las-cosas/143-la-radio-galena?Itemid=542>
- IBERDROLA. (1 de 1 de 2019). www.iberdrola.com. Obtenido de iberdrola:
<https://www.iberdrola.com/medio-ambiente/microplasticos-amenaza-para-la-salud>
- ieee.org. (2 de 9 de 2010). reach.ieee.org. Obtenido de reach.ieee.org:
<https://reach.ieee.org/multimedia/crystal-radio/>
- IEEEExplore. (21 de 4 de 2022). ieeexplore.ieee.org. Obtenido de ieee.org:
<https://ieeexplore.ieee.org/document/9761153>
- IEEEExplore. (28 de 10 de 2022). ieeexplore.ieee.org. Obtenido de ieee.org:
<https://ieeexplore.ieee.org/document/9596398>
- Montalbano, E. (3 de 8 de 2022). www.designnews.com. Obtenido de designnews.com:
<https://www.designnews.com/medical/smart-necklace-can-track-blood-sugar-through-sweat>
- National Geographic. (30 de 6 de 2022). www.nationalgeographic.com. Obtenido de nationalgeographic:
https://www.nationalgeographic.com.es/ciencia/detectan-microplasticos-90-agua-embotellada_14456
- oscar. (15 de 8 de 2016). codigoelectronica.com. Obtenido de codigoelectronica.com:
<http://codigoelectronica.com/blog/ladron-julios>
- rs-online. (6 de 12 de 2022). [rs-online](http://rs-online.com). Obtenido de rs-online: <https://es.rs-online.com/web/p/circuitos-integrados-de-sensores-de-luz-y-de-color/1807471>
- rs-online. (7 de 6 de 2022). [www.rsonline.com](http://rsonline.com). Obtenido de rsonline: <https://es.rs-online.com/web/p/circuitos-integrados-de-sensores-ambientales/1024162>
- RTVE.es / AGENCIAS. (2 de 12 de 2022). rtve.es. Obtenido de rtve.es:
<https://www.rtve.es/noticias/20221202/precio-luz-espana-diciembre-electricidad/2410640.shtml>
- Sector Electricidad. (7 de 12 de 2014). www.sectorelectricidad.com. Obtenido de sectorelectricidad.com:
<https://www.sectorelectricidad.com/10948/tesla-y-el-proyecto-wardenclyffe-electricidad-inalambrica-para-todo-el-mundo-2/>
- Sensirion. (6 de 12 de 2022). <https://sensirion.com>. Obtenido de <https://sensirion.com>:
https://sensirion.com/media/documents/4B40CEF3/61642381/Sensirion_Humidity_Sensors_SHT85_Datasheet.pdf
- sensirion.com. (6 de 12 de 2022). <https://sensirion.com>. Obtenido de <https://sensirion.com>:
<https://sensirion.com/products/catalog/SHT85/>
- VISHAY. (6 de 12 de 2022). www.vishay.com. Obtenido de pdf1.alldatasheet.com:
<https://pdf1.alldatasheet.com/datasheet-pdf/view/1124354/VISHAY/VEML7700.html>

ANEXOS

ANEXO I. ENLACES DE DOCUMENTOS DE SHAREPOINT

Videos de pruebas del dispositivo:

