Fases de Desarrollo del Firmware (Metodología Ágil)

El desarrollo del *firmware* se ha estructurado en fases incrementales para asegurar la estabilidad y el funcionamiento modular del sistema antes de integrar la lógica compleja de control.

Fase 1: Arquitectura Lógica y Diseño de Datos

Objetivo Central: Establecer la arquitectura modular del código y diseñar la estructura de datos compartida, asegurando la separación de responsabilidades entre los componentes lógicos.

- **1.1 Definición Modular:** Separar el código en archivos (main.ino, .cpp, .h) para definir las responsabilidades de cada módulo (Control, I/O, Sensores).
- **1.2 Diseño del Estado Compartido:** Diseñar la estructura de datos global (EstadoCompartido) que contendrá toda la información relevante del sistema (estado actual, flags de botones, volumen, errores).
- **Criterio de Finalización:** Existencia de un diseño documentado del *struct* de datos compartidos y un mapa de comunicación que define qué módulo accede a cada dato.

Fase 2: Andamiaje de FreeRTOS e Inicialización

Objetivo Central: Implementar el entorno concurrente de FreeRTOS, verificando la creación y ejecución independiente de las tareas del sistema.

- **2.1 Esqueleto de Tareas:** Crear la estructura de 2 a 3 tareas básicas (e.g., TaskControl, TaskDebug) con una funcionalidad mínima (solo impresión en serie).
- **2.2 Inicialización del RTOS:** Configurar el setup() para inicializar el sistema y crear las tareas con xTaskCreatePinnedToCore.
- **Criterio de Finalización:** Confirmación visual en el monitor serie de que todas las tareas se están ejecutando concurrentemente sin bloqueos ni reinicios del microcontrolador.

Fase 3: Estado Compartido y Gestión de Concurrencia

Objetivo Central: Proteger la integridad de los datos compartidos mediante la implementación de un Mutex, resolviendo el problema de acceso concurrente entre tareas.

- **3.1 Implementación del Mutex:** Crear el *SemaphoreHandle* (Mutex) para proteger la estructura de EstadoCompartido.
- **3.2 Rutinas de Acceso Seguro:** Implementar las funciones de lectura y escritura (leerEstado(), escribirEstado()) que utilicen el Mutex.
- **Criterio de Finalización:** Verificación de que una tarea puede modificar el estado y otra puede observarlo correctamente, demostrando que el Mutex gestiona el acceso sin corrupción de datos.

Fase 4: Integración de Entradas Físicas (Botones)

Objetivo Central: Desarrollar un módulo de lectura estable de los botones, mitigando el efecto de rebote (*debounce*).

- **4.1 Lógica de *Debounce*:** Implementar el algoritmo de *debounce* para la lectura digital de los pines de los botones (START, STOP, LLENAR).
- **4.2 Módulo de Tareas:** Integrar la lectura de botones dentro de una tarea periódica que actualice los *flags* correspondientes en EstadoCompartido.
- **Criterio de Finalización:** Demostración de una única actualización de estado por cada pulsación de botón, sin lecturas duplicadas o falsos positivos por rebote.

Fase 5: Control de Actuador (Bomba)

Objetivo Central: Integrar y controlar el actuador principal (minibomba) a través de la etapa de potencia, validando la salida digital.

- **5.1 Funciones de Control:** Implementar las funciones bombaOn() y bombaOff() para manejar la señal digital de activación de la etapa de potencia.
- **5.2 Pruebas Aisladas:** Realizar pruebas de encendido/apagado de la bomba controladas por los *flags* de los botones, sin intervención de la lógica de la FSM.
- **Criterio de Finalización:** Confirmación de que la minibomba se activa y desactiva de forma fiable en respuesta a las condiciones de entrada, verificando la conexión física y de potencia.

Fase 6: Integración del Sensor Ultrasónico

Objetivo Central: Implementar la lectura del sensor ultrasónico y aplicar lógica de estabilidad para una detección fiable de la presencia de la botella.

