ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Tecnología Electrónica

Área de Tecnología Electrónica

TRABAJO FIN DE GRADO

Supervisor de consumo de agua

Grado en

Ingeniería Electrónica Industrial

Autor: Alejandro Vela Alarcón

Tutor: Francisco José Sánchez Pacheco

Cotutor Pedro Juan Sotorrío Ruiz

MÁLAGA, mayo de 2025

DECLARACIÓN DE ORIGINALIDAD DEL

PROYECTO/TRABAJO FIN DE GRADO

D./ Dña.: Alejandro Vela Alarcón

Titulación: Grado en Ingeniería Electrónica Industrial

Título del Proyecto/Trabajo: Supervisor de consumo de agua

DECLARA BAJO SU RESPONSABILIDAD

Ser autor/a del texto entregado y que no ha sido presentado con anterioridad, ni total ni parcialmente, para superar materias previamente cursadas en esta u otras titulaciones de la Universidad de Málaga o cualquier otra institución de educación superior u otro tipo de fin.

Así mismo, declara no haber trasgredido ninguna norma universitaria con respecto al plagio ni a las leyes establecidas que protegen la propiedad intelectual, así como que las fuentes utilizadas han sido citadas adecuadamente.

En Málaga, a 24 de Mayo de 2025

Fdo.: .....

Resumen.

En un contexto de creciente preocupación por la escasez de agua, este trabajo presenta el diseño y desarrollo de un prototipo de sistema supervisor de consumo de agua, que permite el control detallado del consumo del preciado líquido elemento.

Se propone una solución tecnológica basada en el microcontrolador ESP32 con conectividad WiFi, que permite al usuario monitorizar el consumo de agua.

El sistema emplea un caudalímetro para la medición del flujo, cuyos datos son procesados por el microcontrolador y puestos a disposición del usuario a través de una interfaz web alojada por el propio sistema. Esta solución tiene como objetivo fomentar un uso más consciente y eficiente del agua.

Índice

Índice de figuras

Índice de tablas

Glosario

Debounce → Técnica utilizada para eliminar las oscilaciones eléctricas no deseadas al accionar un pulsador o interruptor, evitando lecturas erróneas de múltiples pulsos.

HTTP → Protocolo de comunicación utilizado en la web para la transferencia de información entre un cliente (generalmente un navegador) y un servidor web.

Hz (Hertzios) → Unidad de frecuencia que indica el número de ciclos por segundo de una señal periódica.

IoT (Internet of Things) → Red de dispositivos físicos interconectados que recopilan, comparten y gestionan datos a través de Internet, permitiendo la automatización y el control remoto.

LSB (Least Significant Bit) → El bit menos significativo de una secuencia binaria; representa la menor potencia de dos y determina los incrementos más pequeños del valor.

mA (Miliamperio) → Unidad de medida de la corriente eléctrica equivalente a una milésima parte de un amperio.

MSB (Most Significant Bit) → El bit más significativo de una secuencia binaria; representa la mayor potencia de dos y determina la magnitud más alta del valor.

Servidor web → Sistema software que procesa solicitudes enviadas a través del protocolo HTTP (o HTTPS) y responde enviando páginas web al cliente.

SPI (Serial Peripheral Interface) → Protocolo de comunicación síncrono utilizado para transmitir datos entre un microcontrolador y dispositivos periféricos a altas velocidades.

timeStamp o Epoch → Marca temporal que indica la fecha y hora exacta asociada a un evento o registro de datos.

USB (Universal Serial Bus) → Estándar de comunicación y alimentación que permite conectar y transferir datos entre dispositivos electrónicos y ordenadores de forma rápida y sencilla.

V (Voltio) → Unidad de medida del potencial eléctrico o diferencia de potencial entre dos puntos de un circuito.

Capítulo I

1. Introducción

1.1. Antecedentes

En los últimos años, el control y la monitorización del consumo de agua han aumentado su importancia, tanto en el ámbito doméstico como industrial. Esto responde al aumento de la demanda de agua y a la creciente preocupación por las sequías. Como consecuencia, han surgido diversos sistemas comerciales de medición que permiten visualizar el consumo de agua en tiempo real y detectar posibles fugas o excesos, como los contadores inteligentes. Algunos ejemplos de estos dispositivos son "Itron® IntelliH2O" [9], "Kamstrup® MULTICAL 21" [10] o "Sensus® iPERL"[11], que integran tecnologías de comunicación como "LoRaWAN" o "NB-IoT" para el envío de datos y permiten una gestión remota y eficiente del consumo hídrico.

Sin embargo, muchas de estas soluciones tienen un coste elevado o dependen de infraestructuras externas, como servidores en la nube, aplicaciones móviles propietarias o integraciones complejas con sistemas de domótica. Esto ha impulsado la aparición de proyectos de bajo coste y de código abierto, basados en microcontroladores como Arduino, ESP8266 o ESP32, que permiten a usuarios y desarrolladores crear sus propios sistemas de monitorización personalizados.

El microcontrolador ESP32, en particular, se ha convertido en una plataforma muy popular en el desarrollo de dispositivos IoT, gracias a su capacidad de procesamiento, bajo consumo energético, conectividad WiFi integrada y su amplio soporte por parte de la comunidad. Su uso en proyectos de control y comunicación lo hace ideal para aplicaciones como la que se plantea en este trabajo.

En cuanto a la medición de caudal, los sensores más comunes utilizan tecnologías basadas en turbinas internas, es decir, pequeños rotores que giran al paso del agua y generan pulsos eléctricos proporcionales al volumen que fluye. Estos sensores, como los modelos YF-B10 [4] o similares, son fáciles de integrar con microcontroladores y permiten obtener mediciones razonablemente precisas.

1.2. Objetivo

El objetivo de este trabajo es diseñar y desarrollar un prototipo funcional de un sistema supervisor de consumo de agua, utilizando un microcontrolador ESP32 [1] y tecnología de comunicación WiFi. El sistema se orienta tanto a entornos domésticos como industriales, y se plantea como una solución de bajo coste y fácilmente escalable que permite conocer el consumo de agua en tiempo real a través de una interfaz web accesible desde cualquier dispositivo conectado a la misma red.

A través de esta interfaz, se facilita el acceso a información detallada como el volumen total de agua consumido, las lecturas acumuladas a lo largo del tiempo, registros temporales de cada lectura y el histórico de consumo entre fechas seleccionadas por el usuario. La información se presenta mediante tablas de datos que pueden consultarse y descargarse en formato CSV para su posterior análisis.

Además, este trabajo sienta las bases para una posible evolución hacia un producto comercializable, teniendo en cuenta la creciente necesidad de sistemas de gestión eficiente del agua debido a la escasez de recursos hídricos en determinadas regiones.

1.3. Metodología

El desarrollo del proyecto se ha llevado a cabo en varias fases. En primer lugar, se definen los objetivos del sistema y se seleccionaron los componentes necesarios. A continuación, se diseña el sistema de medición, utilizando un circuito simulado para los pulsos de caudal y registros de desplazamiento para transferir los datos en serie.

Se desarrolla el programa del microcontrolador ESP32, que se encarga de contar los pulsos, calcular el consumo de agua y mostrar los datos en una pantalla y una página web accesible por WiFi.

Finalmente, se realizan pruebas para validar el funcionamiento del prototipo y garantizar la correcta presentación de los datos.

Capítulo II

2. Visión general del sistema

El proyecto se ha realizado usando como principal componente una placa que contiene el microcontrolador ESP32 integrado en ella, se muestra una imagen en la figura 1.

Esta placa tiene patillas a las que se puede conectar los distintos periféricos (pantalla, botones, contadores…) con los que opera el microcontrolador. A su vez, esta placa está conectada a un adaptador para una mayor facilidad a la hora de realizar las conexiones.

Este adaptador consiste en un módulo de expansión que facilita la conexión entre el microcontrolador y los diferentes dispositivos externos. Proporciona una distribución más ordenada de las patillas, y mejora la robustez de las conexiones durante el desarrollo y las pruebas.

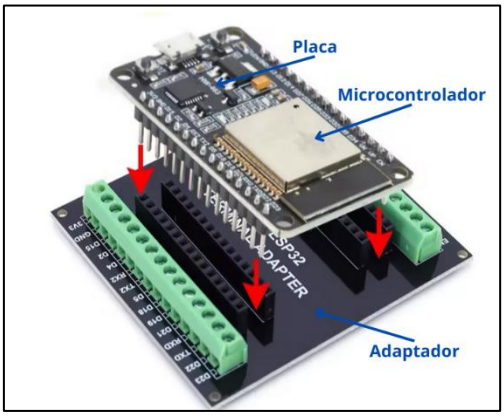


Figura 1. Microcontrolador, placa y adaptador.
Fuente aliexpress.com.

2.1. Diagrama de bloques del sistema

En la figura 2 se muestra el diagrama de bloques del sistema, en el que se representan las principales partes que lo componen, organizadas en torno al microcontrolador. Este se sitúa en el centro del esquema, ya que actúa como núcleo de control y coordinación entre los distintos módulos. El sistema se divide en cuatro bloques funcionales, que se describen a continuación:

- Conteo: Este bloque se encarga de medir el caudal de agua mediante el conteo de pulsos generados por un sensor de flujo. La señal del sensor es procesada por un contador, y posteriormente los datos son preparados para su lectura utilizando un registro de desplazamiento. El microcontrolador gestiona las señales de control del contador y del registro de desplazamiento.

- Alimentación: Proporciona la energía necesaria para el funcionamiento del microcontrolador y del resto de los módulos. La alimentación se realiza mediante una fuente de energía externa conectada al sistema.

- Interfaz lógica: Permite la comunicación remota entre el sistema y el usuario a través de una red local. Mediante una conexión inalámbrica, el usuario puede acceder a una interfaz web para visualizar en tiempo real el consumo registrado y consultar otros datos relevantes del sistema.

- Interfaz física: Facilita la interacción directa con el sistema sin necesidad de conexión a Internet. Está compuesta por una pantalla que muestra información de las últimas lecturas, una memoria externa donde se almacenan los datos registrados, y botones que permiten ejecutar distintas funciones como activación de modos de funcionamiento.

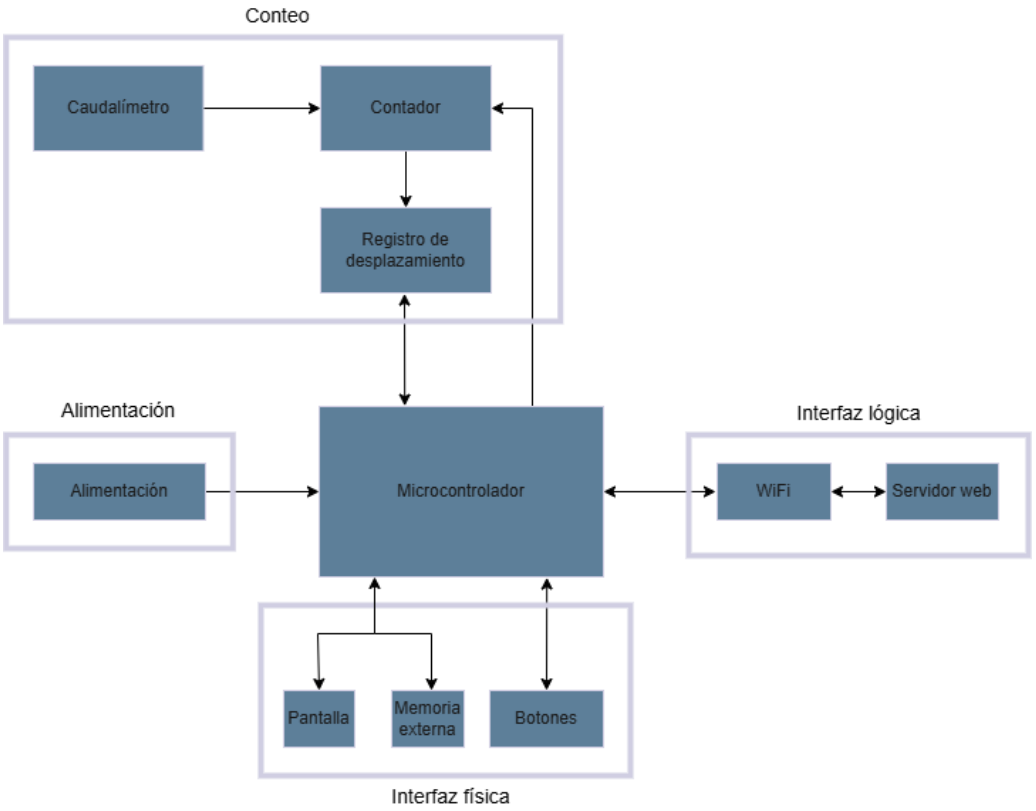


Figura 2. Diagrama de bloques del sistema.
Fuente propia.

2.2. Funcionamiento general

2.2.1. Alimentación

El sistema necesita de una fuente de 5 V para funcionar, esto es proporcionado mediante un cable USB que se conecta a la placa en la que está el microcontrolador ESP32. El resto de los componentes funcionan a 3,3 V, que es la tensión que proporciona la placa.

2.2.2. Conteo

Con foto de su lugar en el prototipo mostrada en la figura 3, y esquema mostrado en la figura 4. Se emplea el temporizador TLC555 [12] porque permite generar pulsos de frecuencia ajustable de forma precisa y estable, simulando así el comportamiento de un caudalímetro real.

El circuito del TLC555 funciona como un simulador de flujómetro. En concreto, simula el caudalímetro YF-B10.

Su facilidad de configuración y fiabilidad lo convierten en una herramienta adecuada para realizar pruebas controladas del sistema sin depender de un flujo físico de agua.

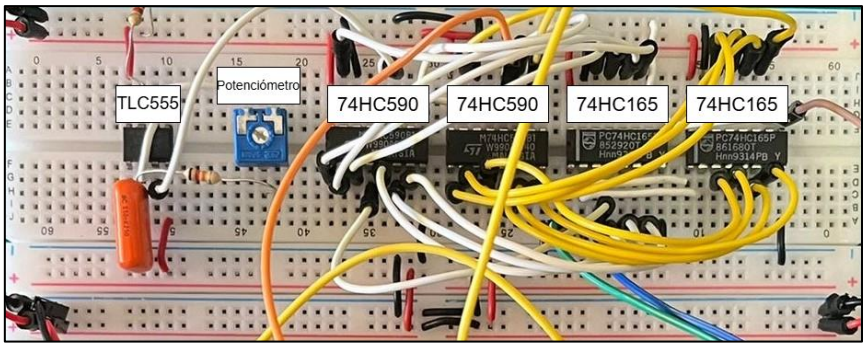


Figura 3. Grupo de conteo.
Fuente propia.

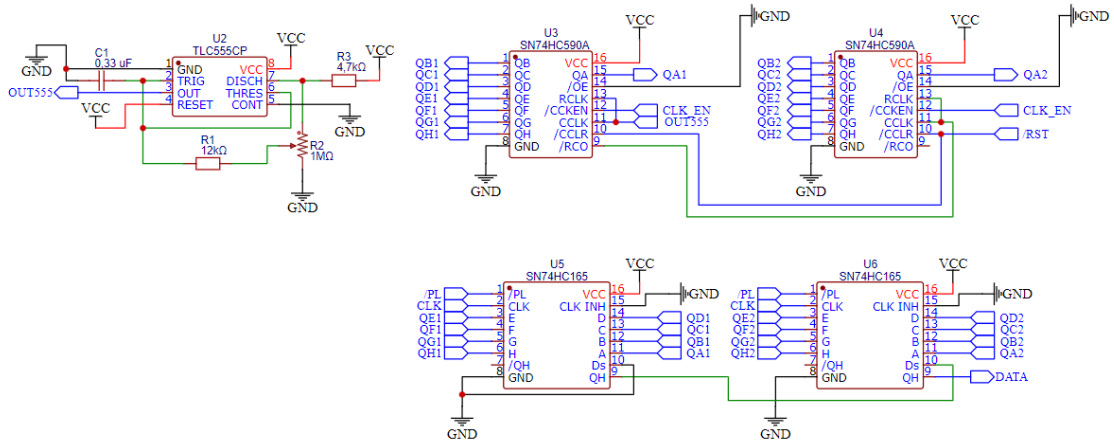


Figura 4. Esquema del grupo de conteo.
Fuente propia.

El circuito del simulador de flujómetro (TLC555), con esquema mostrado en la figura 5, tiene como salida pulsos eléctricos de onda cuadrada. Estos pulsos tienen una frecuencia ajustable (de 2,15 a 152,04 Hz) mediante un potenciómetro, lo que simula un mayor o menor caudal de agua.

Teniendo en cuenta que el fabricante del caudalímetro simulado (YF-B10) proporciona la siguiente ecuación:

$$F = 6 \cdot Q \ - 8$$

Donde:

- F: Frecuencia (en Hz)
- Q: caudal en (L/min)

Se presenta la tabla 1 donde se muestra la relación entre frecuencia de los pulsos de onda cuadrada y el caudal.

Frecuencia (Hz)	Caudal (L/min)	Nota
2,15	0,297	Frecuencia mínima del simulador de flujómetro.
152,04	26,67	Frecuencia máxima del simulador de flujómetro.

Tabla 1. Relación entre frecuencia y caudal

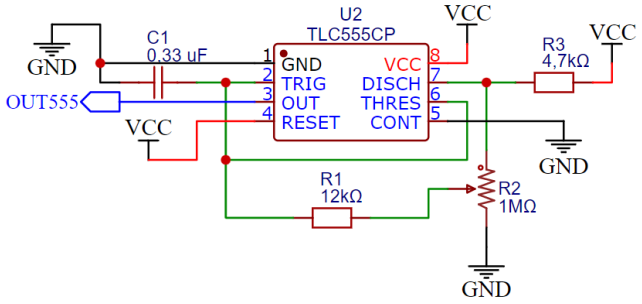


Figura 5. Esquema del simulador de flujómetro.
Fuente propia.

La salida del simulador de flujómetro (TLC555) se conecta a la señal CLK del primero de dos contadores 74HC590 [13]. Los dos contadores, con esquema mostrado en la figura a 6, están conectados en cascada de manera que al ser cada uno de 8 bits, el sistema dispone de un contador de 16 bits.

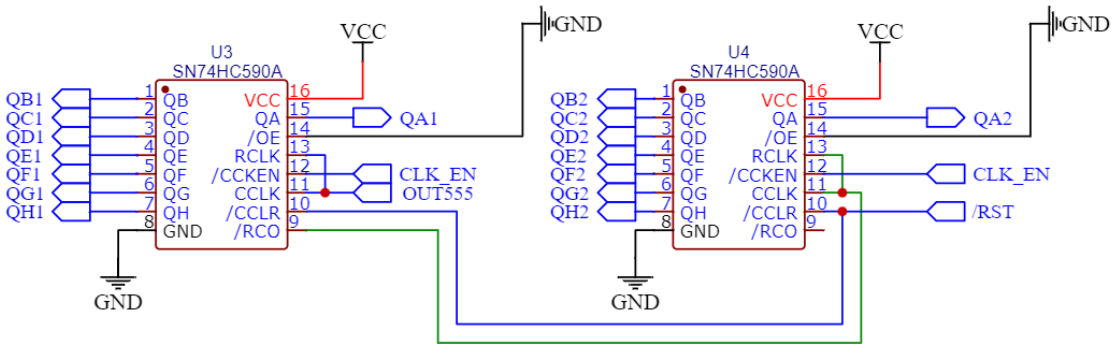


Figura 6. Esquema del contador.
Fuente propia.

Los registros de desplazamiento son dos 74HC165 [14], con esquema mostrado en la figura 7. Se conecta el 74HC590 correspondiente al LSB con el 74HC165 del LSB, y lo mismo para el MSB. Estos circuitos son usados para convertir los datos en paralelo de los contadores a datos en serie, pudiendo así ahorrar patillas en la placa del microcontrolador

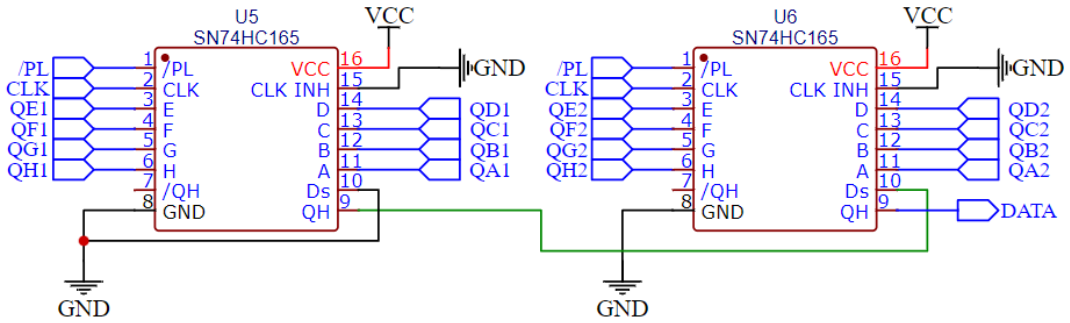


Figura 7. Esquema del registro de desplazamiento.
Fuente propia.

El microcontrolador gestiona las señales de los circuitos integrados descritos de manera que lee en serie el dato del registro de desplazamiento y reinicia a cero los contadores cuando el dato se ha leído.

2.2.3. Interfaz lógica

El microcontrolador necesita conectarse a una red WiFi con Internet, tanto para poder actualizar la fecha y hora como para que el usuario pueda acceder a los datos a través del servidor web. Esta interacción se muestra en la figura 8.

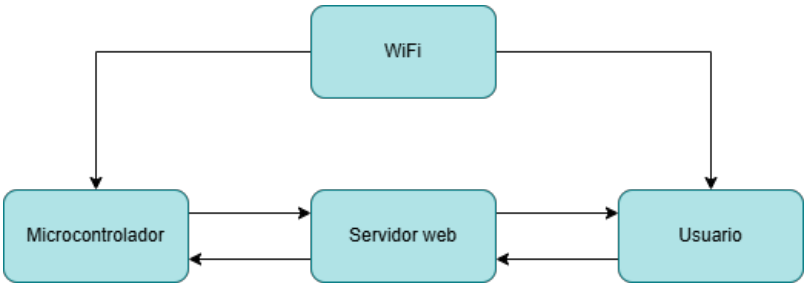


Figura 8. Esquema del grupo interfaz lógica.
Fuente propia.

Una vez que el microcontrolador se conecta a Internet a través de WiFi, si el usuario posee un dispositivo con capacidad de utilizar navegadores web, y el sistema está en modo comunicación WiFi, al conectarse a la misma red WiFi y escribir "alejandrovelatfg.local" se accede al menú mostrado en la figura 9.

Figura 9. Menú del servidor web.
Fuente propia.

Mediante el servidor web se puede apagar o encender la pantalla, apagar las comunicaciones, leer los consumos, leer los parámetros de funcionamiento, o descargar un archivo "hora.csv" que almacena todas las lecturas.

