Implementadas

1. **Compatibilidad con Diferentes Dispositivos y Contextos de Prueba**
    - **Problemática**: Se requiere probar el sistema en varios dispositivos y en distintos modos de operación.
    - **Solución**: Implementación de opciones de precompilación para facilitar configuraciones específicas de cada dispositivo. Soporte para modos de ejecución en: *demo*, *calibración* y *ensayo*, permitiendo ajustar el comportamiento del sistema según el contexto.
2. **Necesidad de Depuración en Tiempo Real**
    - **Problemática**: Es esencial depurar el sistema y observar su comportamiento en tiempo real para un análisis adecuado.
    - **Solución**: Integración de comandos de depuración mediante el puerto serie para el envío y monitoreo de logs. Soporte para diferentes niveles de logeo (en formato JSON y plotter) que permiten una visualización detallada y en tiempo real de las señales y estados del sistema.

Problemáticas

1. **Configuración Dinámica de Parámetros de Ensayo y Calibración**
    - **Problemática**: Es necesario ajustar y guardar datos de configuración y calibración para la ejecución de ensayos repetitivos.
    - **Solución**: Soporte de administración de datos en memoria para permitir ajustes persistentes de parámetros de calibración y ensayo, asegurando que estos datos se mantengan incluso tras un reinicio.
2. **Manejo de Entrada y Salida de Datos**
    - **Problemática**: El sistema debe gestionar de manera eficiente tanto la entrada como la salida de datos para interactuar con otros dispositivos y con el usuario.
    - **Solución**: Implementación de un servidor web con peticiones HTTP GET y PUT para la entrada y salida de datos. Adicionalmente, el puerto serie está habilitado para la salida de datos, facilitando la integración con herramientas externas y permitiendo un registro de datos versátil y en tiempo real.

Diseño de Máquina de Estados y Métodos No Bloqueantes

Este proyecto de control de LED en ESP32 implementa una arquitectura modular que utiliza una **máquina de estados** para gestionar la lógica de control y permite el desarrollo de sistemas de control paralelos y complejos. El uso de métodos no bloqueantes en el ESP32 es crucial para garantizar que múltiples procesos se ejecuten sin interferencias, lo cual es común en aplicaciones industriales y de control embebido.

Ventajas de los Métodos No Bloqueantes y la Ejecución en Paralelo

1. **Máquinas de Estados en Paralelo**
    - Al utilizar métodos no bloqueantes (por ejemplo, con temporizadores y funciones sin "delay"), se pueden ejecutar varias máquinas de estados en paralelo. En este contexto, el sistema puede gestionar varios subsistemas al mismo tiempo, como el control de un LED y la gestión de una puerta motorizada.
    - Este diseño permite que cada máquina de estados funcione de forma independiente pero dentro de un ciclo principal de control, facilitando así que cada proceso esté sincronizado y responda en tiempo real sin bloquear el funcionamiento general del sistema.

Control de Sistemas Realimentados (PID)

- En aplicaciones avanzadas, es común el uso de controladores PID (Proporcional, Integral, Derivativo) en sistemas realimentados para ajustar automáticamente la respuesta de un dispositivo en función de un error medido.
- En este proyecto, la máquina de estados principal puede incorporar una sub-máquina de estados para gestionar el control PID en paralelo, permitiendo un control preciso en aplicaciones como la posición de una puerta motorizada, donde la posición o velocidad deben ser reguladas de forma continua.

Filtros de Señales para Datos en Tiempo Real

- En sistemas embebidos, es común la aplicación de filtros digitales para mejorar la calidad de las señales de entrada antes de procesarlas. Un ejemplo de este proyecto es el uso de filtros para "suavizar" las señales de sensores ruidosos. Al emplear métodos no bloqueantes, el sistema puede aplicar filtros sin interrumpir otros procesos.
- Los filtros digitales, como el filtro de media móvil o el filtro de Kalman, pueden integrarse en las máquinas de estados de forma que cada lectura de sensor se procese para eliminar ruido, asegurando así que los datos sean precisos y fiables.
    -