- **6.1 Medición y Filtrado:** Desarrollar la función medirDistanciaCm() y aplicar un filtrado básico para asegurar la estabilidad de la lectura.
- **6.2 Actualización de Estado:** Integrar la lectura dentro de una tarea periódica que convierta la distancia medida en un valor booleano (botellaPresente) y actualice EstadoCompartido.
- **Criterio de Finalización:** El sistema reporta de forma suave y precisa la presencia o ausencia de la botella al mover el objeto dentro y fuera del rango definido.

Fase 7: Medición de Flujo y Volumen (ISR)

Objetivo Central: Utilizar interrupciones externas para obtener un conteo preciso de los pulsos del sensor de flujo y calcular el volumen en mililitros.

- **7.1 Implementación de la ISR:** Desarrollar la rutina de interrupción de servicio (ISR) que incremente un contador al detectar cada pulso del sensor de flujo.
- **7.2 Tarea de Cálculo:** Crear una tarea que lea el contador acumulado periódicamente, aplique el factor de calibración (PULSOS_POR_LITRO) y actualice EstadoCompartido.volumenMedidoMl.
- **Criterio de Finalización:** Tras una prueba de llenado controlada, el volumen medido en serie es congruente con el volumen físico llenado, demostrando la precisión del sensor y la ISR.

Fase 8: Desarrollo de la Máquina de Estados (FSM Lógica)

Objetivo Central: Diseñar e implementar el núcleo de la lógica de control del sistema en forma de Máquina de Estados Finita.

- **8.1 Bosquejo de FSM:** Definir formalmente todos los estados (DETENIDO, ESPERANDO, LLENANDO, ERROR, etc.) y las transiciones entre ellos.
- **8.2 Implementación "en Seco":** La TaskControl se implementa para leer entradas y cambiar de estado según la FSM, pero **sin tocar la bomba ni el volumen** (solo imprimiendo mensajes).

- **Criterio de Finalización:** El monitor serie muestra una secuencia de estados lógicamente correcta al simular las entradas (pulsar botones, simular botella) según el diagrama de la FSM.

Fase 9: Integración Funcional (FSM Completa)

Objetivo Central: Conectar las salidas de la FSM con los actuadores y sensores para lograr el comportamiento de llenado automático.

- **9.1 Ajuste de Transiciones:** Modificar la FSM para que, al entrar en el estado **LLENANDO**, active la bomba, reinicie el contador de flujo y, al alcanzar el volumen objetivo, la desactive.
- **9.2 Manejo de Errores:** Implementar la lógica de transición al estado **ERROR** ante fallas críticas (p. ej., botella retirada durante llenado, *timeout*).
- **Criterio de Finalización:** El sistema realiza un ciclo completo de llenado automático (de **ESPERANDO BOTELLA** a **LLENADO COMPLETADO**) con la precisión y detenciones de seguridad esperadas.

Fase 10: Interfaz de Usuario (Pantalla LCD)

Objetivo Central: Desarrollar una tarea dedicada a la interfaz de usuario, asegurando que la visualización de datos sea no bloqueante y solo de lectura.

- **10.1 Diseño de UI:** Definir qué información crítica se muestra en cada línea de la pantalla (estado, volumen o mensaje de error).
- **10.2 Tarea de Observación:** Implementar una TaskUI que **solo lea** la información de EstadoCompartido (a través del Mutex) y actualice la pantalla LCD.
- **Criterio de Finalización:** La pantalla muestra de forma estable y clara el estado actual del sistema a lo largo de todo el ciclo de llenado y en las condiciones de error.

Fase 11: Afinamiento y Documentación Final

Objetivo Central: Optimizar el rendimiento, revisar la calidad del código y completar toda la documentación necesaria para la entrega del proyecto.

- **11.1 Optimización del RTOS:** Revisar las prioridades de las tareas de FreeRTOS y los tiempos de *delay* para asegurar la eficiencia del sistema.
- **11.2 Calidad de Código:** Realizar la limpieza del código, refactorización y comentarios exhaustivos en zonas críticas (FSM, ISR, Mutex).
- **Criterio de Finalización:** El *firmware* pasa una prueba de resistencia sin fallos y la documentación (README.md, informes, diagramas) está completa y alineada con el código final.

No tomada en cuenta en el README.md porque esta es una etapa de prueba y terminar de modulizar, eliminar números fantasmas, entre otros