2.2.4. Interfaz física

Consta de una pantalla, una tarjeta SD y botones.

Tanto la pantalla como la tarjeta SD usan el protocolo SPI para transmitir la información.

La pantalla TFT [15], observada en la figura 10, es un modelo de 2,4 pulgadas con una resolución de 240x320 píxeles, que permite mostrar texto e información gráfica con buena legibilidad. Además, funciona a 3,3 V, compatible con el microcontrolador, y tiene un controlador ST7735 que facilita la comunicación SPI.

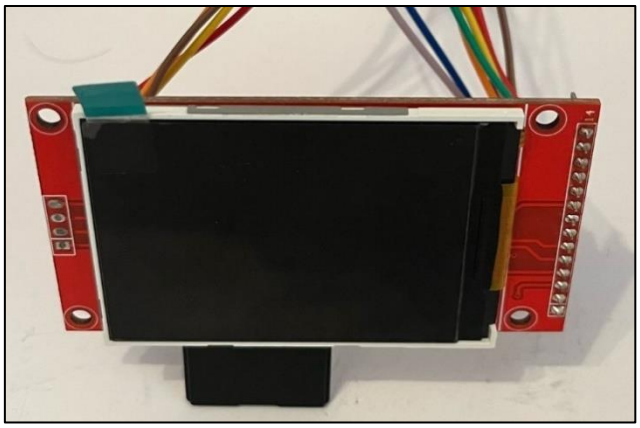


Figura 10. Pantalla TFT.
Fuente propia.

La tarjeta SD utilizada es de tipo microSD con formato FAT32 y capacidad de hasta 4 GB. Opera también a 3,3 V y permite almacenar tanto las lecturas históricas como los archivos de configuración, garantizando un acceso rápido y fiable mediante la interfaz SPI.

En la pantalla TFT se muestra el modo de funcionamiento en el que se encuentra el sistema (Sin comunicaciones, con comunicación WiFi o reposo). También se muestran las últimas ocho lecturas con sus respectivas fecha y litros.

En la tarjeta SD se guarda un archivo ".txt" que contiene la cantidad absoluta de litros que se han leído desde que el dispositivo inició su funcionamiento. También contiene un archivo "hora.csv" que contiene las lecturas de consumo. El esquema de la Pantalla TFT y la tarjeta SD se muestra en la figura 11.

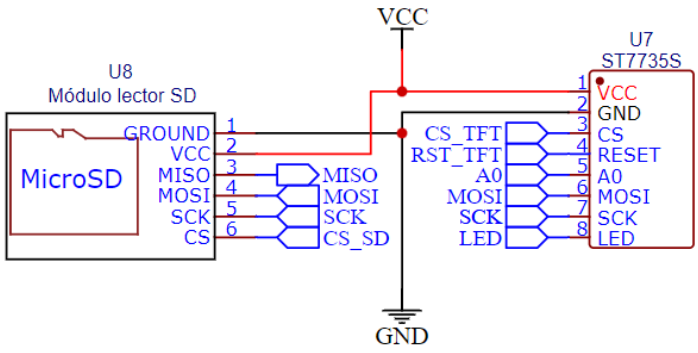


Figura 11. Esquema de la pantalla y el lector SD.
Fuente propia.

También cuenta con tres botones, mostrados en la figura 12 y esquema mostrado en la figura 13.

- Botón azul: Activa o desactiva el modo reposo.
- Botón verde: Activa o desactiva la comunicación WiFi.

- Botón blanco: Enciende o apaga la pantalla.

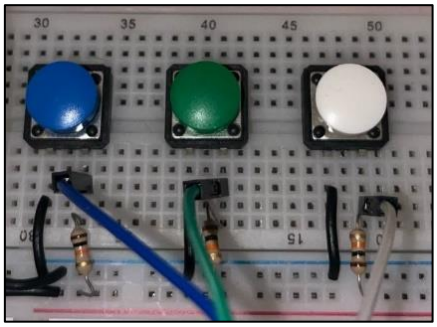


Figura 12. Botones.
Fuente propia.

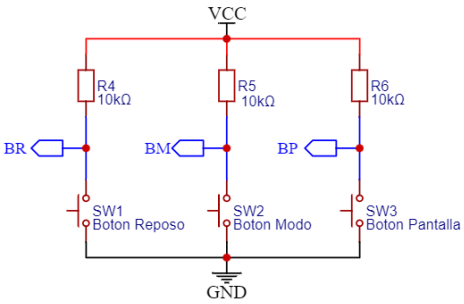


Figura 13. Esquema de los botones.
Fuente propia.

2.2.5. Modos de funcionamiento

El sistema está diseñado para realizar de forma continua, siempre que haya una conexión USB, la lectura del consumo de agua a través de un contador conectado al registro de desplazamiento. Este registro permite al microcontrolador ESP32 obtener el número de pulsos generados por el sensor de caudal.

La información se procesa y se almacena de forma periódica en un archivo denominado "hora.csv", el cual contiene el valor acumulado de litros consumidos y la marca temporal (fecha y hora) asociada a cada registro.

Un fragmento de este archivo se muestra en la figura 14. El formato del almacenamiento de los datos es el siguiente:

XXXXXXXX,YYYYYYYYYY

Donde:

- XXXXXXXX : Representa el valor en litros de la lectura. Siendo el máximo de 99999999 L.
- YYYYYYYYYY : Representa el valor en formato epoch de la fecha de la lectura.

42	00003760,1744110658
43	00003799,1744110688
44	00003838,1744110718
45	00003877,1744110748

Figura 14. Fragmento del archivo "hora.csv".
Fuente propia.

El sistema dispone de tres modos de funcionamiento que determinan cómo se gestionan las lecturas y la interacción con el usuario:

- Modo sin comunicaciones (modo por defecto):

  Al iniciar, el sistema se encuentra en este modo. Se realizan las lecturas de los pulsos y se guardan automáticamente en la tarjeta SD. El microcontrolador permanece a la espera de una posible interacción mediante los botones físicos, permitiendo al usuario seleccionar otro modo de operación si lo desea.

- Modo comunicación WiFi:

  Activado mediante la pulsación del botón verde. En este modo, el microcontrolador habilita un servidor web accesible desde cualquier dispositivo conectado a la misma red WiFi. A través de este servidor, el usuario puede consultar las lecturas almacenadas, descargar los datos o interactuar con otras funciones implementadas en la interfaz web.

- Modo reposo (modo de bajo consumo):

  Activado al pulsar el botón azul. En este estado, el microcontrolador entra en un modo de suspensión para reducir significativamente su consumo energético [2]. Se despierta automáticamente cada cuatro minutos para realizar una nueva lectura y guardarla en la tarjeta SD. Durante este modo, se deshabilitan las interrupciones de los botones verde y blanco para evitar cambios de estado accidentales.

Capítulo III

3. Diseño del sistema

En este apartado se describe la estructura general del sistema desarrollado, integrando tanto la circuitería como el programa necesario para su funcionamiento.
El diseño sigue un enfoque modular que facilita el mantenimiento y futuras ampliaciones. La combinación de sensores, procesamiento y comunicación permite obtener una solución fiable para el monitoreo del consumo de agua.

3.1. Circuitos

La circuitería es la base física del sistema y está compuesto por sensores, módulos de procesamiento, almacenamiento y visualización.

Cada componente ha sido seleccionado considerando criterios como fiabilidad, bajo consumo y facilidad de integración.

Aquí se detallan las conexiones y la lógica de funcionamiento entre los elementos que permiten realizar la medición y una interacción eficiente con el usuario.

3.1.1. Lista de componentes

A continuación, en la tabla 2 se muestran los distintos componentes y una breve descripción. La lista completa de componentes se encuentra en el anexo B.

Componente	Modelo	Función
Microcontrolador	ESP32-WROOM-32	Control principal del sistema
Contador binario	74HC590	Cuenta los pulsos generados
Registro paralelo a serie	74HC165	Permite leer los datos del contador
Generador de pulsos	TLC555	Simula los pulsos de un caudalímetro
Pantalla	TFT SPI 2.4" ST7735	Muestra los datos al usuario, y almacena datos en la tarjeta SD.
Botones	Pulsadores estándar	Control de funciones del sistema

Tabla 2. Lista de componentes

3.1.2. Descripción y justificación técnica

3.1.2.1. Microcontrolador ESP32

El microcontrolador ESP32-WROOM-32, mostrado en la figura 15, es el microcontrolador seleccionado para este prototipo de supervisor de agua. Este módulo, basado en el microcontrolador ESP32 de Espressif Systems, ofrece las capacidades necesarias de procesamiento y conectividad requeridas para el sistema.

Figura 15. WROOM-32.
Fuente mouser.es

Las características técnicas más relevantes para este proyecto incluyen:

- Procesador de doble núcleo Xtensa LX6 a 240 MHz, que permite manejar eficientemente las tareas concurrentes del sistema.
- 512 KB de SRAM y 4 MB de memoria flash, suficiente para almacenar el programa y los mensajes a enviar.
- Conectividad WiFi integrada para el servidor web local.
- Múltiples conexiones de entrada y/o salida para la interfaz con los circuitos externos, de las cuales se usan un total de 17.
- Bajo consumo en modo reposo.

El microcontrolador ESP32-WROOM-32 realiza las siguientes funciones principales:

- Lectura de los pulsos generados por el simulador de flujómetro y contadores.
- Gestión de los registros de desplazamiento 74HC590 y 74HC165 para almacenar los conteos.
- Control de la pantalla TFT para visualización local.
- Creación de un servidor web para acceso remoto a los datos.

- Manejo de los tres botones de control (reposo, pantalla y comunicaciones).

La selección de este microcontrolador se justifica por:

- Conectividad WiFi integrada sin necesidad de circuitería adicional.
- Las numerosas patillas de entrada y/o salida facilitan la conexión con los circuitos externos (74HC590 y 74HC165).
- Los modos de bajo consumo son esenciales para reducir el consumo energético.

3.1.2.2. Contador binario 74HC590

El sistema dispone de dos circuitos integrados 74HC590 configurados en cascada para formar un contador binario de 16 bits. Esta solución permite el conteo continuo de pulsos generados por el simulador de flujómetro, manteniendo su operación independientemente del estado del microcontrolador principal.

Cada 74HC590 es un contador binario sincrónico de 8 bits con las siguientes características relevantes para el proyecto:

- Registro de salida incorporado que mantiene el último valor contado
- Entrada de reloj (CLK) para incrementar el contador
- Salida de acarreo (RCO) para conexión en cascada
- Reinicio maestro (MR) para reiniciar el conteo
- Funcionamiento con alimentación de 5 V compatible con señales de 3,3 V

El esquema de este dispositivo es mostrado en la figura 16.

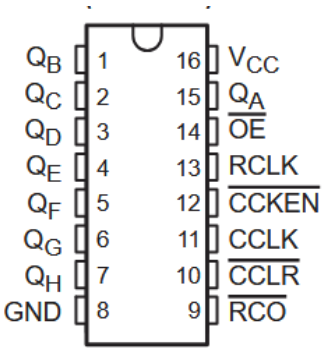


Figura 16. Distribución de patillas del 74HC590.
Fuente Texas Instruments.

La conexión de este dispositivo se muestra en la figura 17, y su funcionamiento es el siguiente:

1. El primer 74HC590 recibe directamente los pulsos del temporizador 555
2. Cuando alcanza su valor máximo (255), activa su salida RCO
3. Esta señal RCO incrementa el segundo contador 74HC590
4. El sistema completo puede contar hasta 65.535 pulsos (2^16 - 1) cuyo valor es el de las señales QA1-QH1, QB2-QH2.

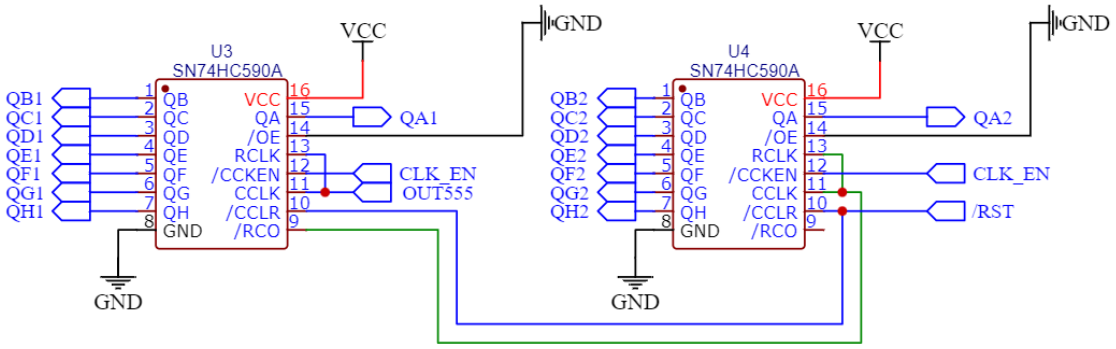


Figura 17. Esquema del contador.
Fuente propia.

La selección de este circuito integrado se justifica por:

- Funcionamiento autónomo respecto al microcontrolador
- Capacidad de mantener el conteo durante periodos de reposo del microcontrolador ESP32
- Precisión garantizada al ser una solución mediante circuitería
- Bajo consumo de 4,021 µA en funcionamiento continuo, calculado en Anexo A.

El diseño permite al microcontrolador ESP32 leer los valores de los contadores cuando lo requiere, sin interferir con el proceso de conteo. La frecuencia máxima de funcionamiento (25 MHz) asegura un amplio margen frente a la frecuencia de pulsos generada por el simulador de flujómetro (158,6 Hz máxima frecuencia).

3.1.2.3. Registro de desplazamiento 74HC165

El sistema emplea dos registros de desplazamiento 74HC165 para leer los 16 bits de los contadores 74HC590 mediante comunicación serie. Estos circuitos integrados permiten una interfaz eficiente con el microcontrolador ESP32 usando solo 3 patillas de control.

Principales características en esta implementación:

- Dos registros en cascada para manejar 16 bits (8 bits cada uno)
- Conversión paralelo-serie para minimizar conexiones al microcontrolador ESP32
- Compatibilidad con niveles lógicos de 3,3 V del microcontrolador
- Frecuencia máxima de funcionamiento de 25 MHz

El esquema de este dispositivo es mostrado en la figura 18.

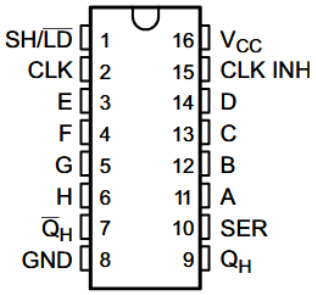


Figura 18. Distribución de patillas del 74HC165.
Fuente Texas Instruments.

Su conexionado se muestra en la figura 19, y su funcionamiento se realiza en los siguientes pasos:

1. Carga paralela: Al activar SH/LD, se captura instantáneamente el valor de los contadores
2. Desplazamiento serie: El microcontrolador lee los bits secuencialmente mediante pulsos de CLK
3. Cadena de datos: La salida QH del primer registro se conecta a la entrada Ds del segundo

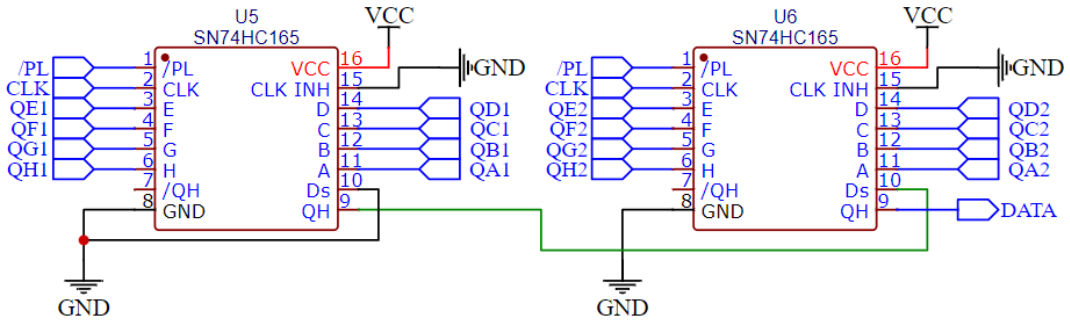


Figura 19. Esquema del registro de desplazamiento.
Fuente propia.

Justificación para este proyecto:

- Reduce de 16 a 3 las conexiones necesarias al microcontrolador ESP32 (SH/LD, CLK, QH)
- Permite lectura no destructiva de los contadores
- Mantiene la integridad de datos durante la transferencia
- Consumo energético mínimo, 0,69 nA, calculado en Anexo A.

Esta solución optimiza la interfaz con el microcontrolador manteniendo precisión y eficiencia energética, características esenciales para el prototipo de contador de agua inteligente.

3.1.2.4. Simulador de flujómetro

El sistema emplea un circuito integrado TLC555 configurado en modo astable como generador de pulsos, el circuito usado es mostrado en la figura 20. Este componente cumple la función de simular la señal digital emitida por un caudalímetro del tipo YF-B10, ajustable en un rango de frecuencias entre 2,15 y 152,04 Hz. Esto permite validar el sistema de conteo y adquisición sin necesidad de disponer físicamente del sensor durante las fases iniciales de diseño, prueba y calibración.

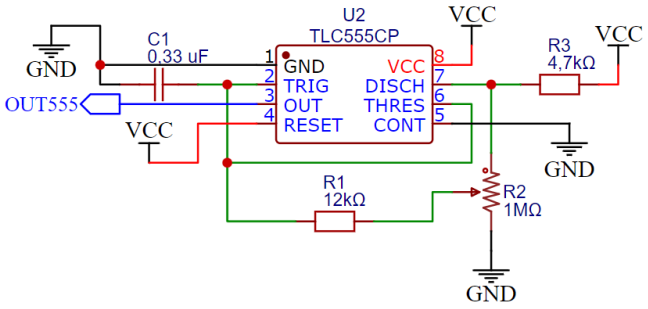


Figura 20. Esquema del simulador de medidor de flujo.
Fuente propia.

El TLC555 es una versión CMOS del clásico temporizador 555, especialmente seleccionada por su capacidad de trabajar con tensiones de alimentación tan bajas como 2 V, lo que lo hace plenamente compatible con el sistema basado en lógica de 3,3 V, la cual es la alimentación aportada por la placa en la que está el microcontrolador ESP32.

Las características más relevantes del TLC555 para este proyecto son:

- Funcionamiento en un amplio rango de tensión (2 V a 15 V)
- Baja corriente de consumo, ideal para aplicaciones de bajo consumo
- Capacidad de generar señales cuadradas estables en modo astable
- Compatibilidad directa con entradas lógicas de microcontroladores modernos

La configuración astable seleccionada permite generar una onda cuadrada continua cuya frecuencia está determinada por una resistencia fija R1, una resistencia variable R2 (formada por un potenciómetro de 1 MΩ en serie con una resistencia de 12 kΩ) y un condensador de 0,33 µF de tipo poliéster. La ecuación que rige el funcionamiento es:

$$f = \frac{1{,}44}{(R_1 + 2 \cdot R_2) \cdot C}$$

Donde:

- f es la frecuencia de salida (Hz)
- R_1 = 4,7 kΩ (resistencia fija)
- R_2 = 12 kΩ + potenciómetro de 1 MΩ
- C = 0,33 µF (condensador de poliéster)

Con estos valores, la frecuencia de salida puede ajustarse manualmente entre 2,15 Hz (0,297 L/min) y 152,04 Hz (26,67 L/min), cubriendo así el rango de frecuencias típico del sensor YF-B10, que varía entre aproximadamente 5,68 Hz y 142 Hz en función del flujo de agua.

La señal de salida del TLC555 es enviada a la entrada de reloj de un contador binario 74HC590, que actúa como sistema de conteo de eventos, permitiendo medir y registrar el número de pulsos generados.

Justificación para este proyecto:

- Simplicidad de diseño: no requiere programación ni configuración digital
- Fiabilidad: generación de pulsos estable y precisa
- Compatibilidad con 3,3 V, sin necesidad de adaptadores de nivel lógico
- Versatilidad: permite validar el sistema completo sin depender de un sensor físico real

Gracias a estas características, el TLC555 se presenta como una solución adecuada y eficiente para simular el comportamiento dinámico del caudalímetro YF-B10 en el entorno de desarrollo del proyecto.

3.1.2.5. Pantalla TFT 2,4" con controlador ST7735

El sistema implementa una pantalla TFT de 2,4 pulgadas con controlador ST7735, que integra además un lector de tarjetas SD. Esta pantalla permite la visualización local de los datos del contador de agua, ofreciendo una interfaz gráfica intuitiva y de bajo consumo energético gracias al control de la retroiluminación. En la figura 21 se muestra su esquema.

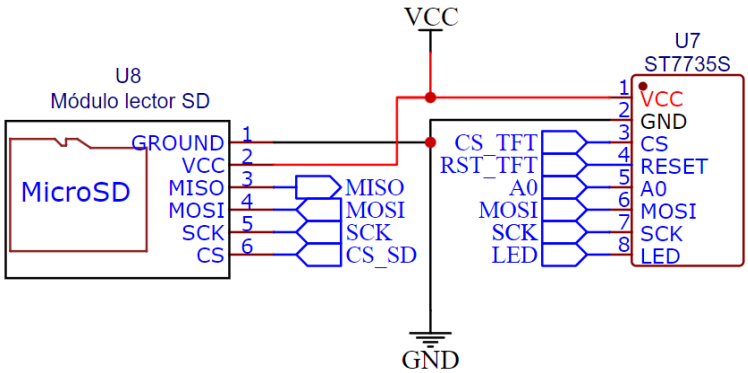


Figura 21. Esquema de la pantalla TFT.
Fuente propia.

La pantalla TFT con controlador ST7735 y lector SD presenta las siguientes características relevantes para este proyecto:

- Resolución de 240x320 píxeles con soporte para 65 K colores
- Interfaz SPI de 4 hilos para comunicación con el microcontrolador ESP32 (compartida con la SD)
- Lector de tarjeta SD integrado compatible con archivos CSV para almacenamiento de lecturas
- Señal de retroiluminación controlado mediante salida PWM del microcontrolador ESP32
- Alimentación a 3,3 V, compatible directamente con el microcontrolador
- Ángulo de visión superior a 160 grados

La implementación se realiza mediante:

1. Conexión SPI utilizando las señales SCLK, MOSI, DC y CS del microcontrolador ESP32
2. Control de retroiluminación mediante PWM en la patilla 13
3. Uso de la librería "TFT_eSPI" optimizada para el controlador ST7735
4. Integración del lector SD mediante la librería "SD.h" para almacenar las lecturas de consumo
5. Visualización de información en dos áreas principales, mostrado en la figura 22:
    - Últimas siete lecturas realizadas
    - Modo de funcionamiento actual

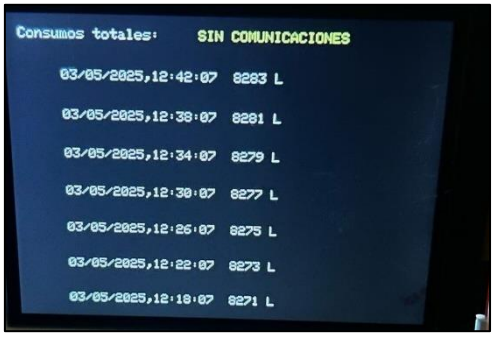


Figura 22. Información en la pantalla TFT.
Fuente propia.