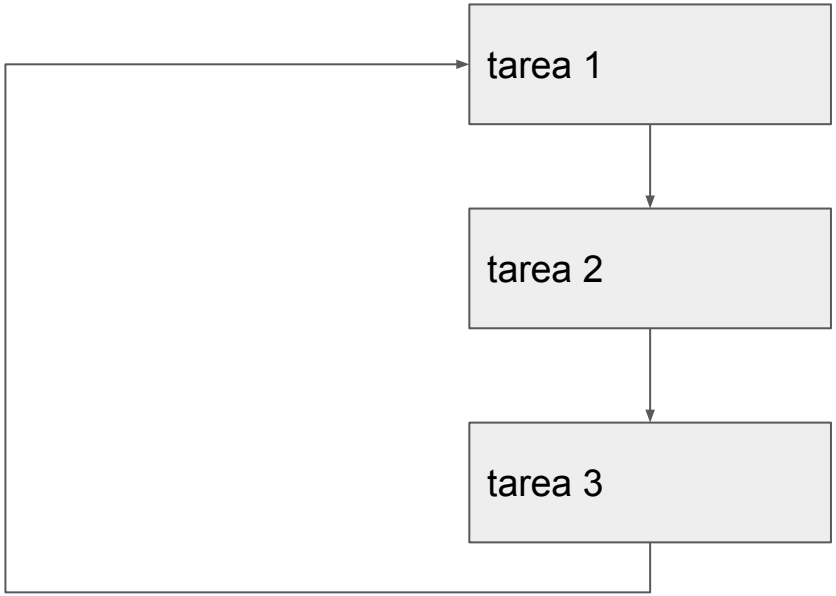
1. **Estados Propios de la Funcionalidad**
    - La máquina de estados debe diseñarse con estados específicos para cada fase de operación, como los estados de encendido y apagado del LED en el ejemplo. Los proyectos más complejos pueden requerir estados adicionales que permitan configuraciones específicas o pruebas de diagnóstico.
    - En este proyecto, los estados principales incluyen `ST_LOOP_INIT`, `ST_LOOP_IDLE`, `ST_LOOP_LED_ON`, y `ST_LOOP_LED_OFF`, con la posibilidad de extenderse a estados de ejecución en modos específicos (como *demo*, *ensayo*, o *calibración*). Esto permite una transición fluida entre tareas y una operación optimizada según la fase de ejecución.

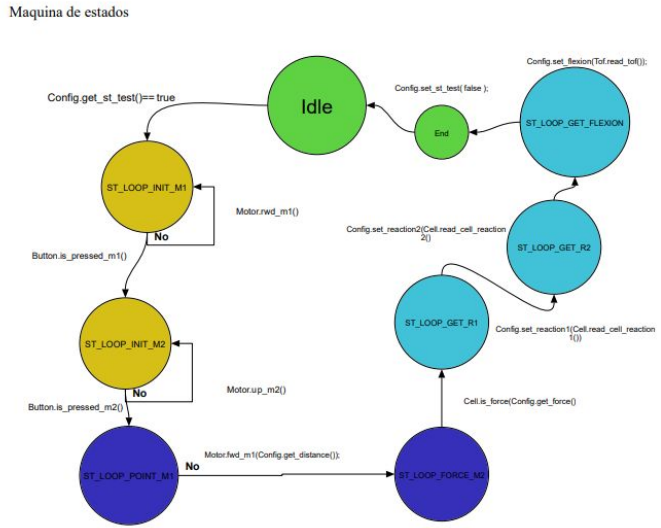
Ejemplo de Expansión en Proyectos Reales

En un entorno industrial, esta misma estructura puede ampliarse para gestionar sistemas como un brazo robótico, donde cada articulación requiere un control preciso de posición y velocidad mediante un PID en tiempo real, además de una interfaz para monitoreo y ajustes de configuración. La arquitectura modular y no bloqueante permitiría, además, que los diferentes motores del brazo operen de forma coordinada y sin interrupciones, asegurando que el sistema completo funcione de manera eficiente y sin latencias.

Para pasar al próximo estado , solo verifica si terminó o no la tarea.

