UTN Universidad Tecnológica Nacional
Facultad Regional Tucumán
Departamento Electónica

Técnicas Digitales II

Actividad de Formación Teórica 1

Tema: Creación de drivers en STM32CubeIDE, programación de microcontroladores.

ALUMNOS:

- Palomo, José María
- Romano Sanchez, Marcos Joel
- Coronel Moreira, Carlos Javier
- Yapura, Yain Yasir

vencimiento: 27 de septiembre de 2024

Introducción

Ventajas de usar drivers en stm32

El uso de drivers en el desarrollo de proyectos con microcontroladores STM32 a través de STM32CubeIDE ofrece una serie de beneficios significativos que agilizan y optimizan el proceso de desarrollo. A continuación, se detallan las principales ventajas:

1. Abstracción del Hardware

- **Simplificación:** Los drivers ocultan la complejidad de los registros y configuraciones específicas de cada periférico, permitiendo al desarrollador centrarse en la lógica de la aplicación.
- **Portabilidad:** Facilita la adaptación del código a diferentes modelos de STM32, ya que los drivers suelen ser genéricos para una familia de periféricos.

Ejemplo: Imagina que quieres controlar un LED conectado a un pin GPIO de tu STM32. En lugar de tener que lidiar directamente con los registros del GPIO, puedes usar una función de un driver que se llame GPIO_SetPinState(GPIO_PortA, GPIO_Pin_5, GPIO_PIN_SET). Esta función se encarga de escribir el valor correcto en el registro correspondiente para encender el LED, ocultando la complejidad subyacente.

2. Reutilización de Código

- **Eficiencia:** Al encapsular la funcionalidad de un periférico en un driver, se puede reutilizar este código en múltiples proyectos, evitando la duplicación de esfuerzo.
- **Mantenibilidad:** Los cambios en la implementación de un periférico se realizan en un solo lugar, facilitando las actualizaciones y correcciones.

Ejemplo: Supongamos que tienes un proyecto donde necesitas leer datos de un sensor de temperatura y otro donde necesitas controlar un motor paso a paso. Ambos proyectos podrían utilizar drivers para los periféricos SPI o I2C, que son comunes para comunicarse con estos sensores y actuadores. Al reutilizar estos drivers, ahorras tiempo y esfuerzo en la implementación de estas funcionalidades.

3. Aumento de la Productividad

- **Desarrollo más rápido:** Los drivers preconfigurados permiten iniciar un proyecto rápidamente, reduciendo el tiempo de desarrollo.
- **Menor margen de error:** Al utilizar componentes de código probados y confiables, se disminuye la probabilidad de introducir errores en la aplicación.

Ejemplo: Estás desarrollando un sistema de control para un dron. En lugar de escribir desde cero el código para controlar los motores, puedes utilizar drivers preexistentes que te proporcionen funciones para configurar la velocidad, dirección y otros parámetros de los motores. Esto te permite concentrarte en la lógica de control del dron, acelerando significativamente el desarrollo.

4. Mejor Organización del Código

- **Modularidad:** Los drivers dividen el código en módulos bien definidos, lo que facilita la comprensión y el mantenimiento del proyecto.
- **Legibilidad:** Un código bien estructurado con drivers es más fácil de leer y entender, tanto para el desarrollador original como para otros.

Ejemplo: Imagina que tu proyecto involucra el uso de un sensor de temperatura, un acelerómetro y una pantalla LCD. Cada uno de estos componentes podría tener su propio driver, lo que permite organizar el código de manera modular. Por ejemplo, un archivo temperature_sensor.c contendría las funciones del driver para el sensor de temperatura, mientras que un archivo lcd_driver.c contendría las funciones para la pantalla LCD.

5. Facilidad de Depuración

- **Aislamiento de problemas:** Al identificar un error, es más sencillo aislar el problema en un módulo específico (el driver) y realizar las correcciones necesarias.
- **Herramientas de depuración:** STM32CubeIDE proporciona herramientas para depurar el código a nivel de driver, lo que agiliza la resolución de problemas.

Ejemplo: Si tu programa no funciona como esperas y sospechas que el problema está en la comunicación con el sensor de temperatura, puedes utilizar un depurador para establecer puntos de interrupción en las funciones del driver correspondiente. De esta manera, podrás examinar el estado de las variables y rastrear la ejecución del código paso a paso para identificar la causa del error.

6. Integración con STM32CubeMX

- **Generación automática de código:** STM32CubeMX genera automáticamente el código de configuración de los periféricos, incluyendo las llamadas a los drivers, lo que simplifica aún más el proceso de desarrollo.
- **Configuración gráfica:** La interfaz gráfica de STM32CubeMX permite configurar los periféricos de forma visual, lo que reduce la posibilidad de errores de configuración.