Justificación para este proyecto:

- Visualización clara e inmediata del consumo

- Control preciso de brillo mediante PWM (desde 0% hasta 100%)
- Capacidad de apagado completo para ahorro energético
- Compatibilidad total con los niveles lógicos del microcontrolador ESP32

El diseño permite al microcontrolador ESP32 gestionar eficientemente la pantalla, alternando entre estados de máxima visibilidad y bajo consumo según las necesidades operativas. La resolución de 240x320 píxeles proporciona espacio suficiente para mostrar toda la información relevante de forma organizada y legible.

3.1.2.6. Interfaz de botones

El sistema implementa tres botones, mostrados en la figura 23, con polarización positiva para el control de funciones básicas del contador de agua inteligente. Cada botón sigue un esquema de conexión idéntico y robusto que garantiza un funcionamiento fiable. El esquema de los botones se muestra en la figura 24.

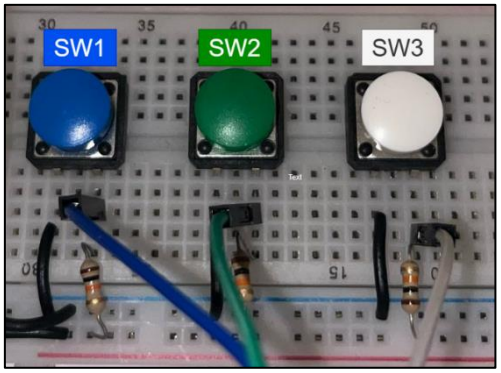


Figura 23. Botones.
Fuente propia.

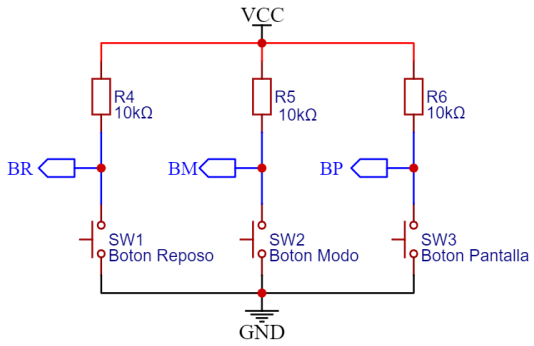


Figura 24. Esquema de los botones.
Fuente propia.

Cada botón utiliza la siguiente disposición:

- Resistencia polarización positiva de 10 kΩ conectada a 3,3 V
- Terminal superior del botón conectada al punto medio (entre resistencia y patilla al microcontrolador)
- Terminal inferior del botón conectada directamente a 0 V
- Patilla del ESP32 configurado como entrada con detección de flanco descendente

Asignación de botones y funciones:

1. Botón de modo reposo (botón azul)

- Función: Alterna entre modo activo y bajo consumo
- Conectado a la patilla 32 del microcontrolador

2. Botón de pantalla (botón blanco)

   - Función: Controla el estado de la pantalla TFT
   - Conectado a la patilla 35 del microcontrolador

3. Botón de comunicaciones (botón verde)

   - Función: Gestiona la conectividad WiFi
   - Conectado a la patilla 34 del microcontrolador

Justificación para este proyecto:

- Mayor inmunidad al ruido que las resistencias internas
- Consistencia en el comportamiento de todos los botones
- Fácil diagnóstico de fallos

Detalles técnicos:

- Todas las resistencias de polarización positiva son de 1/4W ±5%
- Patillas del microcontrolador configurados como entradas digitales
- Implementado filtrado de efecto rebote de 200 ms
- Consumo despreciable en estado inactivo (<1µA por botón)

Esta configuración de los botones, combinada con la lógica del programa, proporciona una interfaz de usuario robusta y fiable.

3.1.3. Esquema eléctrico

A continuación, se presenta el esquema eléctrico completo del sistema en la figura 23, representado también en el Anexo C. Se muestra la interconexión entre los diferentes componentes: el microcontrolador ESP32, el generador de pulsos con el temporizador TLC555, los contadores binarios 74HC590, los registros de desplazamiento 74HC165, la pantalla TFT que incorpora lector de tarjeta SD, y los botones de control.

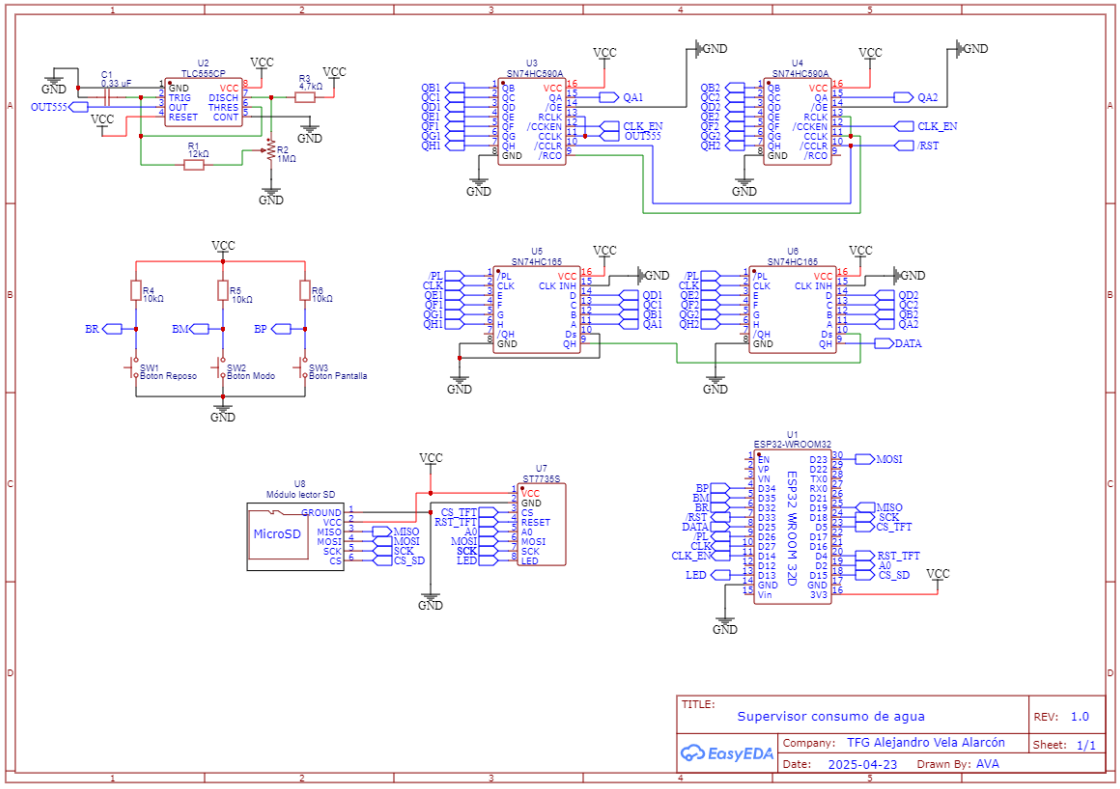


Figura 23. Esquema eléctrico del sistema.
Fuente propia.

3.1.4. Montaje y prototipado

El montaje del sistema se ha realizado sobre una placa de prototipado, lo que ha permitido una construcción modular y fácilmente modificable durante el desarrollo y las pruebas. Esta elección ha facilitado el diagnóstico de errores, la reorganización de conexiones y la sustitución rápida de componentes.

Durante el proceso de prototipado, se ha seguido el esquema eléctrico descrito en el apartado anterior, procurando mantener una distribución ordenada y clara de los elementos para minimizar errores de conexión.

El microcontrolador ESP32 utilizado se presenta en una placa adaptadora con doble sistema de conexión, que permite la inserción directa en la placa de desarrollo. Esta característica ha aportado una mayor comodidad a la hora de conectar los distintos módulos y reorganizar el cableado según las necesidades de prueba.

En la figura 24 se muestra una imagen del montaje físico del sistema.

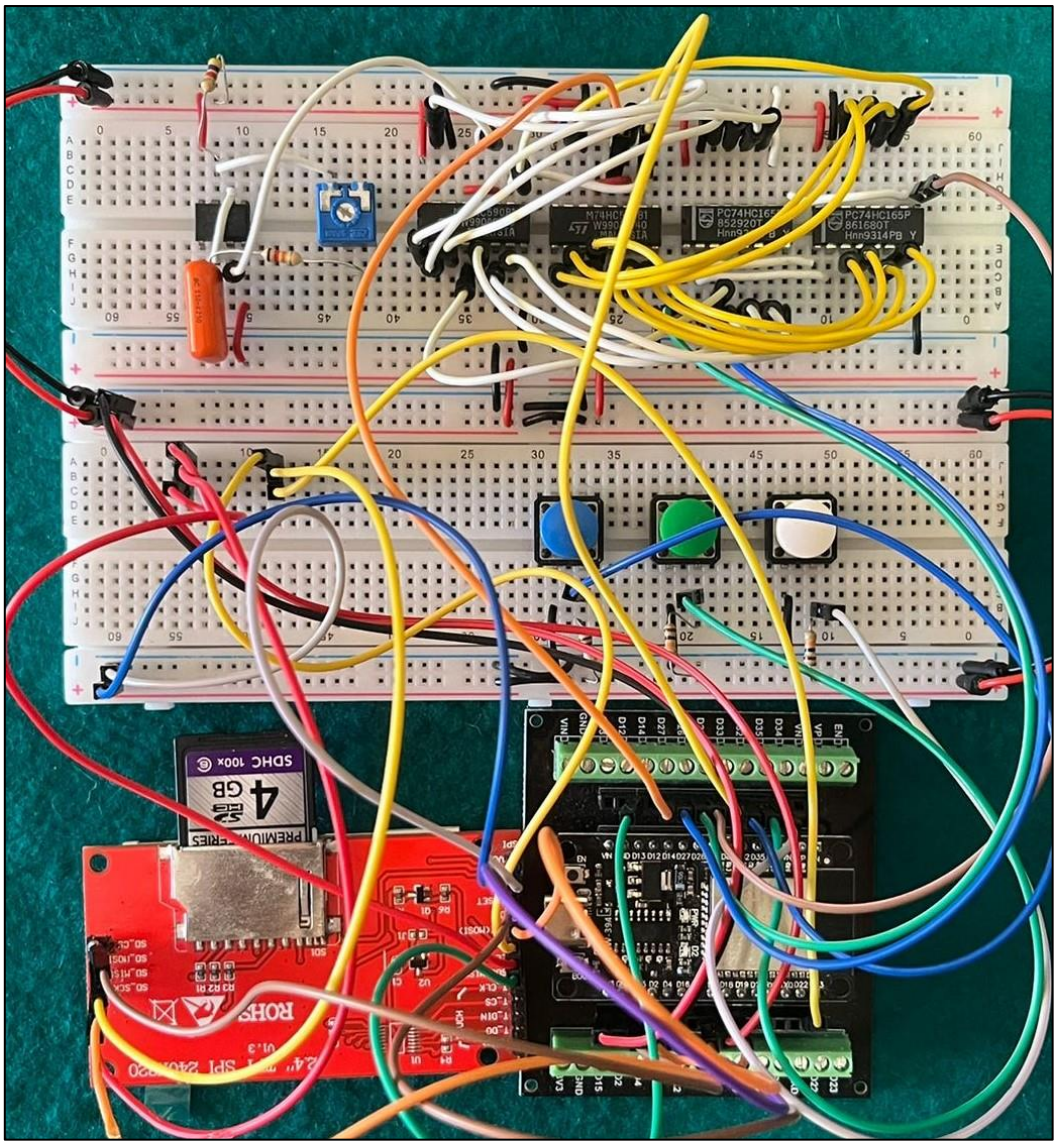


Figura 24. Montaje físico del sistema.
Fuente propia.

3.2. Programa

En este apartado se describe el código de programación desarrollado para el funcionamiento del sistema, implementado sobre el microcontrolador ESP32.

El programa ha sido elaborado en lenguaje C++, utilizando el entorno de desarrollo PlatformIO integrado en Visual Studio Code, y basado en el entorno de Arduino. Esta combinación de herramientas proporciona una plataforma flexible, eficiente y ampliamente documentada para el desarrollo de sistemas embebidos.

El programa gestiona la lectura de los pulsos acumulados en la lógica externa, el control de los modos de bajo consumo, la actualización de la información en la pantalla TFT y en el servidor web, así como la interacción con el usuario a través de los botones físicos.

A continuación, se presenta el código fuente dividido en fragmentos explicando de manera detallada la función de cada sección y su relación con el comportamiento general del sistema. Esta metodología facilita la comprensión del flujo de ejecución y de las decisiones técnicas adoptadas durante el desarrollo. La totalidad del código se encuentra en el anexo D.

3.2.1. Inclusión de librerías

En la figura 25 se incluyen las librerías necesarias para el funcionamiento del prototipo. Se utilizan varias librerías que permiten la interacción con diferentes componentes del sistema:

- WiFiClient: Para establecer la conexión Wi-Fi del ESP32. (Línea 6).
- WebServer: Para configurar y gestionar un servidor web local que permita la visualización de los datos a través de una página web. (Línea 7).
- ESPmDNS: Para implementar mDNS, lo que facilita la creación de una URL local personalizada para acceder a la página web. (Línea 8).
- TFT_eSPI: Para controlar y manejar la pantalla TFT donde se visualizarán los datos. (Línea 9).
- SD: Para interactuar con la tarjeta SD, necesaria para almacenar información y registros. (Línea 10).
- time.h: Para gestionar el tiempo y la fecha del sistema. (Línea 11).
- TimeLib: Para la manipulación avanzada de fechas y horas en distintos formatos. (Línea 12).

```
5   //  *** INCLUSIÓN DE LIBRERÍAS ***
6   #include <WiFiClient.h>   // Librería necesaria para la conexión WiFi
7   #include <WebServer.h>    // Librería necesaria para crear una página web
8   #include <ESPmDNS.h>      // Librería necesaria para mDNS (personalizar URL de la página web)
9   #include <TFT_eSPI.h>     // Librería necesaria para la pantalla TFT SPI
10  #include <SD.h>           // Librería necesaria para la lectura/escritura de la tarjeta SD
11  #include <time.h>         // Librería necesaria para actualizar y controlar la fecha y hora
12  #include <TimeLib.h>      // Librería necesaria para manejar distintos formatos de tiempo
```

Figura 25. Inclusión de librerías.
Fuente propia.

3.2.2. Declaración de variables

En este apartado se definen las variables necesarias para controlar los diferentes aspectos del sistema. En la figura 26 se definen:

- Variables de control de patillas: Se definen las patillas de entrada/salida para los componentes conectados, como los botones y el control del brillo de la pantalla. Las patillas del protocolo SPI se declaran en la librería de la pantalla TFT. (Líneas 15 a 25).
- Variable de control de la pantalla TFT: Se define el objeto "TFT" que controla las funciones de la librería del controlador ST7735, que es el controlador de la pantalla TFT. (Línea 28).
- Variables de Wi-Fi: Se especifican el SSID y la contraseña para conectar el ESP32 a la red Wi-Fi. (Líneas 31 y 32).
- Variables de fecha y hora: Se define el servidor NTP y las estructuras necesarias para obtener y manejar la fecha y hora. (Líneas 35 a 38).

```
14  // *** DECLARACIÓN DE VARIABLES ***
15  #define LOAD_PIN 26   // Carga paralela 74HC165 (/PL)
16  #define CLK_PIN 27    // Reloj 74HC165          (CLK)
17  #define CLK_EN_PIN 14 // Habilita el reloj del 74HC590
18  #define DATA_PIN 25   // Salida del 74HC165 MSB (Data)
19  #define RESET_PIN 33  // Reset de los 74HC590   (/RST)
20
21  #define PIN_BLANCO 34 // Pin del botón asignado a la pantalla (BP)
22  #define PIN_VERDE 35  // Pin del botón asignado al modo       (BM)
23  #define PIN_AZUL 32   // Pin del botón asignado al reposo     (BR)
24
25  #define PIN_LED 13    // Pin que ajusta el brillo de la pantalla
26
27  // Variables de la pantalla
28  TFT_eSPI TFT;   // Objeto con el que se llama a las funciones de la pantalla
29
30  // Variables de conexión WiFi
31  const char* ssid = "SSID-generico";    // SSID de WiFi al que se conecta el ESP32
32  const char* pass = "PASS-generico";  // Contraseña del WiFi al que se conecta el ESP32
33
34  // Variables de fecha y hora
35  const char* ntpServer = "0.es.pool.ntp.org";  // Servidor NTP de donde se obtiene la fecha y hora
36  char timeStringBuff[50];
37  struct tm timeinfo;                // Objeto con el que se llama a las funciones de fecha y hora
38  String tiempoEnEpoch = "";
```

Figura 26. Declaración de variables (I).
Fuente propia.

En la figura 27 se definen:

- Variables de temporizador: Se configura un temporizador que permite medir el consumo de agua a intervalos específicos. (Líneas 41, 42 y 43).
- Variables de botones: Se definen las variables para detectar las pulsaciones de los botones que controlan la pantalla, el modo y el reposo. (Líneas 46 a 50).
- Variables de consumo de agua: Se utilizan arreglos para almacenar el historial de consumos y las fechas asociadas a esos consumos. (Líneas 53 a 56).
- Servidor web: Se crea un objeto de servidor para permitir la visualización remota de los datos a través de una página web. (Línea 59).

```
40  // Variables del temporizador que registra el consumo de agua
41  hw_timer_t *timer = NULL;   // Declarar el temporizador
42  volatile bool timerFlag = false;    // Bandera para saber cuándo se activa la interrupción
43  int configuracionsegundoslectura = 240;   // 4 minutos entre lecturas
44
45  // Variables de botones
46  bool BotonVerde = false;    // Variable booleana pulsación botón verde
47  bool BotonBlanco = false;   // Variable booleana pulsación botón blanco
48  bool BotonAzul_isAsleep = false;    // Variable booleana pulsación botón Azul
49  int ModoComunicacion = 0;   // Variable para detectar si está o no comunicación WiFi
50  bool is_pantallaEncendida = true;   // Variable para detectar si la pantalla está o no encendida
51
52  // Variables para almacenar los consumos
53  int pulseHistory[7] = {0};    // Array para almacenar los últimos consumos (Litros)
54  String pulseTimes[7] = {""};  // Array para almacenar los últimos consumos (Fecha y hora)
55  int litrosCount = 0;          // Se crea una variable para llevar la cuenta de litros
56  char litrosTotales[9];        // Se crea una variable para que el valor siempre sea de 8 dígitos
57
58  // Variables WebServer
59  WebServer server(80); // Crea una instancia del servidor en el puerto 80
```

Figura 27. Declaración de variables (II).
Fuente propia.

3.2.3. Declaración de funciones

3.2.3.1. Funciones de visualización y manejo de datos

Este subapartado describe las funciones utilizadas para gestionar y visualizar los datos en la pantalla TFT, así como las operaciones de actualización de los arreglos de consumo y tiempo.

En la figura 28 se definen:

- Función "mostrarUltimosConsumos()". (Líneas 62 a 74).

  Esta función se encarga de mostrar los últimos 7 consumos de agua y sus marcas de tiempo en la pantalla TFT.

La función limpia la pantalla, luego imprime el título "Consumos totales" y, utilizando un bucle “for”, muestra los valores de los últimos consumos y sus tiempos correspondientes en diferentes posiciones en la pantalla.

- Función “mostrarArribaDerecha()”. (Líneas 76 a 81).

Permite mostrar un texto en la parte superior derecha de la pantalla, usando un color personalizado.

El texto proporcionado como parámetro se coloca en la parte superior derecha de la pantalla con un tamaño de texto pequeño y el color especificado.

- Función “RotarActualizarArray()”. (Lineas 83 a 88).

Actualiza un arreglo de 7 elementos, desplazando los valores hacia atrás y colocando el nuevo valor en la primera posición.

Esta función asegura que el arreglo siempre contenga los últimos 7 valores, rotando los elementos anteriores hacia atrás y añadiendo el nuevo valor al principio del arreglo.

- Función “RotarActualizarTiempo()”. (Líneas 90 a 98).

Actualiza un arreglo con las fechas y horas más recientes, obteniendo la hora local actual.

La función obtiene la fecha y hora actual y las guarda en el arreglo “Tiempo”, desplazando los valores existentes hacia atrás para hacer espacio al nuevo registro.

```
62  void mostrarUltimosConsumos() {
63      TFT.fillScreen(TFT_BLACK);
64      TFT.setTextColor(TFT_WHITE);
65      TFT.setTextSize(2);
66      TFT.setCursor(10, 10);
67      TFT.setTextSize(1);
68      TFT.println("Consumos totales:");
69      for (int i = 0; i < 7; i++) {
70          TFT.setCursor(40, 40 + i * 30);
71          TFT.print(pulseTimes[i]);
72          TFT.printf("  %d L", pulseHistory[i]);
73      }
74  }
75
76  void mostrarArribaDerecha(String textoquemostrar, int color){
77    TFT.setTextColor(color, TFT_BLACK);
78    TFT.setCursor(140, 10);
79    TFT.setTextSize(1);
80    TFT.println(textoquemostrar);
81  }
82
83  void RotarActualizarArray(int Array[7], int Actualizador){
84    for(int i = 5; i >= 0; i--){
85      Array[i+1] = Array[i];
86    }
87    Array[0] = Actualizador;
88  }
89
90  void RotarActualizarTiempo(String Tiempo[7]){
91    if (getLocalTime(&timeinfo)) {
92      strftime(timeStringBuff, sizeof(timeStringBuff), "%d/%m/%Y,%H:%M:%S", &timeinfo);
93    for(int i = 5; i >= 0; i--){
94      Tiempo[i+1] = Tiempo[i];
95    }
96    Tiempo[0] = String(timeStringBuff);
97    }
98  }
```

Figura 28. Funciones de visualización y manejo de datos.
Fuente propia.

3.2.3.2. Funciones de conversión de fechas y tiempos

Este subapartado describe dos funciones clave que permiten convertir fechas y horas entre formatos legibles y el formato de "*timestamp*" (marca de tiempo) utilizado en el sistema, se muestran en la figura 29:

- Función "convertirFechaATimestamp()". (Líneas 102 a 118).