Máquinas de estados





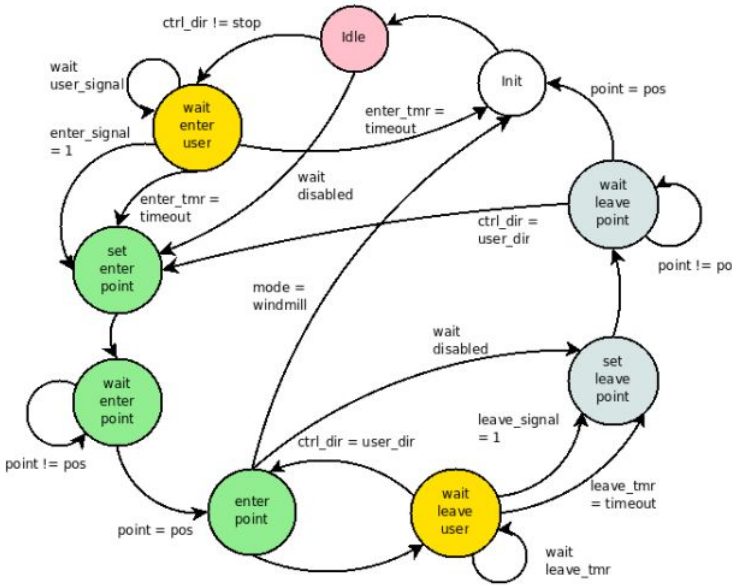
ver codigo flexion_viga.ino

Una condición se implementa con switch case

Para pasar al próximo estado , verifica más de una condición

Máquinas de estados

Control de puntos




```
/**
 * Si se esta cerrando y se vuelve a solicitar una apertura igual al anterior.
 */
else if( (key != IDLE_STATE) && (mov_state != key) && (door_state == IDLE_STATE) )
{
    mov_state = key;
    door_state = key;

    p_point = open_door( key, p_point );
}

/**
 * Codigo que se ejecuta cuando la puerta esta abierta.
 */
else if( (cfg_get_mode() == MODE_DOOR) && (door_state != IDLE_STATE) && (mov_state == IDLE_STATE) )
{
    /**
     * Dispara el timer que controla el tiempo que la puerta permanece
     * abierta.
     */
    if( tmr_get_timer( TIMER_DOOR ) == -1 )
    {
        tmr_set_timer( TIMER_DOOR, p_point->acs_time );
    }
    else if ( tmr_get_timer( TIMER_DOOR ) == 0 )
    {
        /**
         * Cuando el timer expira, lo apaga y selecciona el estado de retorno
         * dependiendo del sentido de apertura.
         */
        tmr_timer_off( TIMER_DOOR );
        pid_reset();

        if( door_state == REW_STATE )
```

Ln 133, Col 12 Tab Size: 4

10°C Mayorm. nub

ver codigo main.c puerta motorizada

mayor a condición se implementa con if else

Sistema de eventos

Evolución de la Máquina de Estados a un Sistema Basado en Eventos

Cuando un sistema embebido se vuelve lo suficientemente complejo, una simple máquina de estados puede llegar a sus límites. Esta complejidad no es necesariamente el resultado de un mal diseño, sino de la naturaleza avanzada y multifuncional de la aplicación (por ejemplo, sistemas de automatización industrial, domótica compleja, o dispositivos de IoT con múltiples sensores y actuadores). En estos casos, se recomienda evolucionar hacia una **arquitectura basada en eventos**, que permite gestionar múltiples fuentes de eventos de forma eficiente y escalable.

Sistemas Basados en Eventos

Un sistema de eventos organiza la ejecución en función de la ocurrencia de eventos externos (por ejemplo, señales de sensores) e internos (por ejemplo, temporizadores y flags de estado). La implementación de un sistema basado en eventos permite separar claramente cada proceso y facilita la coordinación entre tareas complejas, pero también introduce una mayor complejidad en la implementación y el mantenimiento del código.

Para gestionar este tipo de arquitectura, se pueden emplear dos enfoques principales: **multihilos** (o concurrencia con hilos) y **polling**. Ambos tienen sus ventajas y desventajas, y su uso depende del contexto y de los requisitos específicos del sistema.

Sistema de eventos

Enfoque de Ejecución con Hilos (Multithreading)

1. **Descripción**
    - En un sistema de hilos, cada tarea o proceso crítico se asigna a un hilo separado que puede ejecutarse en paralelo (especialmente en procesadores de múltiples núcleos como el ESP32).
    - Esta ejecución en paralelo permite que cada proceso tenga su propio "espacio de trabajo" y sea independiente de los demás, lo cual es ideal para sistemas donde diferentes eventos requieren procesamiento en tiempo real o casi real.
2. **Ventajas del Multithreading en Sistemas Basados en Eventos**
    - **Paralelismo Real**: Permite la ejecución real en paralelo de tareas, reduciendo la posibilidad de interferencias entre procesos críticos.
    - **Escalabilidad**: Facilita la expansión de funciones y módulos, ya que se pueden añadir nuevos hilos para nuevas tareas sin impactar los hilos existentes.
    - **Mejor Uso de Procesadores Multinúcleo**: En dispositivos como el ESP32, que cuenta con varios núcleos, los hilos pueden asignarse a diferentes núcleos, mejorando la eficiencia y el rendimiento general.