- Ota (físico):
<https://upm365.sharepoint.com/:v:/s/SBC22M01/Eewntrl3au1Hmop1PO3AOOMBzqb8QnGsR1z73WN1styu0w?e=E20GDS>
- Ota (pantalla)
<https://upm365.sharepoint.com/:v:/s/SBC22M01/Efi9V8WOv0lOmeBLwme17lwBa5WUrxW7Ewhrq39C2HnHsQ?e=1cNqP2>
- Lecturas de los sensores:
<https://upm365.sharepoint.com/:v:/s/SBC22M01/EYQQIoVNdFBEhmtr67G2yVUBiVNeBLzpR5rDJ6eTSanxHA?e=IjDNPe> y
https://upm365.sharepoint.com/:v:/s/SBC22M01/EfTCbjWygPVNnZ3RAMLanAUBCpqbnB5gV52ccQ8UNp-E_g?e=JdYVfk
- Acceso servidor wifi:
https://upm365.sharepoint.com/:v:/s/SBC22M01/EfTCbjWygPVNnZ3RAMLanAUBCpqbnB5gV52ccQ8UNp-E_g?e=JdYVfk
- Acceso Telegram:
https://upm365.sharepoint.com/:v:/s/SBC22M01/EfTCbjWygPVNnZ3RAMLanAUBCpqbnB5gV52ccQ8UNp-E_g?e=JdYVfk
- Prueba de respuesta del display y el switch a lecturas
<https://upm365.sharepoint.com/:v:/s/SBC22M01/EZcNa64KObdLpUyJTS6meABLw8mWvbnkW06GTyaAtfURw?e=cqhWPJ>
- https://upm365.sharepoint.com/:v:/s/SBC22M01/EfTCbjWygPVNnZ3RAMLanAUBCpqbnB5gV52ccQ8UNp-E_g?e=JdYVfk
- Prueba proyecto final:
 - Dinamo (no portable):
https://upm365.sharepoint.com/:v:/s/SBC22M01/EW9PiQrXd5BHvNhHAzFM3G8Bnxv4Yi7e3WqXQ7-f2CZx_w?e=sAAh5I
 - Placa solar (no portable):
https://upm365.sharepoint.com/:v:/s/SBC22M01/EW9PiQrXd5BHvNhHAzFM3G8Bnxv4Yi7e3WqXQ7-f2CZx_w?e=Q0fSh4
 - Portabilidad:
https://upm365.sharepoint.com/:v:/s/SBC22M01/EVkJQFkZFPOVHq2uC2D4cUIUBHb7QVpNHDkfv15NMP06_MA?e=EOXBxr

Documentos de componentes específicos pedidos

- <https://upm365.sharepoint.com/:t:/s/SBC22M01/EXKw43l2QfFNmrYVJt1E2h4BuoUElffUdWBqeLGDY6Bmgg?e=7ITUHf>

Archivos de impresión 3D del chasis

- <https://upm365.sharepoint.com/:f:/s/SBC22M01/ElWkPAu-v6xNhaQXhVe7zD8BFqoSjBBn31gPomwuwmFGQ?e=iskskW>

ANEXO II. CÓDIGO GITHUB

Código general de nuestro GitHub:

https://github.com/tardisfromtornspace/SBC_22_M_01/tree/main/simple

En el fichero sdkconfig se indican **algunas de las configuraciones del dispositivo**. Ver código:

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/77892d5a90777bb76c47367ee9e1e87046708849/simple/sdkconfig#L138

sh2lib en nuestro código:

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/8ef4beb11bd3d9e9c73ab4b9fb0adb6a39717e43/simple/components/sh2lib/sh2lib.c#L14

ssd1306_i2c:

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/components/ssd1306/ssd1306.c#L1

Switches diferentes respecto al ejemplo original de Thingsboard:

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/661ea3224f9320f38c3634cac50292e4f2267927/ota/esp32-ota/main/main.c#L89

Secciones LED-switch y comunes

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L96

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L191

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L220

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L1507 El programa principal en sí.

Secciones I2C básicas

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L101

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L214 Esta no se utiliza, pero la incluimos para demostrar que lo intentamos y el código sería funcional en otros

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L278

Secciones ADC

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L138

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L207

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L748

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L1381 Este configura los pines analógicos en sí.

Secciones MQTT

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L680

Secciones http del servidor básico

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L1107

Secciones http2 de Telegram

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L165

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L777

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L894 esta sección está incluida arriba, pero se destaca por tener los comandos que recibe del Telegram y cómo responder a ellos.

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L1321

Secciones de retorno de fábrica

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L1352

Secciones de sleep

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L188

https://github.com/tardisfromtornspace/SBC_22_M_01/blob/08513953429cdc7efc440db3a7f16a2977df49df/simple/main/main.c#L1394 la rutina del sleep en sí.