  Convierte una fecha y una hora, dadas como cadenas de texto, en un *timestamp* (número de segundos desde el 1 de enero de 1970).

  Esta función toma dos parámetros: una fecha en formato "dd/mm/yyyy" y una hora en formato "hh:mm:ss". Primero extrae los valores de día, mes, año, hora, minuto y segundo de las cadenas, y luego los organiza en un objeto "tmElements_t". Utiliza la función "makeTime()" para convertir estos valores a

un "timestamp" (un número entero que representa los segundos desde la medianoche del 1 de enero de 1970).

- Función convertirTimestamp(). Líneas (121 a 129).

Convierte un "timestamp" en una fecha y hora legibles.

Esta función toma un "timestamp" (en formato de cadena) como parámetro y lo convierte en una fecha y hora legible utilizando la función "localtime()" para obtener la estructura "tm". Luego, se utiliza "strftime()" para formatear esta fecha en una cadena con el formato "dd/mm/yyyy hh:mm:ss".

```
101  // Función para convertir fecha y hora a timestamp
102  int convertirFechaATimestamp(String fecha, String hora) {
103    int dia, mes, ano, horaInt, minuto, segundo;
104    sscanf(fecha.c_str(), "%d/%d/%d", &dia, &mes, &ano);
105    sscanf(hora.c_str(), "%d:%d:%d", &horaInt, &minuto, &segundo);
106
107    // Crear un objeto tmElements_t para almacenar la fecha y hora
108    tmElements_t tm;
109    tm.Day = dia;
110    tm.Month = mes;
111    tm.Year = ano - 1970;  // Year es el número de años desde 1970
112    tm.Hour = horaInt;
113    tm.Minute = minuto;
114    tm.Second = segundo;
115
116    // Usamos makeTime para convertir a timestamp
117    return makeTime(tm);
118  }
119
120  // Función que convierte un String timestamp a fecha legible
121  String convertirTimestamp(String timestamp) {
122    time_t tiempo = (time_t)timestamp.toInt();
123    struct tm *timeinfo = localtime(&tiempo);
124
125    char buffer[30];
126    strftime(buffer, sizeof(buffer), "%d/%m/%Y %H:%M:%S", timeinfo);
127
128    return String(buffer);
129  }
```

Figura 29. Funciones de conversión de fechas y tiempos.
Fuente propia.

3.2.3.3. Función de lectura de archivos CSV para obtener lecturas entre dos fechas

En este subapartado se describe la función "obtenerLecturasEntreFechas()", la cual permite obtener las lecturas registradas en un archivo CSV entre dos fechas específicas proporcionadas por el usuario.

La función "obtenerLecturasEntreFechas()" recibe dos parámetros de tipo String, correspondientes a la fecha de inicio y la fecha de fin (Línea 132). El objetivo es obtener las lecturas de un archivo CSV entre estos dos valores de tiempo.

En la figura 30 se muestra la validación de fechas:

Primero, se declaran las variables que se usan a lo largo del programa, las cuales son para comprobar el rango, para comprobar cual es la primera lectura dentro del rango y el resultado. (Líneas 134, 135 y 136).

Posteriormente, se convierten las fechas de inicio y fin a "timestamps" utilizando la función "convertirFechaATimestamp()" (Líneas 140 y 141). Luego se realizan varias comprobaciones:

- Se valida que las fechas no sean anteriores al 01/01/2025. (Líneas 142 y 146).
- Se verifica que la fecha de inicio no sea posterior a la fecha de fin. (Línea 150).
- Se valida que la diferencia entre las fechas no supere el límite de 33 días (2.851.200 segundos). (Línea 154).
- Si alguna de estas condiciones no se cumple, se devuelve un mensaje de error. (Líneas 143, 144, 147, 148, 151, 152, 155, 156).

```
131 // Función para leer el archivo CSV y obtener las lecturas entre dos fechas
132 String obtenerLecturasEntreFechas(String fechaInicio, String fechaFin) {
133
134   bool is_rangoCorrecto = false;          // Booleana para bucle explicado más abajo
135   bool is_primeraLecturaFiltrada = false; // Booleana para condicional explicado más abajo
136   String resultado = "Lecturas entre las fechas solicitadas:\n";  // String donde se guardará lo que leerá el usuario
137
138   // Se convierte la fecha de un formato legible a un formato en el que es más fácil comparar (epoch)
139   int timestampInicio = convertirFechaATimestamp(fechaInicio.substring(0, 10), "00:00:00");
140   int timestampFin = convertirFechaATimestamp(fechaFin.substring(0, 10), "23:59:59");
141
142   if(timestampInicio < 1735689600){
143     resultado = "ERROR: Fecha de inicio anterior a 01/01/2025 \n       O formato invalido";
144     return resultado;
145   }
146   else if(timestampFin < 1735689600){
147     resultado = "ERROR: Fecha de inicio anterior a 01/01/2025 \n       O formato invalido";
148     return resultado;
149   }
150   else if(timestampInicio > timestampFin){
151     resultado = "ERROR: Fecha de inicio mayor que fecha de fin";
152     return resultado;
153   }
154   else if(timestampFin - timestampInicio > 2851200){
155     resultado = "ERROR: Diferencia de fechas superior al limite";
156     return resultado;
157   }
```

Figura 30. Función "obtenerLecturasEntreFechas()" (I).
Fuente propia.

En la figura 31 se muestra la declaración de variables a usar para mostrar el mensaje final al usuario (Líneas 159 a 169), y la apertura del archivo donde se guardan todos los datos ("SD.open("/hora.csv")") (Líneas 171 a 174).

```
159  int primeraLecturaFiltrada; // Variable para registrar la primera lectura del rango indicado
160  int ultimaLecturaFiltrada;  // Variable para registrar la ultima lectura del rango indicado
161
162  // Variables para la búsqueda binaria
163  int leftByte = 0;   // La izquierda siempre es 0 ya que lo único que se guarda en ese archivo son las fechas y consumos
164  int rightByte;      // La derecha depende del tamaño del archivo
165  int byteBusqueda;   // Variable que se usa para indicar que byte del archivo se va a leer
166
167  String linea = "";
168  String valor = "";
169  String epoch = "";
170
171  File archivo = SD.open("/hora.csv");  // Abre el archivo hora.csv
172  if (!archivo) {
173    return "No se pudo abrir el archivo.";
174  }
```

Figura 31. Función "obtenerLecturasEntreFechas()" (II).
Fuente propia.

En la figura 32 se muestra la búsqueda binaria que se realiza para encontrar el rango de fechas pedido por el usuario:

- Se abre el archivo CSV y se busca el valor medio (en bytes) dentro del rango de fechas. (Líneas 176 a 192).
- Si el valor medio corresponde a una fecha dentro del rango especificado, se marca el rango como válido. (Líneas 201 a 204).
- Si no, se ajustan los límites de la búsqueda (izquierda o derecha) para continuar la búsqueda binaria. (Líneas 193 a 200).

```
176  rightByte = archivo.size();
177
178  while(!is_rangoCorrecto){
179    if(byteBusqueda != leftByte + (rightByte - leftByte)/2){  // Si busca 2 veces seguidas en el mismo byte, es que no hay rango correcto.
180    byteBusqueda = leftByte + (rightByte - leftByte)/2;       // Siempre busca el valor medio entre izquierda y derecha
181    }
182    else{
183      resultado = "No hay lecturas entre las fechas especificadas";
184      return resultado;
185    }
186
187    archivo.seek(byteBusqueda);     // Así, aunque se pueda perder una linea, aseguramos que está leyendo la linea completa.
188    archivo.readStringUntil('\n');  // No importa que se pierda una linea ya que posteriormente se busca el primer valor.
189
190    linea = archivo.readStringUntil('\n');  // Lee hasta el salto de línea
191    epoch = linea.substring(9,19);          // Se lee el tiempo en epoch para poder comparar con las fechas ingresadas por el usuario.
192
193    if (timestampInicio > epoch.toInt()) {  // Si la fecha de inicio es mayor que la del valor medio
194                                            // Los valores buscados quedan por encima del valor medio
195      leftByte = byteBusqueda;              // Ahora la izquierda es el valor medio
196    }
197    else if (epoch.toInt() > timestampFin){ // Si el valor medio es mayor que la fecha fin
198                                            // Los valores buscados quedan por debajo del valor medio
199      rightByte = byteBusqueda;             // Ahora la derecha es el valor medio
200    }
201    else if (timestampInicio < epoch.toInt() && timestampFin > epoch.toInt()){  // Si el valor medio es mayor que el de inicio
202                                                                                // y menor que el de fin
203      is_rangoCorrecto = true;                                                  // Hemos entrado en el rango
204    }
205  }
```

Figura 32. Función "obtenerLecturasEntreFechas()" (III).
Fuente propia.

En la figura 33 se muestra las operaciones realizadas una vez que el valor medio se encuentra dentro del rango de fechas pedidas:

- Se lee línea por línea del archivo en orden ascendente hasta que se encuentra una fecha más antigua al rango solicitado. Esto sirve para encontrar cual es la primera fecha dentro del rango. (Líneas 207 a 218).

```
207   while(timestampInicio < epoch.toInt() && byteBusqueda > 0){   // Mientras la fecha de inicio sea menor que la que se está leyendo
208     // y byteBusqueda no sea negativo.
209
210     archivo.seek(byteBusqueda);
211     archivo.readStringUntil('\n');
212
213     linea = archivo.readStringUntil('\n');  // Mismo razonamiento que antes, se pueden saltar lineas pero no importa, después se leen.
214     epoch = linea.substring(9,19);          // Se lee al final para hacer la comprobación del "while" al principio.
215
216     byteBusqueda = byteBusqueda - 21;       // Cada mensaje ocupa 21 bytes, por lo que vamos leyendo el mensaje anterior hasta encontrar
217                                             // el primer mensaje.
218   }
```

Figura 33. Función "obtenerLecturasEntreFechas()" (IV).
Fuente propia.

En la figuras 34 y 35 se muestran las operaciones realizadas para obtener los datos del resultado final.

- Se asegura que la posición de búsqueda (byteBusqueda) no sea negativa. Si es menor que 0, se corrige a 0 para evitar errores de lectura en el archivo. (Líneas 220, 221 y 222).

- Se extrae de la línea leída el valor de la lectura (los primeros 8 caracteres), que representa los litros acumulados. (Línea 224).

- Se inicia un bucle que se mantiene mientras las fechas leídas en las líneas sigan dentro del rango, o todavía queden líneas por leer (Línea 226). Dentro del bucle:

    - Se convierte la fecha ("epoch") a un formato legible (día/mes/año y hora), usando la función "convertirTimestamp()". (Línea 230).

    - Se añade al resultado la fecha legible y la lectura correspondiente, para mostrarlo al usuario al final. (Línea 232).

    - Se comprueba si es la primera lectura dentro del rango. Si es así, se guarda el valor en "primeraLecturaFiltrada" y se marca "is_primeraLecturaFiltrada" como "true" para que no vuelva a sobreescribirse. (Líneas 234 a 237).

    - Se actualiza la variable ultimaLecturaFiltrada con el valor actual. Así, al terminar el bucle, contendrá la última lectura del rango. (Línea 238).

- Se lee la siguiente línea del archivo y se actualizan (los litros acumulados y la fecha). (Líneas 240 a 242).

```
220  if(byteBusqueda < 0){
221    byteBusqueda = 0; // Si ha llegado al inicio del archivo, es igual a cero para que no sea negativo.
222  }
223
224  valor = linea.substring(0,8);
225
226  while(timestampFin > epoch.toInt() && archivo.available()){ // Mientras la fecha de fin sea mayor que la que se está leyendo
227                                                              // y queden lineas por leer
228  // (puede pasar que el usuario ingrese una fecha mayor que la ultima registrada)
229
230    String fecha = convertirTimestamp(epoch); // Se pasa de los segundos en epoch a fechas legibles
231
232    resultado += "Fecha: " + fecha + " Lectura: " + valor + "\n"; // Se añade fecha y valor a lo que ve el usuario.
```

Figura 34. Función "obtenerLecturasEntreFechas()" (V).
Fuente propia.

```
234    if(!is_primeraLecturaFiltrada){            // La primera vez que se lee una fecha correcta, es decir, la primera del rango
235      primeraLecturaFiltrada = valor.toInt();  // Se guarda como primera lectura para poder hacer la resta entre primera y ultima
236      is_primeraLecturaFiltrada = true;
237    }
238    ultimaLecturaFiltrada  = valor.toInt();    // Cada vez que se lee, se guarda como ultima lectura.
239
240    linea = archivo.readStringUntil('\n');  // Lee linea a linea
241    valor = linea.substring(0,8);           // Se leen los valores al final, para que no entren valores fuera de rango
242    epoch = linea.substring(9,19);
243  }
```

Figura 35. Función "obtenerLecturasEntreFechas()" (VI).
Fuente propia.

En la figura 36 se muestra como se cierra el archivo cuando ya se ha terminado de elaborar el mensaje, y se añade al final el cálculo de la diferencia de lecturas. Se calcula la diferencia entre la primera y la última lectura del rango y se muestra en el resultado. (Líneas 245 a 250).

```
245    archivo.close();  // Cerramos el archivo
246    resultado += "\n \n \n";
247    resultado += "Litros entre las fechas elegidas = ";
248    resultado += ultimaLecturaFiltrada - primeraLecturaFiltrada;
249    return resultado;
250  }
```

Figura 36. Función "obtenerLecturasEntreFechas()" (VII).
Fuente propia.

3.2.3.4. Funciones HTML

En este apartado se recogen las funciones que generan el código HTML que pone en funcionamiento el servidor web con el que interactúa el usuario.

En la figura 37 se muestra la función "convertirSaltosDeLineaHTML". Esta función recorre todo el texto recibido carácter por carácter. Cada vez que detecta un salto de línea "\n", inserta en su lugar una etiqueta HTML "
". De esta forma, los saltos de línea son visibles al mostrar el texto en una página web HTML. (Líneas 253 a 263).

```
252  // Función para convertir \n a <br> en HTML
253  String convertirSaltosDeLineaHTML(String texto) {
254    String resultado = "";
255    for (int i = 0; i < texto.length(); i++) {
256      if (texto[i] == '\n') {
257        resultado += "<br>";
258      } else {
259        resultado += texto[i];
260      }
261    }
262    return resultado;
263  }
```

Figura 37. Función convertir saltos de línea.
Fuente propia.

En la figura 38 se muestra la función "handleRoot()". Esta función responde a la ruta raíz "/" del servidor. Envía una página "HTML" que actúa como menú principal, ofreciendo enlaces a diferentes funcionalidades del sistema, como controlar la pantalla, gestionar comunicaciones y acceder a lecturas de datos. (Líneas 265 a 269).

```
265  // Funciones para manejar las rutas
266  void handleRoot() {
267    server.send(200, "text/html",
268      "<h1>Menu del supervisor de consumo de agua</h1><ul><li><a href='/dir1'>
269  }
```

Figura 38. Función "handleRoot()".
Fuente propia.

En la figura 39 se muestra la función "handleDir1()". Al acceder a "/dir1", esta función enciende la pantalla ajustando el brillo al máximo mediante "ledcWrite". También actualiza el estado de la variable "is_pantallaEncendida" y muestra un mensaje en el navegador confirmando la acción, junto con un botón para volver al menú principal. (Líneas 271 a 281).

```
271  void handleDir1() { // Activar pantalla
272    String html = "<html><body>";
273    html += "<h1>Encendiendo la pantalla...</h1>";
274    html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
275    html += "</body></html>";
276    server.send(200, "text/html", html);
277
278    ledcWrite(0, 255);
279    is_pantallaEncendida = true;
280    Serial.println("Pantalla encendida");
281  }
```

Figura 39. Función "handleDir1()".
Fuente propia.

En la figura 40 se muestra la función "handleDir2()". Esta función se activa al acceder a "/dir2". Apaga la pantalla reduciendo el brillo a cero y actualiza la

variable “is_pantallaEncendida”. También muestra una página de confirmación con un botón para regresar al menú. (Líneas 283 a 293).

```
283  void handleDir2() {
284    String html = "<html><body>";
285    html += "<h1>Apagando la pantalla...</h1>";
286    html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
287    html += "</body></html>";
288    server.send(200, "text/html", html);
289
290    ledcWrite(0, 0);
291    is_pantallaEncendida = false;
292    Serial.println("Pantalla apagada");
293  }
```

Figura 40. Función "handleDir2()".
Fuente propia.

En la figura 41 se muestra la función “handleDir3()”. Al acceder a “/dir3”, esta función desactiva las comunicaciones estableciendo “ModoComunicacion” en 0. Envía una página de confirmación al navegador con un botón para volver al menú principal. (Líneas 295 a 303).

```
295  void handleDir3() {
296    String html = "<html><body>";
297    html += "<h1>Apagando las comunicaciones...</h1>";
298    html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
299    html += "</body></html>";
300    server.send(200, "text/html", html);
301
302    ModoComunicacion = 0;
303  }
```

Figura 41. Función "handleDir3()".
Fuente propia.

En la figura 42 se muestra la función “handleDir4FormularioLecturas()”. Esta función presenta un formulario “HTML” al usuario para ingresar una fecha de inicio y una de fin, con el objetivo de filtrar lecturas de datos entre esas fechas. Al enviar el formulario, se realiza una solicitud “GET” a la ruta “/lecturas”. (Líneas 305 a 320).

```
305  void handleDir4FormularioLecturas() {
306    String html = "<html><body>";
307    html += "<h2>Filtrar lecturas entre fechas</h2>";
308    html += "<h4>Maximo 30 dias de diferencia entre fechas<h4><br>";
309    html += "<form action='/lecturas' method='GET'>";
310    html += "<label for='inicio'>Fecha de inicio (dd/mm/yyyy):</label><br>";
311    html += "<input type='text' id='inicio' name='inicio' placeholder='Ej: 01/01/2025' required><br><br>";
312    html += "<label for='fin'>Fecha de fin (dd/mm/yyyy):</label><br>";
313    html += "<input type='text' id='fin' name='fin' placeholder='Ej: 01/02/2025' required><br><br>";
314    html += "<input type='submit' value='Filtrar lecturas'>";
315    html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
316    html += "</form>";
317    html += "</body></html>";
318
319    server.send(200, "text/html", html);
320  }
```

Figura 42. Función "handleDir4FormularioLecturas()".
Fuente propia.

En la figura 43 se muestra la función “handleDir5Lecturas()”. Esta función procesa los datos enviados desde el formulario de fechas. Verifica que ambas fechas estén presentes y llama a “obtenerLecturasEntreFechas()” para obtener las lecturas correspondientes (Líneas 322 a 332). Luego, convierte los saltos de línea en el resultado a etiquetas “
” para una correcta visualización en HTML y envía la respuesta al navegador (Líneas 333 a 340).

```
322  void handleDir5Lecturas() {
323    String fechaInicio = server.arg("inicio");
324    String fechaFin = server.arg("fin");
325    String resultado;
326
327    if (fechaInicio != "" && fechaFin != "") {
328      resultado = obtenerLecturasEntreFechas(fechaInicio, fechaFin);
329    } else {
330      resultado = "<h2>Error: Debes ingresar ambas fechas.</h2>";
331    }
332
333    resultado = convertirSaltosDeLineaHTML(resultado);
334    String html = "<html><body>";
335    html += resultado;
336    html += "<br><a href='/'>Volver al formulario</a>";
337    html += "</body></html>";
338
339    server.send(200, "text/html", html);
340  }
```

Figura 43. Función "handleDir5Lecturas()".
Fuente propia.

En la figura 44 se muestra la función “handleDir6()”. Al acceder a “/dir6”, esta función muestra al usuario el parámetro “configuracionsegundoslectura”, que indica el intervalo de tiempo entre lecturas. También proporciona un botón para regresar al menú principal. (Líneas 342 a 350).

```
342  void handleDir6() {
343    String html = "<html><body>";
344    html += "<h2>Segundos entre lecturas = ";
345    html += configuracionsegundoslectura;
346    html += "</h2>";
347    html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
348    html += "</body></html>";
349    server.send(200, "text/html", html);
350  }
```

Figura 44. Función "handleDir6()".
Fuente propia.

3.2.3.5. Funciones para manejar los circuitos integrados de conteo

En la figura 45 se muestra la función "resetCounters()". Esta función pone a cero los contadores de los circuitos integrados. El procedimiento consiste en:

- Poner la patilla de RESET a nivel bajo (LOW). (Línea 354).
- Esperar 10 microsegundos para asegurar que el pulso de RESET sea detectado correctamente. (Línea 355).
- Volver a poner la patilla de RESET en nivel alto (HIGH), completando así el ciclo de reinicio. (Línea 356).

Este reinicio es necesario para limpiar el contador antes de empezar una nueva lectura de datos.

```
352  // Funciones para los circuitos integrados
353  void resetCounters() {
354    digitalWrite(RESET_PIN, LOW);
355    delayMicroseconds(10);
356    digitalWrite(RESET_PIN, HIGH);
357  }
```

Figura 45. Función "resetCounters()".
Fuente propia.

En la figura 46 se muestra la función "readCounters()" la cual funciona de la siguiente manera:

1. Inhibición del reloj (Línea 360):
   Se pone "CLK_EN_PIN" en "HIGH" para detener temporalmente el reloj de los contadores.

2. Captura de datos (Líneas 362–364):
   Se pone "LOAD_PIN" en "LOW" para capturar los valores actuales de los contadores. Tras un pequeño retardo (1 ms), se vuelve a poner en "HIGH" para fijar los datos.

3. Reinicio del contador (Línea 366):

Se llama a "resetCounters()" para dejar el sistema listo para una nueva cuenta.

4. Reanudación del reloj (Línea 368):
   Se pone "CLK_EN_PIN" en "LOW" para reanudar el conteo normal.

5. Lectura en serie (Líneas 371–377):
   Se inicializa "data" en 0. En un bucle de 16 iteraciones, se lee cada bit desde "DATA_PIN" (MSB primero) y se genera manualmente un pulso de reloj ("CLK_PIN" a "HIGH", luego "LOW" con 1 ms de retardo).

6. Retorno del resultado (Línea 378):
   Se devuelve el valor de 16 bits almacenado en "data".

```
359  uint16_t readCounters() {
360    digitalWrite(CLK_EN_PIN, HIGH); // 1. Inhibe el reloj de los contadores
361    delay(1);
362    digitalWrite(LOAD_PIN, LOW);    // 2. Carga los datos en el 74HC165 (LOAD a LOW)
363    delay(1);
364    digitalWrite(LOAD_PIN, HIGH);   // Vuelve a desactivar la carga
365    delay(1);
366    resetCounters();                // 3. Reinicia los contadores
367    delay(1);
368    digitalWrite(CLK_EN_PIN, LOW);  // 4. Reanuda el reloj de los contadores
369
370
371    uint16_t data = 0;
372    for (int i = 0; i < 16; i++) {  // 5. Lee los 16 bits en serie (MSB first)
373      data |= (digitalRead(DATA_PIN) << (15 - i));
374      digitalWrite(CLK_PIN, HIGH);
375      delay(1);
376      digitalWrite(CLK_PIN, LOW);
377    }
378    return data;                    // 6. Retorno de la información
379  }
```

Figura 46. Función "readCounters()".
Fuente propia.