Desventajas del Multithreading

- **Complejidad de Sincronización**: La comunicación entre hilos puede requerir mecanismos de sincronización (como semáforos o mutexes) para evitar conflictos, lo cual incrementa la complejidad y el riesgo de errores.
- **Consumo de Recursos**: Cada hilo consume memoria y recursos del sistema, por lo que demasiados hilos pueden sobrecargar el sistema, especialmente en hardware con recursos limitados.

Enfoque de Ejecución con Polling

1. **Descripción**
   - En el enfoque de polling, el sistema revisa continuamente el estado de cada evento (sensores, temporizadores, etc.) en un bucle. Cuando detecta que una condición se cumple, ejecuta la acción correspondiente.
   - Aunque puede parecer menos eficiente que el multithreading, el polling es una alternativa válida en muchos sistemas embebidos debido a su simplicidad y bajo consumo de recursos.
2. **Ventajas del Polling en Sistemas Basados en Eventos**
   - **Simplicidad**: El polling es generalmente más fácil de implementar y depurar, ya que no requiere la compleja sincronización que el multithreading.
   - **Menor Consumo de Memoria**: Como el polling utiliza un solo bucle principal en lugar de múltiples hilos, tiende a consumir menos memoria y recursos, lo cual es ideal en dispositivos de baja potencia.
3. **Desventajas del Polling**
   - **Latencia**: El polling puede introducir latencia en la respuesta a eventos, especialmente si el bucle principal es largo y la tarea no se ejecuta de inmediato. Esto puede ser un problema en sistemas que necesitan respuestas en tiempo real.
   - **Ineficiencia en Tareas Simultáneas**: Si varios eventos requieren atención simultáneamente, el sistema puede experimentar retrasos al procesar cada evento en secuencia.

Sistema de eventos

Elección entre Multithreading y Polling en Sistemas de Eventos

La decisión de usar hilos o polling depende de los requisitos de la aplicación. Aquí algunos consejos para la selección:

- **Usar Multithreading** cuando el sistema tenga varios procesos críticos que deben ejecutarse en paralelo (por ejemplo, monitoreo de sensores en tiempo real y control de actuadores).
- **Usar Polling** en aplicaciones donde la simplicidad sea una prioridad y las tareas no sean tan críticas en cuanto a tiempo de respuesta (por ejemplo, sistemas de monitoreo pasivos o interfaces de usuario con baja demanda).

Para sistemas embebidos avanzados, una combinación de ambos enfoques también es posible: se pueden usar hilos para procesos críticos de baja latencia y polling para tareas de baja prioridad, optimizando así el rendimiento sin sobrecargar el sistema

IDF - Arduino -

Comparación: Arduino vs. ESP-IDF en el Desarrollo de Proyectos Embebidos Complejos con ESP32

Arduino

Ventajas:

1. **Simplicidad y Accesibilidad**: La plataforma Arduino proporciona una API fácil de usar y abstrae muchas complejidades. Esto permite a los desarrolladores implementar funcionalidad avanzada rápidamente, sin necesidad de aprender los detalles de bajo nivel.
2. **Compatibilidad y Soporte Extenso**: Con un gran ecosistema de bibliotecas y una comunidad activa, Arduino facilita el acceso a recursos y soporte para desarrollar desde prototipos hasta proyectos funcionales.
3. **Separación Automática de Núcleos**: En el ESP32, Arduino suele manejar automáticamente la asignación de procesos de comunicación en el núcleo PRO, mientras el usuario programa en el núcleo APP. Esto permite que los desarrolladores se concentren en el desarrollo de funcionalidades sin preocuparse por la concurrencia.
4. **Ideal para Proyectos Simples a Moderadamente Complejos**: Arduino cubre la mayoría de los casos de uso donde se requiere control de periféricos, comunicación básica, y manejo de entradas/salidas con bajo esfuerzo de implementación.

Desventajas:

1. **Limitaciones en la Gestión de Hilos**: Arduino, al manejar gran parte de la comunicación en un núcleo, puede limitar el rendimiento de aplicaciones que requieren procesamiento paralelo intenso, ya que los hilos personalizados son más difíciles de implementar de manera eficiente.
2. **Capacidades de Bajo Nivel Reducidas**: Aunque simplifica el desarrollo, Arduino abstrae los detalles de bajo nivel, lo cual puede limitar el control fino en aplicaciones de alto rendimiento que demandan acceso específico a hardware.
3. **Menor Optimización para Procesos en Tiempo Real**: La falta de un control fino sobre el manejo de memoria y tareas en Arduino puede ser una desventaja en aplicaciones donde el tiempo de respuesta es crítico.