3.2.3.6. Manejo de interrupciones

En la figura 47 se muestran las variables definidas para el efecto rebote de los pulsadores. El "debounceDelay" o retardo del rebote, evita lecturas falsas causadas por rebotes eléctricos, ignorando señales muy seguidas que no son pulsaciones reales.

Se definen variables para almacenar el último instante en el que ocurrió una interrupción de cada botón (Blanco, Verde, Azul). (Líneas 384 a 387).

```
381  // ** INTERRUPCIONES **
382
383  // *** Variables para el debounce ***
384  volatile unsigned long lastInterruptTimeBlanco = 0;
385  volatile unsigned long lastInterruptTimeVerde = 0;
386  volatile unsigned long lastInterruptTimeAzul = 0;
387  const unsigned long debounceDelay = 200;  // Tiempo de debounce en ms
```

Figura 47. Variables para el "debounce".
Fuente propia.

En la figura 48 se muestra que ocurre cuando se detecta una pulsación del botón Blanco:

- Se aplica el filtro de rebotes en el botón para que no haya falsas pulsaciones ("debounce"). (Líneas 391, 392 y 393).
- Se alterna el estado de encendido/apagado de la pantalla. (Líneas 394 a 401).
- Si se enciende, además, se actualiza el contenido mostrado ("most7arUltimosConsumos"). (Línea 402).

```
389  // Botón para la pantalla
390  void IRAM_ATTR handleInterruptBlanco() {
391    unsigned long currentTime = millis();
392    if (currentTime - lastInterruptTimeBlanco > debounceDelay) {
393      lastInterruptTimeBlanco = currentTime;
394      if (is_pantallaEncendida) {
395        ledcWrite(0, 0);
396        is_pantallaEncendida = false;
397        Serial.println("Pantalla apagada");
398      } else {
399        ledcWrite(0, 255);
400        is_pantallaEncendida = true;
401        Serial.println("Pantalla encendida");
402        mostrarUltimosConsumos();
403      }
404    }
```

Figura 48. Interrupción del botón blanco.
Fuente propia.

En la figura 49 se muestra que ocurre cuando se pulsa el botón verde:

- Se aplica el filtro de rebotes en el botón para que no haya falsas pulsaciones ("debounce"). (Líneas 409, 410 y 411).
- Se incrementa el modo de comunicación, ciclando entre 0 y 1 (modos disponibles). (Línea 412).

```
407  // Botón para modo de comunicación
408  void IRAM_ATTR handleInterruptVerde() {
409    unsigned long currentTime = millis();
410    if (currentTime - lastInterruptTimeVerde > debounceDelay) {
411      lastInterruptTimeVerde = currentTime;
412      ModoComunicacion = (ModoComunicacion + 1)%2;
413    }
414  }
```

Figura 49. Interrupción del botón verde.
Fuente propia.

En la figura 50 se muestra que ocurre cuando se pulsa el botón Azul:

- Se aplica el filtro de rebotes en el botón para que no haya falsas pulsaciones ("debounce"). (Líneas 418 y 419).
- Se alterna el estado de una variable ("BotonAzul_isAsleep"), que controla el modo reposo [3] en el bucle principal. (Línea 420).

```
416  // Botón para modo reposo
417  void IRAM_ATTR handleInterruptAzul() {
418    unsigned long currentTime = millis();
419    if (currentTime - lastInterruptTimeAzul > debounceDelay) {
420      BotonAzul_isAsleep = !BotonAzul_isAsleep;
421    }
422  }
```

Figura 50. Interrupción del botón azul.
Fuente propia.

En la figura 51 se muestra la interrupción por tiempo "onTimer()".

Esta interrupción se dispara periódicamente a través de un temporizador:

- Cambia la bandera "timerFlag" a true. (Línea 426).

Esto permite en el bucle principal "loop()" saber que ha llegado el momento de realizar la lectura de los contadores.

```
424  // Interrupción lectura
425  void IRAM_ATTR onTimer() {
426  timerFlag = true;
427  }
```

Figura 51. Interrupción de lectura.
Fuente propia.

3.2.4. Configuración inicial del sistema "setup()"

La función "setup()" se encarga de configurar todos los módulos y componentes que van a utilizarse durante el funcionamiento del sistema. Se ejecuta una única

vez al encender o reiniciar el microcontrolador. A continuación, se detallan sus partes.

En la figura 52 se inician los siguientes apartados:

- Inicialización de comunicación serie: (Línea 431).

  Se arranca la comunicación por el puerto serie "Serial.begin(9600)", lo que permite enviar mensajes desde el microcontrolador al ordenador para depurar o mostrar información importante en tiempo real.

- Configuración del bus SPI: (Líneas 433 y 434).

  Se inicializa el bus SPI con "SPI.begin(...)", indicando qué patillas corresponden a SCK, MISO, MOSI y CS. Posteriormente, se establece el modo de comunicación "SPI_MODE0", necesario para interactuar con la tarjeta SD y otros dispositivos externos.

- Control del brillo LED: (Líneas 436 a 440).

  Se configura la patilla de control de la retroiluminación de la pantalla mediante PWM ("ledcSetup" y "ledcAttachPin"). Se enciende el LED de la pantalla al 100% "ledcWrite(0, 255)" y se hace una pequeña pausa de 2 segundos para estabilizar el sistema.

```
429  void setup() {
430
431    Serial.begin(9600);             // Inicia el puerto serie para poder monitorizar desde un PC
432
433    SPI.begin(18, 19, 23, 15);      // Inicia el protocolo SPI -> (SCK, MISO, MOSI, CS)
434    SPI.setDataMode(SPI_MODE0);     // SPI MODO 0
435
436    pinMode(PIN_LED, OUTPUT);
437    ledcSetup(0, 5000, 8);
438    ledcAttachPin(PIN_LED, 0);
439    ledcWrite(0, 255);
440    delay(2000);
```

Figura 52. Función "setup()" (I).
Fuente propia.

En la figura 53 se muestra el montaje y lectura inicial de la tarjeta SD:

Se intenta iniciar la tarjeta SD. Si tiene éxito, se busca el archivo "/ultimalectura.txt" que guarda el contador de litros consumidos. Si el archivo existe, se lee el valor inicial de litros; si no, se sigue con un valor predeterminado. (Líneas 443 a 456).

```
442   // Inicia la tarjeta SD
443   if(!SD.begin(15)){
444     Serial.println("Tarjeta SD no iniciada");
445     // return;
446   }
447
448   File file = SD.open("/ultimalectura.txt", FILE_READ);
449   if (file) {
450     litrosCount = file.parseInt();
451     file.close();
452     Serial.print("Valor leído desde el archivo: ");
453     Serial.println(litrosCount);
454   } else {
455     Serial.println("No se encontró el archivo. Usando valor por defecto.");
456   }
```

Figura 53. Función "setup()" (II).
Fuente propia.

En la figura 54 se inician los siguientes apartados:

- Inicialización y mensaje de pantalla:

  Se configura la pantalla TFT: se inicializa, se establece su rotación (orientación del contenido mostrado) y se limpia mostrando un fondo negro. Después, se imprime el mensaje "ENCENDIENDO..." en el centro de la pantalla. (Líneas 459 a 463).

- Conexión a la red WiFi:

  Se conecta a la red WiFi utilizando el SSID y contraseña definidos en el programa. Durante la conexión, se muestra un punto por cada intento, y una vez conectado, se imprime la IP local obtenida. Esto permite que el microcontrolador sea accesible dentro de la red. (Líneas 466 a 473).

```
458   // Inicia la pantalla
459   TFT.begin(5);
460   TFT.setRotation(2);
461   TFT.fillScreen(TFT_BLACK);
462   TFT.setCursor(120,120);
463   TFT.print("ENCENDIENDO...");
464
465   // Inicia WiFi con SSID y contraseña
466   WiFi.begin(ssid, pass);
467   while(WiFi.status() != WL_CONNECTED) {
468     Serial.print(".");
469     delay(10);
470   }
471
472   Serial.print("Se ha conectado al wifi con la ip: ");
473   Serial.println(WiFi.localIP());
```

Figura 54. Función "setup()" (III).
Fuente propia.

En la figura 55 se inician los siguientes apartados:

- Sincronización de la hora mediante “NTP”:

Se configura el sistema para obtener la hora actual desde un servidor de tiempo en Internet ("NTP"). Se ajusta la zona horaria ("configTime") y se comprueba que se ha obtenido la hora correctamente. Esta hora es crítica para registrar eventos en el sistema de forma precisa. (Líneas 476 a 480).

- Configuración de patillas digitales:

Se declaran como salidas o entradas las patillas necesarias para controlar y leer de los circuitos conectados (contadores, botones, etc.).

  - LOAD_PIN, CLK_PIN, RESET_PIN, CLK_EN_PIN → Salidas. (Líneas 483, 484, 486, 487).
  - DATA_PIN → Entrada. (Línea 481).
  - PIN_BLANCO, PIN_VERDE, PIN_AZUL → Entradas internas con resistencia Pull-Up. (Líneas 489, 490, 491).

- Reinicio de contadores:

Se llama a "resetCounters()" para asegurar que los contadores conectados empiezan en cero, eliminando posibles residuos de lecturas anteriores. (Línea 493).

```
475   // Configurar la obtención de la hora desde el servidor NTP
476   configTime(3600, 3600, ntpServer);
477
478   if (!getLocalTime(&timeinfo)) {   // Espera a obtener la hora
479     Serial.println("Fallo al obtener la hora");
480   }
481
482   // Declaración de pines
483   pinMode(LOAD_PIN, OUTPUT);
484   pinMode(CLK_PIN, OUTPUT);
485   pinMode(DATA_PIN, INPUT);
486   pinMode(RESET_PIN, OUTPUT);
487   pinMode(CLK_EN_PIN, OUTPUT);
488
489   pinMode(PIN_BLANCO, INPUT_PULLUP);  // Pin botón blanco como entrada
490   pinMode(PIN_VERDE, INPUT_PULLUP);   // Pin botón verde como entrada
491   pinMode(PIN_AZUL, INPUT_PULLUP);    // Pin botón azul como entrada
492
493   resetCounters();  // Inicia contadores en 0
```

Figura 55. Función "setup()" (IV).
Fuente propia.

En la figura 56 se muestra la configuración del servidor web y rutas HTTP (Líneas 496 a 515). Se define qué función manejará cada ruta del servidor ("/", "/dir1", "/dir2"...).
La función de estas rutas está previamente descrita en el apartado 3.2.3.4. A excepción de la ruta "dir7", que descarga el archivo hora.csv. (Líneas 503 a 514).

```
495  // Define la rutas y la función que la manejará
496  server.on("/", handleRoot);
497  server.on("/dir1", handleDir1);
498  server.on("/dir2", handleDir2);
499  server.on("/dir3", handleDir3);
500  server.on("/dir4", handleDir4FormularioLecturas);
501  server.on("/lecturas", handleDir5Lecturas);
502  server.on("/dir6", handleDir6);
503  server.on("/dir7",   HTTP_GET, []() {
504    File file = SD.open("/hora.csv", FILE_READ);
505    if (file) {
506      server.sendHeader("Content-Type", "text/csv");
507      server.sendHeader("Content-Disposition", "attachment; filename=\"hora.csv\"");
508
509      server.streamFile(file, "text/csv");  // Enviar el archivo como respuesta
510
511      file.close();
512    } else {
513      server.send(404, "text/plain", "Archivo no encontrado");
514    }
515  });
```

Figura 56. Función "setup()" (V).
Fuente propia.

En la figura 57 se inician los siguientes apartados:

- Configuración de mDNS:

  Se inicializa un servicio de nombre de dominio local ("alejandrovelatfg.local"), facilitando el acceso al microcontrolador sin necesidad de conocer su dirección IP. (Líneas 518 a 521).

- Inicialización de servidor HTTP:

  Se arranca el servidor ("server.begin()"), que escuchará las solicitudes desde cualquier navegador de la red. (Línea 524).

```
517  // Iniciar mDNS con el nombre de dominio "alejandrovelatfg"
518  if (!MDNS.begin("alejandrovelatfg")) {
519    Serial.println("Error al iniciar mDNS");
520    return;
521  }
522
523  // Inicia el servidor
524  server.begin();
```

Figura 57. Función "setup()" (VI).
Fuente propia.

En la figura 58 se inician los siguientes apartados:

- Configuración de interrupciones:

  Se configuran:

  - Temporizador:

Se inicia un temporizador de circuitería interna que cada cuatro minutos ("configuracionsegundoslectura") genera una interrupción para registrar el consumo de agua. (Líneas 532 a 538).

- Botones físicos:

Se asignan interrupciones a los botones Blanco, Verde y Azul para ejecutar acciones inmediatas al ser pulsados (encender/apagar pantalla, cambiar modo de comunicación, modo reposo del sistema). (Líneas 541, 542, 543).

- Mostrar últimos consumos registrados:

Finalmente, se llama a "mostrarUltimosConsumos()" para visualizar en la pantalla los últimos datos leídos de consumo de agua, permitiendo así iniciar el sistema con información actualizada. (Línea 545).

```
526   // ** Interrupciones **
527   Serial.println("Iniciando interrupciones...");
528
529   // Temporizador para registrar el consumo en la SD
530
531   // Configurar el temporizador (0: primer temporizador, 80: divisor de frecuencia para 1 MHz)
532   timer = timerBegin(0, 80, true);
533   // Adjuntar la función de manejo de interrupción al temporizador
534   timerAttachInterrupt(timer, &onTimer, true);
535   // Establecer el temporizador para dispararse cada 4 minutos
536   timerAlarmWrite(timer, configuracionsegundoslectura * 1000000, true);
537   // Iniciar el temporizador
538   timerAlarmEnable(timer);
539
540   // Interrupciones botones
541   attachInterrupt(digitalPinToInterrupt(PIN_BLANCO), handleInterruptBlanco, RISING);
542   attachInterrupt(digitalPinToInterrupt(PIN_VERDE), handleInterruptVerde, RISING);
543   attachInterrupt(digitalPinToInterrupt(PIN_AZUL), handleInterruptAzul, RISING);
544
545   mostrarUltimosConsumos();
546 }
```

Figura 58. Función "setup()". (VII).
Fuente propia.

3.2.5. Bucle principal de ejecución "loop()"

La función loop() es la principal ejecución del programa, donde se ejecutan continuamente las tareas en función del estado del sistema. Este bloque revisa condiciones como el modo de suspensión, el temporizador, y la comunicación con el servidor.

En la figura 59 la declaración de la variable "pulseCount" (Línea 549), que almacena los pulsos que se registran del caudalímetro, y el modo reposo, que funciona de la siguiente manera:

Si la variable "BotonAzul_isAsleep" es verdadera (Línea 552), significa que se ha pulsado el botón azul y el sistema debe entrar en modo de bajo consumo. Se

muestra un mensaje de advertencia, se esperan 5 segundos (Líneas 553 y 554) y luego se configura el microcontrolador para:

- Despertar mediante una interrupción en la patilla 32 (botón azul). (Línea 558).
- Despertar también tras un tiempo determinado (240 segundos). (Línea 560).

El sistema entra en "esp_light_sleep_start()" (Línea 563), lo que permite reducir significativamente el consumo eléctrico manteniendo algunos periféricos activos.

```
548  void loop() {
549    int pulseCount = 0;     // Variable que registra todos los pulsos leídos
550
551    // Si el estado es "reposo" (isAsleep es verdadero), entraremos en modo de reposo
552    if (BotonAzul_isAsleep) {
553      mostrarArribaDerecha("Entrando en modo reposo...", TFT_ORANGE);
554      delay(5000);
555      mostrarArribaDerecha("MODO REPOSO               ", TFT_RED);
556
557      // Configuramos la interrupción para despertar el ESP32 cuando el botón se presione
558      esp_sleep_enable_ext0_wakeup(GPIO_NUM_32, 0); // GPIO32 activará la interrupción (nivel bajo)
559      // Configura temporizador de despertar (en microsegundos)
560      esp_sleep_enable_timer_wakeup(240 * 1000000); // 4 minutos
561
562      // Entramos en modo de reposo profundo
563      esp_light_sleep_start();
564      onTimer();
565      BotonAzul_isAsleep = true;
566    }
```

Figura 59. Función "loop()" (I).
Fuente propia.

En las figuras 60 y 61, se muestra las operaciones realizadas cuando se activa el temporizador de lecturas cada cuatro minutos:

1. Lectura de pulsos del contador mediante readCounters(). (Línea 572).
2. Cálculo de litros: los pulsos se transforman a litros de agua usando una fórmula proporcional (litrosCount += pulseCount /340), que depende de las características del sensor de caudal. (Línea 573).
3. Actualización de historial de consumos (pulseHistory) y de tiempos (pulseTimes) para mantener los últimos registros en memoria. (Líneas 574 y 575).
4. Visualización en pantalla: se actualiza la interfaz TFT con la nueva información. (Línea 581).
5. Conversión de tiempo a formato epoch para registrar la fecha/hora con precisión. (Línea 582).

6. Almacenamiento en SD:

    - El archivo /hora.csv guarda los litros actuales y la hora en epoch para mantener un registro histórico. (Líneas 585 a 591).

    - El archivo /ultimalectura.txt guarda el valor más reciente del contador para continuar desde ahí si se reinicia el sistema. (Líneas 596 a 605)

```
567  if (timerFlag) {
568
569    timerFlag = false;  // Restablecer la bandera
570    Serial.println("Interrupción de 4 minutos activada");
571
572    pulseCount = readCounters();
573    litrosCount += pulseCount/340;                // Ecuacion dada por el caudalímetro
574    RotarActualizarArray(pulseHistory, litrosCount);
575    RotarActualizarTiempo(pulseTimes);
576
577    sprintf(litrosTotales, "%08d", litrosCount); // Rellena con 0, un tamaño total de 8, decimal.
578
579    Serial.printf("Valor de litros actuales: %d L\n", litrosCount);
580
581    mostrarUltimosConsumos();         // Actualizar la pantalla TFT
582    tiempoEnEpoch = mktime(&timeinfo);// Convierte en epoch la fecha actual para escribirla en la SD
```

Figura 60. Función "loop()" (II).
Fuente propia.

```
584      // Guarda en la SD los últimos litros de lectura, fecha y hora
585      File file = SD.open("/hora.csv", FILE_APPEND);  // Abre el archivo en modo agregar
586      if (file) {
587        file.print(litrosTotales);
588        file.print(",");
589        file.println(tiempoEnEpoch);
590        file.close();                    // Cerrar el archivo
591        Serial.println("Hora guardada en hora.csv");
592      }
593      else {
594        Serial.println("Error al abrir el archivo para escribir.");
595      }
596      file = SD.open("/ultimalectura.txt", FILE_WRITE);
597      if (file){
598        file.println(litrosCount);
599        file.close();
600        Serial.println("Valor de litros guardado en ultimalectura.txt");
601      }
602      else {
603        Serial.println("Error al abrir el archivo para escribir.");
604      }
605    }
```

Figura 61. Función "loop()" (II).
Fuente propia.

En la figura 62 se muestra como se realiza la gestión de las comunicaciones:

- Si "ModoComunicacion == 0": no hay ningún tipo de comunicación activa, y se muestra el mensaje "SIN COMUNICACIONES". (Líneas 608, 609 y 610).

- Si “ModoComunicacion == 1”: se atienden las solicitudes al servidor web, permitiendo al usuario consultar los datos desde un navegador. Se muestra "COMUNICACION WIFI" en pantalla para informar visualmente del estado. (Líneas 611, 612 y 613).

```
607     if(!BotonAzul_isAsleep){
608       if(ModoComunicacion == 0){ // No hay comunicaciones
609         mostrarArribaDerecha("SIN COMUNICACIONES", TFT_YELLOW);
610       }
611       if(ModoComunicacion == 1){
612         server.handleClient();    // Gestiona las peticiones al servidor web
613         mostrarArribaDerecha("COMUNICACION WIFI ", TFT_CYAN);
614       }
615     }
616   }
```

Figura 62. Función "loop()" (III).
Fuente propia.

Capítulo IV

4. Características técnicas

Este apartado recoge las características más relevantes del sistema en términos de eficiencia energética, capacidad de almacenamiento, frecuencia de lectura y otros aspectos técnicos que definen el comportamiento del prototipo bajo condiciones normales de funcionamiento.

Estas características son o bien proporcionadas por "Espressif" en la hoja de datos del microcontrolador "ESP32", o calculadas en el Anexo A.

4.1. Consumo energético

El ESP32 permite reducir significativamente el consumo energético mediante el uso de modos de bajo consumo. En este proyecto se emplea específicamente el modo "light sleep", que permite mantener algunos periféricos activos y reducir el consumo mientras se conserva la capacidad de responder a interrupciones externas, como las generadas por los botones.

Las estimaciones de consumo para los distintos estados son:

- Modo activo con WiFi encendido: Dependiendo de la carga de red, el sistema tiene un consumo máximo de hasta 220 mA con la pantalla encendida, y de 100 mA con la pantalla apagada. Datos calculados en Anexo A.
- Modo reposo con pantalla apagada: Consumo aproximado de 0,808 mA, lo que representa un ahorro energético notable sin comprometer la capacidad de respuesta. Dato calculado en Anexo A.

4.2. Almacenamiento

- 4 GB de memoria SD que permite almacenar lecturas durante toda la vida útil del sistema.

4.3. Lecturas

- Lecturas de 21 bytes en archivo "hora.csv". Las lecturas destinan 8 bytes al valor del total de litros leídos y 10 bytes a la fecha en formato "epoch".
- Almacenamiento de hasta 65.535 pulsos (16 bits).

- Frecuencia de muestreo de 4 minutos. El microcontrolador despierta cada 4 minutos y lee la cantidad de pulsos almacenada en los contadores.
- Lectura de datos realizada en 4,0074 ms. El tiempo que se tarda en obtener la información del contador es de 4,0074 ms. Dato calculado en Anexo A.
- Caudal máximo legible de 48,18 litros/minuto. Dato calculado en Anexo A.

4.4. Tasa de error y vida útil

- Tasa de error menor al 0,1%. Se comete un fallo de hasta 0,000735 litros/minuto. Dato calculado en Anexo A.
- Vida útil teórica de más de 7 años con consumo durante 24 horas al día a caudal máximo. Dato calculado en Anexo A.

Capítulo V

5. Interfaz de usuario

El sistema desarrollado proporciona al usuario dos formas principales de interactuar y supervisar el consumo de agua: una interfaz local (física) y una interfaz remota (servidor web).