IDF - Arduino -

ESP-IDF

Ventajas:

1. **Control Completo del Hardware**: ESP-IDF permite acceso directo al hardware del ESP32, proporcionando capacidades avanzadas de bajo nivel y opciones de configuración detallada que pueden optimizar el rendimiento para aplicaciones de tiempo real.
2. **Soporte Completo para Multithreading**: Gracias a su integración con FreeRTOS, ESP-IDF facilita la creación de hilos y permite al desarrollador asignar tareas específicas a cada núcleo del ESP32, lo cual es fundamental en aplicaciones con múltiples procesos concurrentes.
3. **Ideal para Sistemas Basados en Eventos Complejos**: La arquitectura de eventos de ESP-IDF, combinada con el uso de hilos y semáforos de FreeRTOS, es adecuada para proyectos que requieren manejo avanzado de eventos y procesamiento paralelo, como sistemas embebidos industriales y automatización avanzada.
4. **Optimización de Rendimiento en Aplicaciones Críticas**: ESP-IDF permite una optimización de recursos superior, útil en sistemas donde el uso eficiente de CPU, memoria, y periferias es crítico para el éxito del sistema.
1.

Desventajas:

1. **Mayor Complejidad**: La curva de aprendizaje de ESP-IDF es notablemente más empinada que la de Arduino, ya que requiere conocimientos en FreeRTOS, manejo de memoria, y configuración específica de hardware. Esto puede resultar prohibitivo para desarrolladores sin experiencia avanzada.
2. **Desarrollo y Depuración Más Lentos**: El ciclo de desarrollo es más lento debido a la necesidad de configurar y ajustar múltiples parámetros, en comparación con la simplicidad de Arduino. Esto puede ser un inconveniente en proyectos que requieren desarrollo rápido o constantes iteraciones.
3. **Sobrecarga en Aplicaciones Simples**: ESP-IDF puede ser excesivo y engorroso para proyectos donde no se requieren procesos de tiempo real o hilos múltiples, haciendo que la simplicidad de Arduino sea preferible en estos casos.

IDF - Arduino -

Impacto de los Nuevos ESP32 a 400 MHz y el Bajo Costo

Con la llegada de los ESP32 de última generación, que ofrecen velocidades de hasta 400 MHz y una buena relación costo-rendimiento, se ha vuelto más difícil justificar el uso de ESP-IDF para una gran mayoría de proyectos de complejidad baja o moderada. En muchos casos, Arduino es suficiente, incluso para aplicaciones con procesamiento intensivo.

1. **Desempeño Suficiente en Arduino**: A 400 MHz, el rendimiento del ESP32 en Arduino es más que adecuado para manejar procesos concurrentes de bajo o moderado nivel de complejidad. Esto hace que el sistema basado en Arduino, con su separación automática de núcleos, se mantenga funcional y eficaz sin necesidad de la complejidad adicional que trae el uso de hilos y sincronización avanzada en ESP-IDF.
2. **Mejor Costo-Beneficio en Proyectos Moderados**: Para aplicaciones comerciales de bajo costo que necesitan funcionalidad avanzada sin una complejidad de tiempo real extrema, Arduino en ESP32 de 400 MHz resulta más rentable en términos de tiempo y costos de desarrollo. Esto se debe a la simplicidad del framework Arduino y a la capacidad de manejar la mayoría de los casos de uso sin limitaciones notables de rendimiento.
3. **Flexibilidad de Arduino para Rápidas Iteraciones de Prototipos**: La simplicidad de Arduino facilita la iteración y el desarrollo rápido de prototipos, lo cual es beneficioso en entornos de desarrollo ágil. Con el hardware más poderoso y la simplicidad del framework, Arduino sigue siendo una opción atractiva para proyectos que necesitan evolucionar rápidamente.

IDF - Arduino -

1. **La elección entre Arduino y ESP-IDF depende de la complejidad del proyecto. En sistemas críticos, de alto rendimiento y que demandan un control preciso del hardware, ESP-IDF sigue siendo la opción preferida. Sin embargo, con el avance del hardware ESP32 estándar de 240MHz a un P4 de 400 MHz, Arduino es suficiente para la mayoría de aplicaciones moderadas, ofreciendo simplicidad, costos reducidos y tiempos de desarrollo rápidos, justificando su uso en muchos casos frente a la complejidad de ESP-IDF.**

   https://www.espressif.com/en/products/socs/esp32-p4

   repositorio: https://github.com/theinsideshine/esp32-led-arduino