El objetivo de ambas interfases es facilitar el acceso a la información sobre el sistema, así como permitir el control de diversas funciones de forma cómoda y flexible.

Las dos modalidades de interfaz son:

- Interfaz local

  Basada en una pantalla TFT a color y tres botones físicos, permite al usuario visualizar en tiempo real los datos de consumo de agua, consultar los registros recientes y gestionar funciones básicas como encender/apagar la pantalla, cambiar el modo de comunicación o activar el modo de reposo (bajo consumo). Esta interfaz está pensada para el uso directo y rápido en el lugar donde está instalado el dispositivo.

- Interfaz remota (servidor web)

  El sistema incorpora un servidor web accesible mediante la red WiFi local. A través de un navegador web, el usuario puede conectarse introduciendo "alejandrovelatfg.local" en la url de su navegador, y consultar un menú con diversas opciones: activar o apagar la pantalla, consultar las lecturas de consumo entre dos fechas, visualizar los parámetros de funcionamiento y descargar los registros en formato CSV.

  Esta interfaz remota está pensada para consultas más detalladas y gestión desde cualquier dispositivo conectado a la red (ordenadores, móviles o tablets), sin necesidad de estar físicamente junto al sistema.

5.1. Pantalla TFT

La pantalla utilizada en este proyecto es una pantalla TFT de 2,4 pulgadas con una resolución de 320x240 píxeles. Esta pantalla, controlada mediante el bus SPI, permite mostrar gráficos y texto con buena calidad y colores vivos, proporcionando una experiencia visual clara y agradable.

Funcionalidades mostradas en pantalla:

- Mensajes informativos: durante operaciones específicas (inicio del sistema, entrada en reposo, activación/desactivación de la pantalla), se muestran mensajes temporales para mantener al usuario informado del estado del sistema.

  En la figura 63 se muestra el mensaje que emite la pantalla cuando se está encendiendo el sistema.

Figura 63. Pantalla encendiendo.
Fuente propia.

- Litros acumulados: visualización constante del volumen total de agua consumido desde el inicio del registro o desde la última puesta a cero.
- Historial de los últimos consumos: muestra los siete registros más recientes de consumo (cada lectura realizada cada 4 minutos), permitiendo una rápida revisión del comportamiento del flujo.
- Modo de funcionamiento: en la esquina superior derecha de la pantalla se indica el estado actual del sistema:
  - "SIN COMUNICACIONES" (amarillo)
  - "COMUNICACION WIFI" (cian)
  - "MODO REPOSO" (rojo)

  En la figura 64 se observa como se muestra la pantalla cuando está en modo "comunicación WiFi", en la figura 65 cuando está en modo "sin comunicaciones" y en la figura 66 cuando está en modo reposo. En todos los casos se observa como el usuario puede consultar las últimas siete lecturas realizadas.

Figura 64. Pantalla con comunicación WiFi.
Fuente propia.

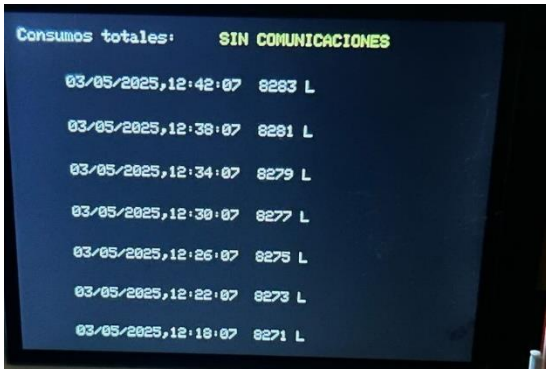


Figura 65. Pantalla sin comunicaciones.
Fuente propia.

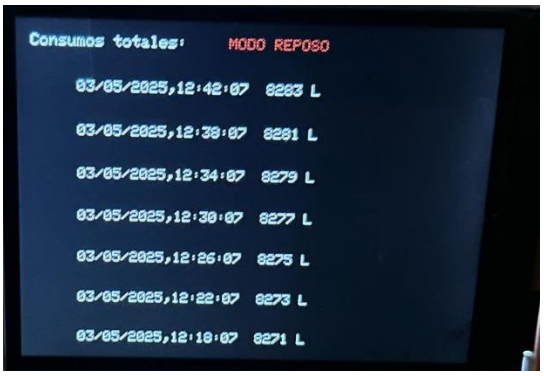


Figura 66. Pantalla en modo reposo.
Fuente propia.

Control de la pantalla:

Para ahorrar energía, la pantalla se puede encender o apagar manualmente usando el botón blanco. Este control se realiza mediante PWM (modulación por ancho de pulso) con la función "ledcWrite(0, *valor*)":

- Valor 255 → Pantalla encendida.
- Valor 0 → Pantalla apagada.

Esto permite al usuario decidir cuándo desea consultar los datos y cuándo desea apagar la pantalla para reducir el consumo eléctrico.

5.2. Servidor web

El sistema integra un servidor web embebido que permite la interacción remota a través de cualquier dispositivo conectado a la misma red WiFi (ordenadores, smartphones, tablets, etc.).

Este servidor web ofrece una interfaz sencilla y accesible mediante un navegador web, sin necesidad de instalar ninguna aplicación adicional.

Estructura y funcionamiento:

Al acceder a la url "alejandrovelatfg.local", el servidor web despliega una página principal (menú), mostrado en la figura 67 con las siguientes opciones:

Figura 67. Menú del webserver.
Fuente propia.

- Activar pantalla: Enciende la pantalla TFT del sistema.
- Apagar pantalla: Apaga la pantalla TFT.
- Apagar comunicaciones: Desactiva temporalmente las funcionalidades de comunicación remota (modo sin comunicaciones).
- Lectura de datos entre fechas: Permite al usuario consultar las lecturas de consumo de agua registradas entre dos fechas concretas, facilitando el análisis histórico del consumo. Esta opción se muestra en la figura 48. En

la figura 49 se muestra un ejemplo introduciendo como fecha de inicio 03/05/2025 y como fecha fin 04/05/2025.

Figura 68. Lectura de datos entre fechas
Fuente propia

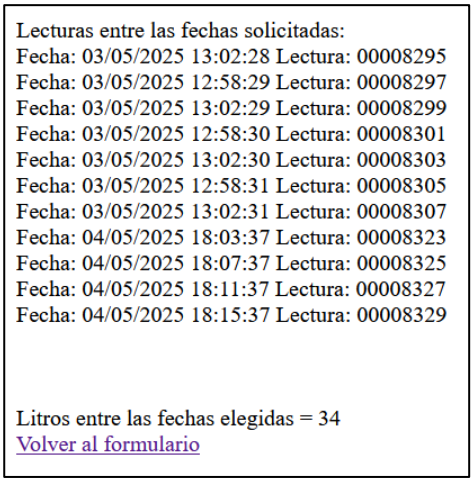


Figura 69. Resultado de lecturas
Fuente propia

- Lectura de parámetros de funcionamiento: Mostrado en la figura 68, proporciona información sobre la configuración actual del sistema, como el intervalo de tiempo entre lecturas. Este apartado queda ampliable a futuras mejoras.

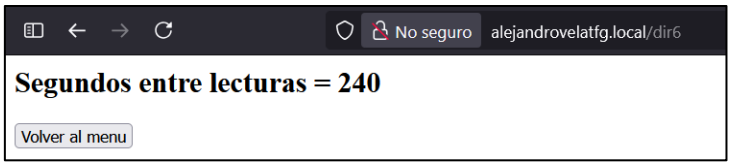


Figura 68. Lectura de parámetros.
Fuente propia.

- Descargar CSV: Descarga el archivo "hora.csv" con el histórico de datos de consumo (litros medidos y "timestamp"), útil para análisis en hojas de cálculo. En la figura 69 se muestra una sección del archivo "hora.csv".

32	00003365,1744110152
33	00003404,1744110182
34	00003444,1744110243
35	00003484,1744110273
36	00003524,1744110345
37	00003564,1744110438
38	00003604,1744110538
39	00003643,1744110568
40	00003682,1744110598
41	00003721,1744110628
42	00003760,1744110658
43	00003799,1744110688
44	00003838,1744110718
45	00003877,1744110748
46	00003916,1744110778
47	00003955,1744110808
48	00003994,1744110838
49	00004033,1744110868

Figura 69. Fragmento del archivo "hora.csv".
Fuente propia.

En caso de que el usuario tenga conocimientos de ofimática, se podría elaborar un gráfico que muestre cómo evoluciona el consumo de agua por hora, véase el ejemplo en la figura 70.

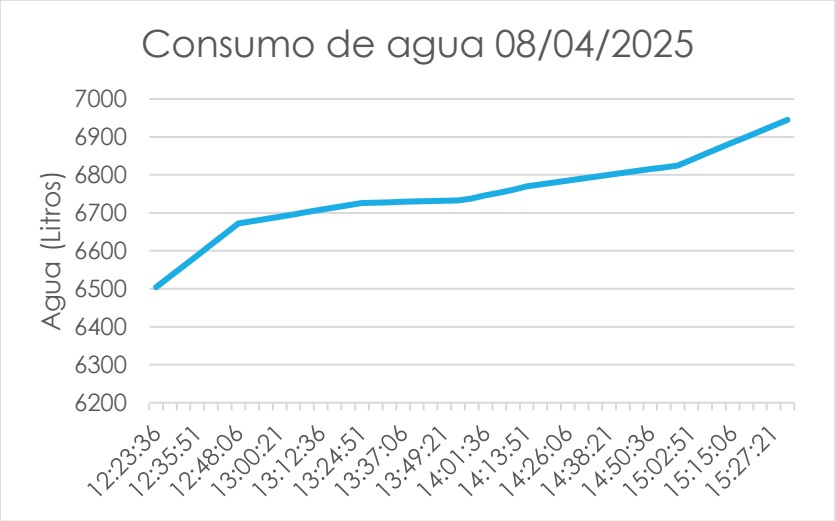


Figura 70. Gráfica de consumo de agua.
Fuente propia.

Implementación técnica:

El servidor web se implementa utilizando las funciones de la librería "Webserver.h" (en este caso simplificada con "server.on()", y responde a diferentes rutas.

Cada una de las rutas invoca una función específica ("handleDirX") que genera la página HTML correspondiente y ejecuta las acciones asociadas.

Capítulo VI

6. Pruebas y validación

En este apartado se documentan las pruebas realizadas para comprobar el correcto funcionamiento del sistema, así como la validación de que cumple con los requisitos definidos.

Las pruebas se han realizado en lo relativo a la circuitería, código del programa e interfaz de usuario.

6.1. Pruebas de circuitería

- **Lectura de caudalímetro**: Se verificó que los pulsos generados por el caudalímetro se cuentan correctamente y se traducen a litros conforme a la fórmula especificada (litros = pulsos / 340).

  El método para comprobar esto es que cuando el potenciómetro está a su valor máximo, el valor de litros debe aumentar en 26,67 L/min. Como la lectura se realiza cada cuatro minutos, se comprobó que la variable de litros aumenta en 106 litros cada cuatro minutos.

- **Pantalla TFT**: Se comprobó que la pantalla se enciende y apaga correctamente, y que muestra los últimos consumos de forma clara.

- **Botones físicos**: Se validó que las interrupciones asociadas a los botones blanco, verde y azul responden de forma adecuada:

  - Botón blanco alterna encendido/apagado de pantalla.
  - Botón verde cambia el modo de comunicación.
  - Botón azul activa/desactiva el modo reposo.

6.2. Pruebas del código del programa

- Servidor web: Se probó el acceso al servidor web desde diferentes dispositivos (ordenador, móvil), validando la carga correcta de las páginas y la ejecución de cada acción.

- Lectura de datos entre fechas: Se comprobó que las fechas introducidas en el formulario filtran correctamente las lecturas almacenadas.

- Guardado en tarjeta SD: Se validó que el archivo hora.csv se actualiza automáticamente cada 4 minutos con los datos de litros y timestamp, y que el archivo ultimalectura.txt guarda el último valor registrado.

6.3. Pruebas de conectividad

- Conexión WiFi: Se realizaron pruebas de conexión a distintas redes WiFi para asegurar la estabilidad y la correcta obtención de la dirección IP.

- mDNS: Se validó que el dispositivo es accesible mediante el nombre de dominio local alejandrovelatfg.local.

6.4. Pruebas de consumo y reposo

- Modo reposo (light sleep): Se comprobó que al activar el modo reposo mediante el botón azul, el consumo de energía disminuye y el sistema despierta correctamente tras el tiempo programado o mediante la interrupción.

6.5. Validación final

El sistema se consideró validado cuando se cumplió que:

- La lectura de consumo es precisa y fiable.

- Los datos se guardan correctamente en la SD y son accesibles por el usuario.

- La interfaz (pantalla TFT y servidor web) permite consultar y gestionar el sistema de forma intuitiva.

- Las interrupciones y modos de energía funcionan sin fallos.

Capítulo VII

7. Conclusiones

El desarrollo de este sistema de supervisión de consumo de agua ha permitido integrar con éxito una solución que combina monitorización, almacenamiento y consulta de datos de forma accesible y fiable.

Se ha conseguido implementar un sistema capaz de:

- Medir el caudal de agua de manera precisa, registrando los datos en una tarjeta SD para su posterior consulta.
- Visualizar el consumo en tiempo real a través de una pantalla TFT, proporcionando información clara al usuario.
- Facilitar el acceso remoto mediante un servidor web, desde el cual se pueden consultar lecturas, descargar datos históricos y configurar parámetros.

El sistema ha demostrado un funcionamiento estable durante las pruebas realizadas, respondiendo correctamente a las acciones del usuario y registrando las mediciones de forma consistente. Las interfases (pantalla y web) ofrecen una experiencia intuitiva, y la posibilidad de activar un modo de bajo consumo aumenta su eficiencia energética.

A pesar de los buenos resultados, se han identificado posibles mejoras para futuras versiones, como la inclusión de mecanismos de autenticación para el servidor web o aumentar la cantidad de parámetros que el usuario puede consultar, así como modificarlos.

En conjunto, el proyecto ha cumplido los objetivos propuestos y proporciona una herramienta práctica para la monitorización de consumos de agua, adaptable a diferentes entornos y necesidades.

Capítulo VIII

8. Opciones de futuro

A pesar de que el sistema desarrollado cumple con los objetivos planteados, existen diversas líneas de mejora y evolución que podrían aplicarse en versiones futuras. Estas opciones están orientadas tanto a optimizar el rendimiento energético como a mejorar la experiencia de usuario y la durabilidad del dispositivo.

8.1. Uso de una batería

La posibilidad de alimentar el sistema mediante una batería recargable permitiría una mayor portabilidad e independencia de la red eléctrica, facilitando su instalación en lugares remotos o de difícil acceso. Esta opción implicaría también un estudio más profundo del consumo energético en diferentes modos de operación para maximizar la autonomía.

8.2. Mejoras gráficas en la interfaz

La interfaz actual podría enriquecerse con una visualización más intuitiva y atractiva, incluyendo gráficos interactivos, iconografía, y esquemas en tiempo real del caudal o consumo acumulado. Estas mejoras aumentarían la usabilidad y facilitarían la interpretación de los datos por parte del usuario.

8.3. Aumento de la vida útil del dispositivo

Optimizar el diseño para reducir el desgaste de sensores o patillas de conexión, permitiría extender la vida útil del dispositivo. También podría considerarse la implementación de algoritmos de reposo más avanzados y encapsulados más robustos para ambientes exteriores.

Capítulo IX

9. Bibliografía

[1] https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf - Última consulta 15/05/2025

[2] https://www.luisllamas.es/esp32-consumo-energia/ - Última consulta 19/04/2025

[3] https://www.luisllamas.es/esp32-sleep-modes/ - Última consulta 19/04/2025

[4] https://digizone.com.ve/wp-content/uploads/2021/06/yf-b10.pdf - Última consulta 18/05/2025

[5] https://github.com/pbolduc/esp32-benchmarks - Última consulta 16/05/2025

[6] https://github.com/martinberlin/ESP32-benchmark - Última consulta 16/05/2025

[7] https://www.ntppool.org/zone/es - Última consulta 16/04/2025

[8] https://randomnerdtutorials.com/esp32-date-time-ntp-client-server-arduino/ - Última consulta 16/04/2025

[9] https://www.itron.com – Última consulta 18/05/2025

[10] https://www.kamstrup.com – Última consulta 18/05/2025

[11] https://www.xylem.com/en-us/brands/sensus/ - Última consulta 18/05/2025

[12] https://www.ti.com/lit/ds/symlink/tlc555.pdf - Última consulta 20/05/2025

[13] https://www.ti.com/lit/ds/symlink/sn74hc590a.pdf - Última consulta 20/05/2025

[14] https://www.ti.com/lit/ds/symlink/sn74hc165.pdf - Última consulta 20/05/2025

[15] http://www.lcdwiki.com/res/MSP2402/QD241801_specification_v1.1.pdf - Última consulta 17/05/2025

Capítulo X

Anexo A
Cálculos

Anexo A. Cálculos.

Frecuencia de lecturas

Se parte del supuesto de que el usuario debe poder consultar hasta un mes de lecturas en datos a través de la interfaz del servidor web.

$$1\ mes = 30\frac{días}{mes} \cdot 24\frac{horas}{día} \cdot 60\frac{minutos}{hora} = \ 43.200\ minutos$$

El límite es la RAM, ya que lo que se muestra al usuario a través del servidor web se almacena en la memoria RAM antes de que se envíe. Se requiere que quede libre un 10% de RAM como seguridad para que no colapse el programa.

Al compilar el programa, la herramienta con la que se realiza muestra la cantidad de bytes que se ocupan en RAM y en Flash, esto es mostrado en la figura 71.

```
RAM:   [==        ]  15.3% (used 50104 bytes from 327680 bytes)
Flash: [===       ]  25.1% (used 952425 bytes from 3801088 bytes)
```

Figura 71. Memoria RAM y Flash usada.
Fuente propia.

Por lo que la cantidad de RAM disponible para el mensaje es:

$$RAM_{DISPONIBLE} = RAM_{TOTAL} - RAM_{10\%} - RAM_{USADA}$$

$$327.680\ bytes - 327.680\ bytes \cdot 0{,}10 - 50.104\ bytes = \ 244.808\ bytes$$

Los mensajes ocupan 21 bytes ya que almacenan una cantidad de litros como un número de 8 dígitos, una coma y el valor en formato "epoch" como un número de 10 dígitos, y un salto de línea. Es decir, que tendría el siguiente formato:

XXXXXXXX,YYYYYYYYYY/n

Donde:

- XXXXXXXX : Representa el valor en litros de la lectura. Lo forman 8 caracteres.
- , : Separa las dos partes de la lectura (valor y fecha). Lo forma 1 carácter.
- YYYYYYYYYY : Representa el valor en formato "epoch" de la fecha de la lectura. Lo forman 10 caracteres.
- /n : En el mensaje no se ve por que es invisible, pero es la nomenclatura del salto de línea. Lo forman 2 caracteres (CR y LF).

Cada carácter ocupa 1 byte por lo que el tamaño de cada lectura se calcula de la siguiente manera:

$$Tamaño\ de\ cada\ lectura = 8\ bytes\ + 1\ byte\ +\ 10\ bytes + 2\ bytes = 21\ bytes$$

Ahora se calcula la cantidad de lecturas que se pueden enviar sin desbordar la RAM:

$$Cantidad\ máxima\ de\ lecturas = \frac{244.808\ bytes}{21\ bytes/lectura} =\ 11.657{,}52\ lecturas$$

Que se deben redondear a 11.657 lecturas ya que el objetivo es no desbordar la memoria RAM.

Ahora se calcula cual debe ser la frecuencia de lectura para que en 30 días haya 11.657 lecturas.

$$Frencuencia\ de\ lectura = \frac{11.657{,}52\ mensajes}{43.200\ minutos} =\ 0{,}26983796\frac{lecturas}{minuto}$$

O lo que es lo mismo:

$$3{,}7059\frac{minutos}{lecturas}$$

Por lo que para redondear a un número que sea lógico por el usuario, y además tener libre más memoria RAM, se elige una frecuencia de lectura de 4 minutos por lectura.

Vida útil teórica

Dado que los fabricantes de los circuitos integrados usados en el sistema no proporcionan su vida útil, la vida útil teórica depende de que la cantidad de litros totales leídos no desborde.

Como se ha visto anteriormente, cada mensaje destina 8 bytes al valor de litros totales leídos.

Por lo que la cantidad máxima de litros totales que se puede leer es 99.999.999 L.

Usando de ejemplo el YF-B10 con un caudal máximo registrable de 25 L/min.

$$Minutos\ máximos\ de\ lecturas = \frac{99999999\ L}{25\ L/min} = 3.999.999{,}96\ minutos$$

Realizando la conversión:

$$3.999.999{,}96\ \text{min} \cdot \frac{1\ hora}{60\ min} \cdot \frac{1\ día}{24\ horas} \cdot \frac{1\ año}{365\ días} = 7{,}61\ años$$

El cálculo realizado estima una vida útil teórica de 7,61 años, partiendo además del supuesto de que el caudal esté leyendo constantemente durante 24 h/día a 25 L/min la cual es su frecuencia máxima.

Este valor se considera adecuado y razonable dentro del contexto de dispositivos electrónicos. Aunque en la práctica muchos equipos electrónicos no se utilizan durante tanto tiempo de forma continua, es habitual que las estimaciones de vida útil reflejen el potencial máximo del dispositivo si se mantiene en condiciones óptimas y sin obsolescencia tecnológica prematura.

Desborde del almacenamiento SD

Para el prototipo se ha usado una tarjeta SD de 4 GB para guardar el archivo "hora.csv", que incrementa su tamaño en 21 bytes con cada lectura que se realiza.

Por lo que la cantidad máxima de lecturas posibles es de:

$$\frac{4.294.967.296\ bytes}{21\ bytes} = 204.522.252\ lecturas$$

Como cada lectura se realiza cada 4 minutos:

$$204.522.252\ lecturas \cdot \frac{4\ minutos}{lectura} = \ 818.089.008\ minutos$$

Que son 1.556 años.

Con lo que se demuestra, en contraste con el apartado anterior, que la vida útil del sistema depende del desborde de la variable de cuenta de litros. Ya que esta variable desborda antes de que se llene el tamaño total de la tarjeta SD.

Máximo de litros por lectura

La cantidad máxima de litros legible depende de los pulsos y del tamaño del contador.

Al ser un contador de 16 bits, el valor máximo de pulsos por lectura es 65.535 (2^16 – 1).

Como cada 340 pulsos equivalen a un litro:

$$\frac{65.535\ pulsos\ máximos}{340\frac{pulsos}{litro}} = 192{,}75\ litros$$

Caudal máximo

El caudal máximo depende del máximo de litros por lectura y la frecuencia de lectura:

$$\frac{192{,}75\ litros}{4\ minutos} = 48{,}18\ litros/minuto$$

Este número desborda el caudal máximo del sensor de flujo que se pretende simular (YF-B10, caudal máximo 25 L/min) por lo que se cumple con las expectativas esperadas.

Tasa de error

La tasa de error se basa en que pueda suceder que se emita un pulso mientras se realiza la lectura de los contadores y este pulso se pierda ya que se inhibe el reloj del contador.

Para esto hay que tener en cuenta cuanto tiempo se inhibe el reloj del contador en la función "readCounters()" representada en la figura 72.

La función "delay()" le da al microcontrolador un tiempo de espera del valor introducido dentro del paréntesis en milisegundos.

El microcontrolador ESP32 tarda un tiempo de entre 0,11µs y 0,19µs en ejecutar una instrucción "digitalWrite" (líneas 360, 361, 363 y 364).

El único momento en el que se pueden perder pulsos es desde que se inhibe el reloj de los contadores (Línea 360) hasta que se reanuda el reloj de los contadores (Línea 364). Por lo que el tiempo en el que se pueden perder pulsos es:

$$Tiempo\ que\ se\ pueden\ perder\ pulsos = 0{,}19\mu s \cdot 4\ digitalWrite + 1\ ms \cdot 4\ delay \quad (1)$$
$$Tiempo\ que\ se\ pueden\ perder\ pulsos = 4{,}000074\ ms$$

```
359  uint16_t readCounters() {
360    digitalWrite(CLK_EN_PIN, HIGH); // 1. Inhibe el reloj de los contadores
361    delay(1);
362    digitalWrite(LOAD_PIN, LOW);    // 2. Carga los datos en el 74HC165 (LOAD a LOW)
363    delay(1);
364    digitalWrite(LOAD_PIN, HIGH);   // Vuelve a desactivar la carga
365    delay(1);
366    resetCounters();                // 3. Reinicia los contadores
367    delay(1);
368    digitalWrite(CLK_EN_PIN, LOW);  // 4. Reanuda el reloj de los contadores
369
370
371    uint16_t data = 0;
372    for (int i = 0; i < 16; i++) {  // 5. Lee los 16 bits en serie (MSB first)
373      data |= (digitalRead(DATA_PIN) << (15 - i));
374      digitalWrite(CLK_PIN, HIGH);
375      delay(1);
376      digitalWrite(CLK_PIN, LOW);
377    }
378    return data;                    // 6. Retorno de la información
379  }
```

Figura 72. Función "resetCounters()".
Fuente propia.

Ahora es necesario calcular cuantos pulsos se pueden llegar a perder en 4,000074 ms.

La frecuencia máxima supuesta para los pulsos es de 152,04 Hz que equivalen a 0,00657 segundos, o lo que es lo mismo, 6,57 milisegundos.

Como el periodo en el que se puede perder un pulso es inferior al periodo mínimo (frecuencia máxima) de generación de pulsos, la única posibilidad de

perder pulsos es de que ese pulso entre dentro de los 4,000074 milisegundos en los que se inhibe el reloj.

Suponiendo que este evento ocurra cada vez que se realiza una lectura, se pierde un pulso cada 4 minutos.

Ahora es necesario calcular a cuantos litros equivale un pulso. Conociendo la relación la cual dice que 340 pulsos equivalen a 1 litro:

$$1\ pulso \cdot \frac{1\ litro}{340\ pulsos} = \ 0{,}00294\ litros$$

Es decir, se pueden llegar a perder 0,00294 litros registrados cada 4 minutos. Como el caudal normalmente se expresa en litros/minuto, se realiza la conversión.

$$Caudal\ perdible = \frac{0{,}00294\ litros}{4\ minutos} = \ 0{,}000735\ litros/minuto$$

La mayor tasa de error se produce en el peor de los casos, es decir, cuando se pierde un pulso durante el tiempo en que el reloj de los contadores está inhibido, como ya se ha planteado anteriormente.

Aunque esta pérdida se ha supuesto a caudales altos (para maximizar los pulsos perdibles), la duración de la inhibición es tan breve que como mucho se pierde un único pulso, incluso si el sensor estuviera midiendo su caudal máximo.

Por lo tanto, para maximizar el error relativo, se realiza el cálculo en condiciones de caudal mínimo permitido por el sensor, ya que perder un pulso en ese caso representa una proporción mayor del volumen total.

Dado que el caudal mínimo es de 1 litro/minuto, y el caudal perdible es de 0,000735 litros/minuto, la tasa de error se calcula así:

$$Tasa\ de\ error = \frac{Caudal\ perdible}{Caudal\ mínimo} \cdot 100 = \frac{0{,}000735\ l/min}{1\ l/min} \cdot 100 = \ 0{,}0735\%$$

Tasa de error que cumple con el *Real Decreto 244/2016 y la Orden ICT/155/2020, Anexo VIII* donde consta que el error máximo permitido para un dispositivo contador de agua debe ser del 2% para agua con una temperatura inferior a 30ºC y del 3% para agua con una temperatura superior a 30ºC.

Cálculo valores TLC555 aestable

La ecuación de la frecuencia de un TLC555 en modo aestable viene dada por la siguiente ecuación:

$$f = \frac{1{,}44}{(R1 + 2 \cdot R2) \cdot C}$$

Los valores en el circuito implementado son los siguientes:

- R1 = 4,7 kΩ
- R2 = 12 kΩ (fija) + 1 MΩ potenciómetro (variable).
- C = 0,33µF

Por lo que el valor de frecuencia máximo viene dado cuando el potenciómetro da su valor mínimo, y el valor de frecuencia mínimo cuando el potenciómetro da su valor máximo.

$$f_{máx} = \frac{1{,}44}{(4{,}7\ k + 2 \cdot 12\ k) \cdot 0{,}33\ \mu} = 152{,}04\ Hz$$

$$f_{mín} = \frac{1{,}44}{(4{,}7\ k + 2 \cdot (1\ M + 12\ k) \cdot 0{,}33\ \mu} = 2{,}15\ Hz$$

$$f_{máx} = \frac{1{,}44}{(12\ k + 2 \cdot 4{,}7\ k) \cdot 0{,}33\ \mu} = 152{,}04\ Hz$$

Cálculo del consumo del contador 74HC590

En la hoja de características del fabricante se indica que la corriente de consumo se calcula con la siguiente ecuación:

$$I_{CC_{(OPR)}} = C_{PD} \cdot V_{CC} \cdot f_{IN} + I_{CC}$$

Donde:

- $I_{CC(OPR)}$: Es la corriente de consumo. Es el valor a calcular
- C_{PD} : Es la capacitancia interna equivalente del circuito integrado. El fabricante proporciona un valor típico de 40 pF.
- VCC : Es la tensión de alimentación. En el circuito este valor es de 3,3 V.
- f_{IN} : Es la frecuencia de entrada. Para calcular el consumo máximo, se usa la frecuencia máxima la cual es de 158,6 Hz.
- I_{CC} : Es el consumo típico proporcionado por el fabricante cuando no está en funcionamiento. El fabricante proporciona un valor típico de 4µA.

Por lo que sustituyendo con los datos del circuito y los aportados por el fabricante se obtiene:

$$I_{CC_{(OPR)}} = 40\ pF \cdot 3{,}3\ V \cdot 158{,}6\ Hz + 4\ \mu A = 4{,}021\ \mu A$$

Cálculo del consumo del registro de desplazamiento 74HC165

En la hoja de características del fabricante se indica que la potencia disipada se calcula con la siguiente ecuación:

$$P_D = C_{PD} \cdot V_{CC}^2 \cdot fi + \Sigma(C_L \cdot V_{CC}^2 \cdot f_o)$$

Donde:

- P_D : Es la potencia disipada en µW. Es el valor que calcular.
- C_{PD} : Es la capacitancia interna equivalente del circuito integrado, debe usarse en pF. El fabricante proporciona un valor típico de 3,5 pF.
- V_{CC} : Es la tensión de alimentación. En el circuito este valor es de 3,3 V.
- f_i : Es la frecuencia de entrada en MHz. Para calcular el consumo máximo, se usa la frecuencia máxima la cual es 158,6 Hz.
- C_L : Es la capacitancia de carga externa debe usarse en pF. El fabricante proporciona un valor típico de 3,5 pF.
- f_o : Es la frecuencia de conmutación de las salidas en MHz. Las salidas conmutan 16 veces cada cuatro minutos. Se calcula el valor de fo:

$$f_o = 16 \cdot f_{conmutación} = 16 \cdot \frac{1}{240\ segundos} = 0{,}0667\ Hz$$

Por lo que sustituyendo con los datos del circuito y los aportados por el fabricante se obtiene:

$$P_D = 3{,}5\ pF \cdot (3{,}3\ V)^2 \cdot 1{,}586 \cdot 10^{-4}\ MHz + \ 3{,}5\ pF \cdot (3{,}3\ V)^2 \cdot 6{,}67 \cdot 10^{-8}\ MHz$$

$$P_D = 6{,}047 \cdot 10^{-15}\ W = 2{,}28 \cdot 10^{-3}\ \mu W$$

Se calcula la intensidad, para posteriormente poder calcular el consumo total:

$$I = \frac{P}{V}$$

Donde:

- I : Intensidad
- P : Potencia
- V : Tensión

$$I = \frac{6{,}0475 \cdot 10^{-3}\ \mu W}{3{,}3\ V} = 6{,}9 \cdot 10^{-4}\ \mu A = 0{,}69 nA$$

Cálculo del consumo total del sistema en modo reposo

El microcontrolador ESP32 consume 0,8 mA cuando se encuentra en el modo "light sleep" (dato proporcionado por el fabricante). El consumo total en modo reposo es la suma del consumo del microcontrolador y el consumo de los circuitos integrados externos que siguen en funcionamiento. Por lo que teniendo en cuenta los consumos calculados anteriormente de los circuitos integrados

$$Corriente_{REPOSO} = \mu C + 74HC590 \cdot 2 + 74HC165 \cdot 2$$

$$Corriente_{REPOSO} = 0{,}8\ mA + 4{,}021\ \mu A \cdot 2 + 0{,}69\ nA \cdot 2$$

$$Corriente_{REPOSO} = 0{,}808\ mA$$

Cálculo del consumo total del sistema en modo activo

El microcontrolador ESP32 consume torno a 100 mA cuando recibe señal WiFi, dependiendo de la carga de red (dato proporcionado por el fabricante) Encender la pantalla supone un aumento del consumo de hasta 120 mA (dato obtenido del fabricante de la pantalla).

Esto supone un consumo con pantalla encendida de:

$$Consumo\ del\ modo\ activo\ con\ pantalla\ encendida = 100\ mA + 120\ mA$$
$$Consumo\ del\ modo\ activo\ con\ pantalla\ encendida = \ 220\ mA$$

Anexo B
Código del programa

src\main.cpp

```cpp
// ##############################
// ##     TFG ALEJANDRO VELA   ##
// ##############################

//  *** INCLUSIÓN DE LIBRERÍAS ***
#include <WiFiClient.h>   // Librería necesaria para la conexión WiFi
#include <WebServer.h>    // Librería necesaria para crear una página web
#include <ESPmDNS.h>      // Librería necesaria para mDNS (personalizar URL de la página web)
#include <TFT_eSPI.h>     // Librería necesaria para la pantalla TFT SPI
#include <SD.h>           // Librería necesaria para la lectura/escritura de la tarjeta SD
#include <time.h>         // Librería necesaria para actualizar y controlar la fecha y hora
#include <TimeLib.h>      // Librería necesaria para manejar distintos formatos de tiempo

// *** DECLARACIÓN DE VARIABLES ***
#define LOAD_PIN 26   // Carga paralela 74HC165        (/PL)
#define CLK_PIN 27    // Reloj 74HC165                 (CLK)
#define CLK_EN_PIN 14 // Habilita el reloj del 74HC590 (CLK_EN)
#define DATA_PIN 25   // Salida del 74HC165 MSB        (Data)
#define RESET_PIN 33  // Reset de los 74HC590          (/RST)

#define PIN_BLANCO 34 // Pin del botón asignado a la pantalla (BP)
#define PIN_VERDE 35  // Pin del botón asignado al modo       (BM)
#define PIN_AZUL 32   // Pin del botón asignado al reposo     (BR)

#define PIN_LED 13    // Pin que ajusta el brillo de la pantalla

// Variables de la pantalla
TFT_eSPI TFT;   // Objeto con el que se llama a las funciones de la pantalla

// Variables de conexión WiFi
const char* ssid = "ONOWANLU";    // SSID de WiFi al que se conecta el ESP32
const char* pass = "4111936231";  // Contraseña del WiFi al que se conecta el ESP32

// Variables de fecha y hora
const char* ntpServer = "0.es.pool.ntp.org";  // Servidor NTP de donde se obtiene la fecha y hora
char timeStringBuff[50];
struct tm timeinfo;                // Objeto con el que se llama a las funciones de fecha y hora
String tiempoEnEpoch = "";

// Variables del temporizador que registra el consumo de agua
hw_timer_t *timer = NULL;   // Declarar el temporizador
volatile bool timerFlag = false;    // Bandera para saber cuándo se activa la interrupción
int configuracionsegundoslectura = 240;   // 4 minutos entre lecturas

```

```
45 // Variables de botones
46 bool BotonVerde = false;    // Variable booleana pulsación botón verde
47 bool BotonBlanco = false;   // Variable booleana pulsación botón blanco
48 bool BotonAzul_isAsleep = false;    // Variable booleana pulsación botón Azul
49 int ModoComunicacion = 0;   // Variable para detectar si está o no comunicación WiFi
50 bool is_pantallaEncendida = true;   // Variable para detectar si la pantalla está o no
   encendida
51
52 // Variables para almacenar los consumos
53 int pulseHistory[7] = {0};    // Array para almacenar los últimos consumos (Litros)
54 String pulseTimes[7] = {""};  // Array para almacenar los últimos consumos (Fecha y
   hora)
55 int litrosCount = 0;          // Se crea una variable para llevar la cuenta de litros
56 char litrosTotales[9];        // Se crea una variable para que el valor siempre sea de
   8 dígitos
57
58 // Variables WebServer
59 WebServer server(80); // Crea una instancia del servidor en el puerto 80
60
61 // DECLARACIÓN FUNCIONES
62 void mostrarUltimosConsumos() {
63     TFT.fillScreen(TFT_BLACK);
64     TFT.setTextColor(TFT_WHITE);
65     TFT.setTextSize(2);
66     TFT.setCursor(10, 10);
67     TFT.setTextSize(1);
68     TFT.println("Consumos totales:");
69     for (int i = 0; i < 7; i++) {
70         TFT.setCursor(40, 40 + i * 30);
71         TFT.print(pulseTimes[i]);
72         TFT.printf("  %d L", pulseHistory[i]);
73     }
74 }
75
76 void mostrarArribaDerecha(String textoquemostrar, int color){
77   TFT.setTextColor(color, TFT_BLACK);
78   TFT.setCursor(140, 10);
79   TFT.setTextSize(1);
80   TFT.println(textoquemostrar);
81 }
82
83 void RotarActualizarArray(int Array[7], int Actualizador){
84   for(int i = 5; i >= 0; i--){
85     Array[i+1] = Array[i];
86   }
87   Array[0] = Actualizador;
88 }
89
90 void RotarActualizarTiempo(String Tiempo[7]){
91   if (getLocalTime(&timeinfo)) {
   // Obtiene fecha y hora actuales
92     strftime(timeStringBuff, sizeof(timeStringBuff), "%d/%m/%Y,%H:%M:%S", &timeinfo);
   // Pasa fecha y hora a String: "DD/MM/YYYY, HH:MM:SS"
```

```
93    for(int i = 5; i >= 0; i--){
94      Tiempo[i+1] = Tiempo[i];
95    }
96    Tiempo[0] = String(timeStringBuff);
97    }
98  }
99
100
101 // Función para convertir fecha y hora a timestamp
102 int convertirFechaATimestamp(String fecha, String hora) {
103   int dia, mes, ano, horaInt, minuto, segundo;
104   sscanf(fecha.c_str(), "%d/%d/%d", &dia, &mes, &ano);
105   sscanf(hora.c_str(), "%d:%d:%d", &horaInt, &minuto, &segundo);
106
107   // Crear un objeto tmElements_t para almacenar la fecha y hora
108   tmElements_t tm;
109   tm.Day = dia;
110   tm.Month = mes;
111   tm.Year = ano - 1970;  // Year es el número de años desde 1970
112   tm.Hour = horaInt;
113   tm.Minute = minuto;
114   tm.Second = segundo;
115
116   // Usamos makeTime para convertir a timestamp
117   return makeTime(tm);
118 }
119
120 // Función que convierte un String timestamp a fecha legible
121 String convertirTimestamp(String timestamp) {
122   time_t tiempo = (time_t)timestamp.toInt();
123   struct tm *timeinfo = localtime(&tiempo);
124
125   char buffer[30];
126   strftime(buffer, sizeof(buffer), "%d/%m/%Y %H:%M:%S", timeinfo);
127
128   return String(buffer);
129 }
130
131 // Función para leer el archivo CSV y obtener las lecturas entre dos fechas
132 String obtenerLecturasEntreFechas(String fechaInicio, String fechaFin) {
133
134   bool is_rangoCorrecto = false;          // Booleana para bucle explicado más abajo
135   bool is_primeraLecturaFiltrada = false; // Booleana para condicional explicado más
    abajo
136   String resultado = "Lecturas entre las fechas solicitadas:\n";  // String donde se
    guardará lo que leerá el usuario
137
138   // Se convierte la fecha de un formato legible a un formato en el que es más fácil
    comparar (epoch)
139   int timestampInicio = convertirFechaATimestamp(fechaInicio.substring(0, 10),
    "00:00:00");
140   int timestampFin = convertirFechaATimestamp(fechaFin.substring(0, 10), "23:59:59");
141
```

```
142   if(timestampInicio < 1735689600){
143     resultado = "ERROR: Fecha de inicio anterior a 01/01/2025 \n       O formato
invalido";
144     return resultado;
145   }
146   else if(timestampFin < 1735689600){
147     resultado = "ERROR: Fecha de inicio anterior a 01/01/2025 \n       O formato
invalido";
148     return resultado;
149   }
150   else if(timestampInicio > timestampFin){
151     resultado = "ERROR: Fecha de inicio mayor que fecha de fin";
152     return resultado;
153   }
154   else if(timestampFin - timestampInicio > 2851200){
155     resultado = "ERROR: Diferencia de fechas superior al limite";
156     return resultado;
157   }
158 
159   int primeraLecturaFiltrada; // Variable para registrar la primera lectura del rango
indicado
160   int ultimaLecturaFiltrada;  // Variable para registrar la ultima lectura del rango
indicado
161 
162   // Variables para la búsqueda binaria
163   int leftByte = 0;   // La izquierda siempre es 0 ya que lo único que se guarda en ese
archivo son las fechas y consumos
164   int rightByte;      // La derecha depende del tamaño del archivo
165   int byteBusqueda;   // Variable que se usa para indicar que byte del archivo se va a
leer
166 
167   String linea = "";
168   String valor = "";
169   String epoch = "";
170 
171   File archivo = SD.open("/hora.csv");  // Abre el archivo hora.csv
172   if (!archivo) {
173     return "No se pudo abrir el archivo.";
174   }
175 
176   rightByte = archivo.size();
177 
178   while(!is_rangoCorrecto){
179     if(byteBusqueda != leftByte + (rightByte - leftByte)/2){  // Si busca 2 veces
seguidas en el mismo byte, es que no hay rango correcto.
180     byteBusqueda = leftByte + (rightByte - leftByte)/2;       // Siempre busca el valor
medio entre izquierda y derecha
181     }
182     else{
183       resultado = "No hay lecturas entre las fechas especificadas";
184       return resultado;
185     }
186 
```

```
187     archivo.seek(byteBusqueda);     // Así, aunque se pueda perder una linea,
aseguramos que está leyendo la linea completa.
188     archivo.readStringUntil('\n');  // No importa que se pierda una linea ya que
posteriormente se busca el primer valor.
189
190     linea = archivo.readStringUntil('\n');  // Lee hasta el salto de línea
191     epoch = linea.substring(9,19);          // Se lee el tiempo en epoch para poder
comparar con las fechas ingresadas por el usuario.
192
193     if (timestampInicio > epoch.toInt()) {  // Si la fecha de inicio es mayor que la
del valor medio
194                                             // Los valores buscados quedan por encima
del valor medio
195       leftByte = byteBusqueda;              // Ahora la izquierda es el valor medio
196     }
197     else if (epoch.toInt() > timestampFin){ // Si el valor medio es mayor que la fecha
fin
198                                             // Los valores buscados quedan por debajo
del valor medio
199       rightByte = byteBusqueda;             // Ahora la derecha es el valor medio
200     }
201     else if (timestampInicio < epoch.toInt() && timestampFin > epoch.toInt()){  // Si
el valor medio es mayor que el de inicio
202                                                                                 // y
menor que el de fin
203       is_rangoCorrecto = true;                                                  //
Hemos entrado en el rango
204     }
205   }
206
207   while(timestampInicio < epoch.toInt() && byteBusqueda > 0){   // Mientras la fecha de
inicio sea menor que la que se está leyendo
208     // y byteBusqueda no sea negativo.
209
210     archivo.seek(byteBusqueda);
211     archivo.readStringUntil('\n');
212
213     linea = archivo.readStringUntil('\n');  // Mismo razonamiento que antes, se pueden
saltar lineas pero no importa, después se leen.
214     epoch = linea.substring(9,19);          // Se lee al final para hacer la
comprobación del "while" al principio.
215
216     byteBusqueda = byteBusqueda - 21;       // Cada mensaje ocupa 21 bytes, por lo que
vamos leyendo el mensaje anterior hasta encontrar
217                                             // el primer mensaje.
218   }
219
220   if(byteBusqueda < 0){
221     byteBusqueda = 0; // Si ha llegado al inicio del archivo, es igual a cero para que
no sea negativo.
222   }
223
224   valor = linea.substring(0,8);
225
```

```
226   while(timestampFin > epoch.toInt() && archivo.available()){ // Mientras la fecha de
fin sea mayor que la que se está leyendo
227                                                               // y queden lineas por
leer
228   // (puede pasar que el usuario ingrese una fecha mayor que la ultima registrada)
229
230     String fecha = convertirTimestamp(epoch); // Se pasa de los segundos en epoch a
fechas legibles
231
232     resultado += "Fecha: " + fecha + " Lectura: " + valor + "\n"; // Se añade fecha y
valor a lo que ve el usuario.
233
234     if(!is_primeraLecturaFiltrada){             // La primera vez que se lee una fecha
correcta, es decir, la primera del rango
235       primeraLecturaFiltrada = valor.toInt();   // Se guarda como primera lectura para
poder hacer la resta entre primera y ultima
236       is_primeraLecturaFiltrada = true;
237     }
238     ultimaLecturaFiltrada  = valor.toInt();     // Cada vez que se lee, se guarda como
ultima lectura.
239
240     linea = archivo.readStringUntil('\n');  // Lee linea a linea
241     valor = linea.substring(0,8);           // Se leen los valores al final, para que
no entren valores fuera de rango
242     epoch = linea.substring(9,19);
243   }
244
245   archivo.close();  // Cerramos el archivo
246   resultado += "\n \n \n";
247   resultado += "Litros entre las fechas elegidas = ";
248   resultado += ultimaLecturaFiltrada - primeraLecturaFiltrada;
249   return resultado;
250 }
251
252 // Función para convertir \n a <br> en HTML
253 String convertirSaltosDeLineaHTML(String texto) {
254   String resultado = "";
255   for (int i = 0; i < texto.length(); i++) {
256     if (texto[i] == '\n') {
257       resultado += "<br>";
258     } else {
259       resultado += texto[i];
260     }
261   }
262   return resultado;
263 }
264
265 // Funciones para manejar las rutas
266 void handleRoot() {
267   server.send(200, "text/html",
268     "<h1>Menu del supervisor de consumo de agua</h1><ul><li><a href='/dir1'>Activar
pantalla</a></li><li><a href='/dir2'>Apagar pantalla</a></li><li><a href='/dir3'>Apagar
comunicaciones</a></li><li><a href='/dir4'>Lectura de datos entre fechas</a></li><li><a
href='/dir6'>Lectura de parametros de funcionamiento</a></li><li><a href='/
```

```
dir7'>Descargar CSV</a></li></ul>");
}

void handleDir1() { // Activar pantalla
  String html = "<html><body>";
  html += "<h1>Encendiendo la pantalla...</h1>";
  html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
  html += "</body></html>";
  server.send(200, "text/html", html);

  ledcWrite(0, 255);
  is_pantallaEncendida = true;
  Serial.println("Pantalla encendida");
 }

void handleDir2() {
  String html = "<html><body>";
  html += "<h1>Apagando la pantalla...</h1>";
  html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
  html += "</body></html>";
  server.send(200, "text/html", html);

  ledcWrite(0, 0);
  is_pantallaEncendida = false;
  Serial.println("Pantalla apagada");
}

void handleDir3() {
  String html = "<html><body>";
  html += "<h1>Apagando las comunicaciones...</h1>";
  html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></p>";
  html += "</body></html>";
  server.send(200, "text/html", html);

  ModoComunicacion = 0;
}

void handleDir4FormularioLecturas() {
  String html = "<html><body>";
  html += "<h2>Filtrar lecturas entre fechas</h2>";
  html += "<h4>Maximo 30 dias de diferencia entre fechas<h4><br>";
  html += "<form action='/lecturas' method='GET'>";
  html += "<label for='inicio'>Fecha de inicio (dd/mm/yyyy):</label><br>";
  html += "<input type='text' id='inicio' name='inicio' placeholder='Ej: 01/01/2025' required><br><br>";
  html += "<label for='fin'>Fecha de fin (dd/mm/yyyy):</label><br>";
  html += "<input type='text' id='fin' name='fin' placeholder='Ej: 01/02/2025' required><br><br>";
  html += "<input type='submit' value='Filtrar lecturas'>";
```

```
315   html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></
      p>";
316   html += "</form>";
317   html += "</body></html>";
318 
319   server.send(200, "text/html", html);
320 }
321 
322 void handleDir5Lecturas() {
323   String fechaInicio = server.arg("inicio");
324   String fechaFin = server.arg("fin");
325   String resultado;
326 
327   if (fechaInicio != "" && fechaFin != "") {
328     resultado = obtenerLecturasEntreFechas(fechaInicio, fechaFin);
329   } else {
330     resultado = "<h2>Error: Debes ingresar ambas fechas.</h2>";
331   }
332 
333   resultado = convertirSaltosDeLineaHTML(resultado);
334   String html = "<html><body>";
335   html += resultado;
336   html += "<br><a href='/'>Volver al formulario</a>";
337   html += "</body></html>";
338 
339   server.send(200, "text/html", html);
340 }
341 
342 void handleDir6() {
343   String html = "<html><body>";
344   html += "<h2>Segundos entre lecturas = ";
345   html += configuracionsegundoslectura;
346   html += "</h2>";
347   html += "<p><button onclick=\"window.location.href='/'\">Volver al menu</button></
      p>";
348   html += "</body></html>";
349   server.send(200, "text/html", html);
350 }
351 
352 // Funciones para los circuitos integrados
353 void resetCounters() {
354   digitalWrite(RESET_PIN, LOW);
355   delayMicroseconds(10);
356   digitalWrite(RESET_PIN, HIGH);
357 }
358 
359 uint16_t readCounters() {
360   digitalWrite(CLK_EN_PIN, HIGH); // 1. Inhibe el reloj de los contadores
361   delay(1);
362   digitalWrite(LOAD_PIN, LOW);    // 2. Carga los datos en el 74HC165 (LOAD a LOW)
363   delay(1);
364   digitalWrite(LOAD_PIN, HIGH);   // Vuelve a desactivar la carga
365   delay(1);
```

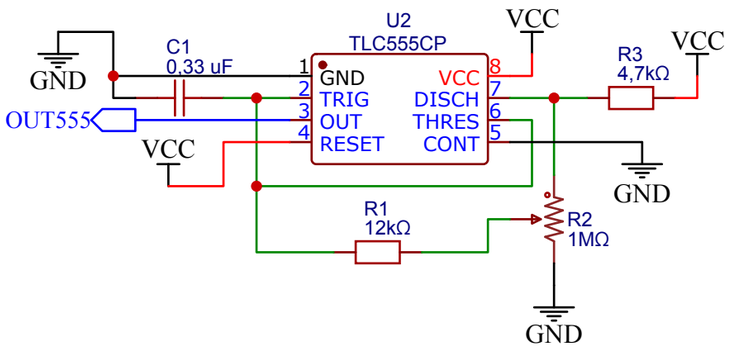
```
419   if (currentTime - lastInterruptTimeAzul > debounceDelay) {
420     BotonAzul_isAsleep = !BotonAzul_isAsleep;
421   }
422 }
423 
424 // Interrupción lectura
425 void IRAM_ATTR onTimer() {
426 timerFlag = true;
427 }
428 
429 void setup() {
430 
431   Serial.begin(9600);             // Inicia el puerto serie para poder monitorizar
    desde un PC
432 
433   SPI.begin(18, 19, 23, 15);      // Inicia el protocolo SPI -> (SCK, MISO, MOSI, CS)
434   SPI.setDataMode(SPI_MODE0);     // SPI MODO 0
435 
436   pinMode(PIN_LED, OUTPUT);
437   ledcSetup(0, 5000, 8);
438   ledcAttachPin(PIN_LED, 0);
439   ledcWrite(0, 255);
440   delay(2000);
441 
442   // Inicia la tarjeta SD
443   if(!SD.begin(15)){
444     Serial.println("Tarjeta SD no iniciada");
445     // return;
446   }
447 
448   File file = SD.open("/ultimalectura.txt", FILE_READ);
449   if (file) {
450     litrosCount = file.parseInt();
451     file.close();
452     Serial.print("Valor leído desde el archivo: ");
453     Serial.println(litrosCount);
454   } else {
455     Serial.println("No se encontró el archivo. Usando valor por defecto.");
456   }
457 
458   // Inicia la pantalla
459   TFT.begin(5);
460   TFT.setRotation(2);
461   TFT.fillScreen(TFT_BLACK);
462   TFT.setCursor(120,120);
463   TFT.print("ENCENDIENDO...");
464 
465   // Inicia WiFi con SSID y contraseña
466   WiFi.begin(ssid, pass);
467   while(WiFi.status() != WL_CONNECTED) {
468     Serial.print(".");
469     delay(10);
470   }
```

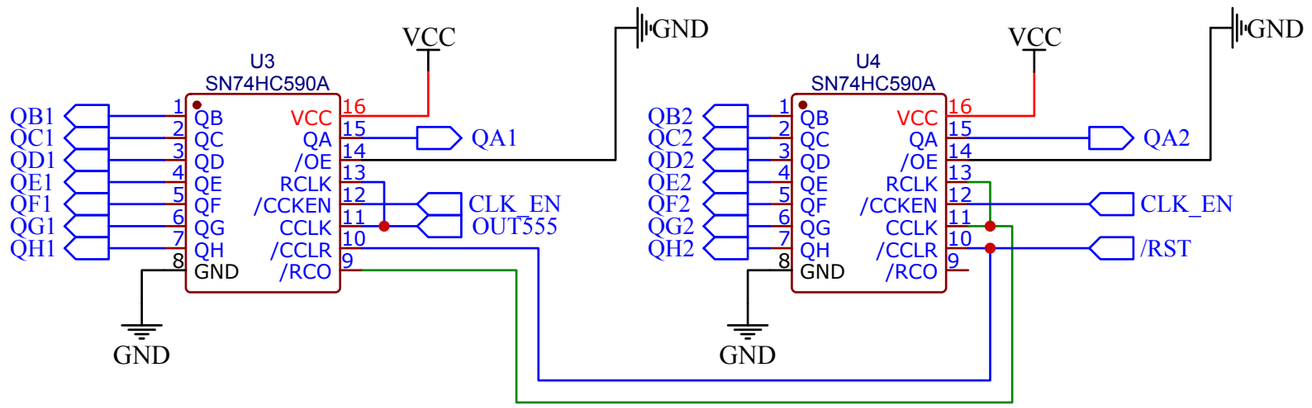
```
471
472   Serial.print("Se ha conectado al wifi con la ip: ");
473   Serial.println(WiFi.localIP());
474
475   // Configurar la obtención de la hora desde el servidor NTP
476   configTime(3600, 3600, ntpServer);
477
478   if (!getLocalTime(&timeinfo)) {   // Espera a obtener la hora
479     Serial.println("Fallo al obtener la hora");
480   }
481
482   // Declaración de pines
483   pinMode(LOAD_PIN, OUTPUT);
484   pinMode(CLK_PIN, OUTPUT);
485   pinMode(DATA_PIN, INPUT);
486   pinMode(RESET_PIN, OUTPUT);
487   pinMode(CLK_EN_PIN, OUTPUT);
488
489   pinMode(PIN_BLANCO, INPUT_PULLUP);  // Pin botón blanco como entrada
490   pinMode(PIN_VERDE, INPUT_PULLUP);   // Pin botón verde como entrada
491   pinMode(PIN_AZUL, INPUT_PULLUP);    // Pin botón azul como entrada
492
493   resetCounters();  // Inicia contadores en 0
494
495   // Define la rutas y la función que la manejará
496   server.on("/", handleRoot);
497   server.on("/dir1", handleDir1);
498   server.on("/dir2", handleDir2);
499   server.on("/dir3", handleDir3);
500   server.on("/dir4", handleDir4FormularioLecturas);
501   server.on("/lecturas", handleDir5Lecturas);
502   server.on("/dir6", handleDir6);
503   server.on("/dir7",   HTTP_GET, []() {
504     File file = SD.open("/hora.csv", FILE_READ);
505     if (file) {
506       server.sendHeader("Content-Type", "text/csv");
507       server.sendHeader("Content-Disposition", "attachment; filename=\"hora.csv\"");
508
509       server.streamFile(file, "text/csv");  // Enviar el archivo como respuesta
510
511       file.close();
512     } else {
513       server.send(404, "text/plain", "Archivo no encontrado");
514     }
515   });
516
517   // Iniciar mDNS con el nombre de dominio "alejandrovelatfg"
518   if (!MDNS.begin("alejandrovelatfg")) {
519     Serial.println("Error al iniciar mDNS");
520     return;
521   }
522
523   // Inicia el servidor
```

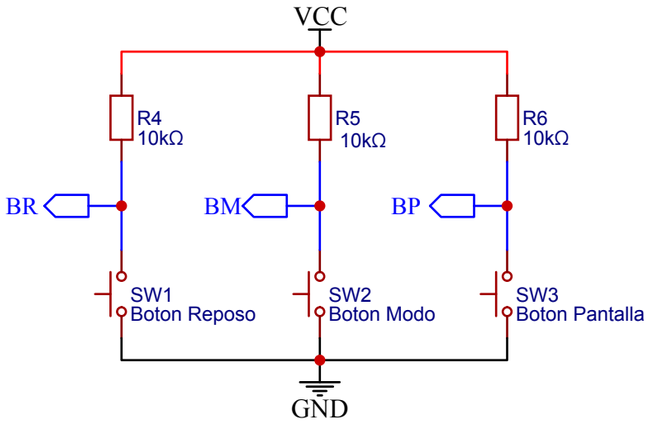
```
524   server.begin();
525
526   // ** Interrupciones **
527   Serial.println("Iniciando interrupciones...");
528
529   // Temporizador para registrar el consumo en la SD
530
531   // Configurar el temporizador (0: primer temporizador, 80: divisor de frecuencia para
    1 MHz)
532   timer = timerBegin(0, 80, true);
533   // Adjuntar la función de manejo de interrupción al temporizador
534   timerAttachInterrupt(timer, &onTimer, true);
535   // Establecer el temporizador para dispararse cada 4 minutos
536   timerAlarmWrite(timer, configuracionsegundoslectura * 1000000, true);
537   // Iniciar el temporizador
538   timerAlarmEnable(timer);
539
540   // Interrupciones botones
541   attachInterrupt(digitalPinToInterrupt(PIN_BLANCO), handleInterruptBlanco, RISING);
542   attachInterrupt(digitalPinToInterrupt(PIN_VERDE), handleInterruptVerde, RISING);
543   attachInterrupt(digitalPinToInterrupt(PIN_AZUL), handleInterruptAzul, RISING);
544
545   mostrarUltimosConsumos();
546 }
547
548 void loop() {
549   int pulseCount = 0;     // Variable que registra todos los pulsos leídos
550
551   // Si el estado es "reposo" (isAsleep es verdadero), entraremos en modo de reposo
552   if (BotonAzul_isAsleep) {
553     mostrarArribaDerecha("Entrando en modo reposo...", TFT_ORANGE);
554     delay(5000);
555     mostrarArribaDerecha("MODO REPOSO               ", TFT_RED);
556
557     // Configuramos la interrupción para despertar el ESP32 cuando el botón se presione
558     esp_sleep_enable_ext0_wakeup(GPIO_NUM_32, 0); // GPIO32 activará la interrupción
    (nivel bajo)
559     // Configura temporizador de despertar (en microsegundos)
560     esp_sleep_enable_timer_wakeup(240 * 1000000); // 4 minutos
561
562     // Entramos en modo de reposo profundo
563     esp_light_sleep_start();
564     onTimer();
565     BotonAzul_isAsleep = true;
566   }
567   if (timerFlag) {
568
569     timerFlag = false;  // Restablecer la bandera
570     Serial.println("Interrupción de 4 minutos activada");
571
572     pulseCount = readCounters();
573     litrosCount += pulseCount/340;               // Ecuacion dada por el caudalímetro
574     RotarActualizarArray(pulseHistory, litrosCount);
```

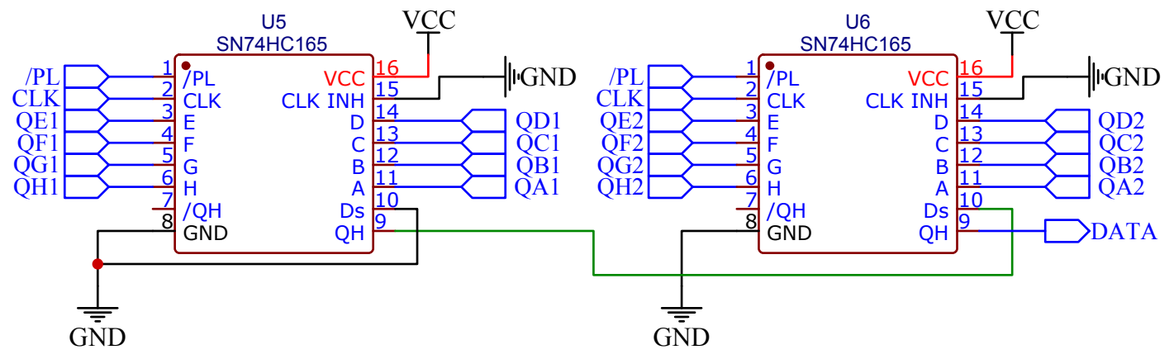
```
575     RotarActualizarTiempo(pulseTimes);
576 
577     sprintf(litrosTotales, "%08d", litrosCount); // Rellena con 0, un tamaño total de
    8, decimal.
578 
579     Serial.printf("Valor de litros actuales: %d L\n", litrosCount);
580 
581     mostrarUltimosConsumos();         // Actualizar la pantalla TFT
582     tiempoEnEpoch = mktime(&timeinfo);// Convierte en epoch la fecha actual para
    escribirla en la SD
583 
584     // Guarda en la SD los últimos litros de lectura, fecha y hora
585     File file = SD.open("/hora.csv", FILE_APPEND);  // Abre el archivo en modo agregar
586     if (file) {
587       file.print(litrosTotales);
588       file.print(",");
589       file.println(tiempoEnEpoch);
590       file.close();                   // Cerrar el archivo
591       Serial.println("Hora guardada en hora.csv");
592     }
593     else {
594       Serial.println("Error al abrir el archivo para escribir.");
595     }
596     file = SD.open("/ultimalectura.txt", FILE_WRITE);
597     if (file){
598       file.println(litrosCount);
599       file.close();
600       Serial.println("Valor de litros guardado en ultimalectura.txt");
601     }
602     else {
603       Serial.println("Error al abrir el archivo para escribir.");
604     }
605   }
606 
607   if(!BotonAzul_isAsleep){
608     if(ModoComunicacion == 0){ // No hay comunicaciones
609       mostrarArribaDerecha("SIN COMUNICACIONES", TFT_YELLOW);
610     }
611     if(ModoComunicacion == 1){
612       server.handleClient();    // Gestiona las peticiones al servidor web
613       mostrarArribaDerecha("COMUNICACION WIFI ", TFT_CYAN);
614     }
615   }
616 }
```

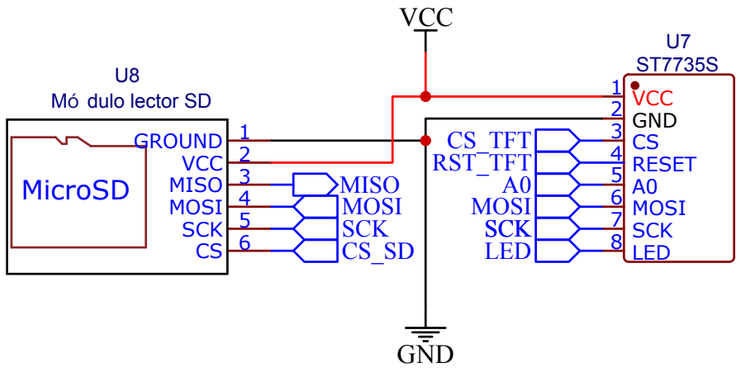
Anexo C
Esquema eléctrico

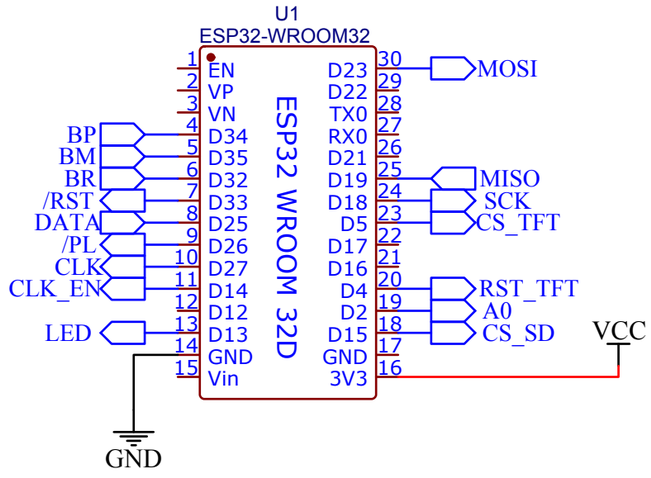












TITLE: Supervisor consumo de agua		REV: 1.0
EasyEDA	Company: TFG Alejandro Vela Alarcó n	Sheet: 1/1
	Date: 2025-04-23 Drawn By: AVA	

Anexo D
Lista de componentes

Nº	Referencia	Componente	Cantidad	Descripción	Modelo	Fabricante
1	U1	ESP32	1	ESP32 DEVKITV1	NodeMCU-ESP32	Guangzhou Xingjiayi Network Technology Co., Ltd
2	U2	Temporizador	1	TLC555CP	TLC555CP	Texas Instruments
3	U3, U4	Contador síncrono 16 bits	2	74HC590	SN74HC590A	Texas Instruments
4	U5, U6	Registro de desplazamiento paralelo-serie	2	74HC165	SN74HC165	Texas Instruments
5	U7	Pantalla TFT 2,4”	1	2.4 tft lcd module	MSP2402	Shenzhen Yanxinda Electronics Co., Ltd
6	U8	Lector tarjeta microSD	1	2.4 tft lcd module	MSP2402	Shenzhen Yanxinda Electronics Co., Ltd
7	SD1	Tarjeta microSD	1	Lexar SDHC 4GB 100X	LSDESXX004G-BNNNG	Lexar
8	R1	Resistencia TLC555	1	12 kΩ; 0,25 W; 5%; Axial; Película de carbón	739-7540	RS Pro
9	R2	Potenciómetro	1	monovuelta,horizontal; 1MΩ; 0,15W; ±30%	CA9V 1M	ACP
10	R3	Resistencia	1	4,7 kΩ; 0,25 W; 5%; Axial; Película de carbón	707-7726	RS Pro
11	R4	Resistencia	1	10 kΩ; 0,25 W; 5%; Axial; Película de carbón	707-7745	RS Pro
12	R5	Resistencia	1	10 kΩ; 0,25 W; 5%; Axial; Película de carbón	707-7745	RS Pro
13	R6	Resistencia	1	10 kΩ; 0,25 W; 5%; Axial; Película de carbón	707-7745	RS Pro
14	C1	Condensador	1	330 nF; 10%; 50V; Cerámico	SR305C334KARTR	KYOCERA AVX
15	SW1	Botón pulsador	1	Pulsador 12x12x7 mm azul	T438	GTIWUNG
16	SW2	Botón pulsador	1	Pulsador 12x12x7 mm verde	T438	GTIWUNG
17	SW3	Botón pulsador	1	Pulsador 12x12x7 mm blanco	T438	GTIWUNG
18	PCB1	Placa de prototipado	1	Solderless Breadboard	MB-102	Shenzhen Doraya Industrial Co., Ltd.
19	—	Cables	60	Cables de dupont para arduino	4110	Kitronik