Ejemplo: Quieres configurar un ADC (convertidor analógico-digital) para leer una señal analógica. En STM32CubeMX, puedes seleccionar el ADC, configurar su resolución, velocidad de muestreo y otros parámetros de forma gráfica. Al generar el código, STM32CubeMX creará automáticamente las llamadas necesarias a los drivers para inicializar el ADC y leer los valores convertidos.

7. Soporte de la comunidad

- **Amplia documentación:** STMicroelectronics proporciona una extensa documentación y ejemplos de código para los drivers, lo que facilita la resolución de dudas.
- **Foros y comunidades:** Existen numerosas comunidades en línea donde los desarrolladores pueden compartir experiencias y resolver problemas relacionados con los drivers.

Ejemplo: Te encuentras con un problema al utilizar un driver SPI para comunicarse con un sensor específico. Puedes buscar en los foros de STMicroelectronics o en otras comunidades en línea para encontrar soluciones o ejemplos de código que te ayuden a resolver el problema.

Creación de driver en stm32

Estructura del Proyecto

- **Organización:** Se propone una estructura clara y jerárquica para organizar los archivos del proyecto, facilitando la gestión y mantenimiento del código.
- **Carpetas:** Se crean carpetas específicas para los archivos de cabecera (Inc), los archivos fuente (Src) y la API (Drivers).
- **Archivos:** Se generan archivos con nombres descriptivos para el archivo fuente (API_GPIO.c) y el archivo de cabecera (API_GPIO.h).

Creación del Driver GPIO

- **Interfaz:** Se define una interfaz clara para el driver, utilizando prototipos de funciones en el archivo de cabecera.
- **Implementación:** Se implementa el código de las funciones en el archivo fuente, encapsulando la lógica de control del GPIO.
- **Inicialización:** Se copia y adapta el código de inicialización del GPIO generado por STM32CubeMX al nuevo archivo fuente del driver.
- **Funciones adicionales:** Se añaden funciones para leer el estado de un pin, escribir en un pin, etc.
- **Uso en el main:** Se modifica el archivo main.c para utilizar las funciones del driver en lugar de las funciones de la HAL directamente.

Conceptos Clave

- **Abstracción:** El driver proporciona una capa de abstracción sobre el hardware, permitiendo al usuario interactuar con el GPIO de forma más sencilla y sin necesidad de conocer los detalles de los registros del microcontrolador.
- **Modularidad:** Al separar la funcionalidad del GPIO en un driver independiente, se facilita la reutilización del código en otros proyectos y se mejora la mantenibilidad.
- **Reutilización:** El driver puede ser utilizado en múltiples proyectos, ahorrando tiempo y esfuerzo en la programación.
- **Claridad y Concisión:** El código está bien estructurado y comentado, lo que facilita su comprensión y modificación.

Detalles para tener en cuenta

Al crear un driver GPIO para STM32, es fundamental tener en cuenta una serie de detalles para garantizar su correcto funcionamiento y evitar errores comunes. A continuación, se presentan algunos aspectos clave:

Comprensión Profunda del Hardware

- **Hoja de datos:** Estudia a fondo la hoja de datos del microcontrolador y del periférico GPIO. Conocer la arquitectura, los registros de control, las posibilidades de configuración y las limitaciones es esencial para una implementación correcta.
- **Configuración de reloj:** Asegúrate de que el reloj del GPIO esté habilitado y configurado correctamente. Un reloj insuficiente puede afectar el funcionamiento del periférico.
- **Restricciones de velocidad:** Respeta las limitaciones de velocidad de los pines GPIO, especialmente al manejar señales de alta frecuencia.

Diseño del Driver

- **Abstracción adecuada:** Define una interfaz clara y concisa para el driver, que oculte la complejidad del hardware y facilite su uso. Evita sobrecargar el driver con funcionalidades innecesarias.
- **Manejo de errores:** Implementa mecanismos para detectar y manejar errores, como por ejemplo, verificar el estado de retorno de las funciones de la HAL.
- **Modularidad:** Organiza el código del driver en funciones pequeñas y bien definidas, lo que facilita la lectura, el mantenimiento y la depuración.
- **Comentarios:** Incluye comentarios claros y concisos para explicar el propósito de cada función y las variables utilizadas.

Implementación del Código

- **Inicialización correcta:** Asegúrate de que el GPIO se inicialice correctamente, configurando la dirección, la velocidad, el modo (entrada/salida), la resistencia de pull-up/pull-down, etc.

- **Acceso a registros:** Utiliza las estructuras de datos y funciones proporcionadas por la HAL para acceder a los registros del GPIO de forma segura y eficiente.
- **Manejo de interrupciones:** Si vas a utilizar interrupciones, configura correctamente el vector de interrupciones y la rutina de servicio de interrupción (ISR).
- **Sincronización:** Si tienes múltiples tareas o procesos que acceden al GPIO, implementa mecanismos de sincronización para evitar conflictos de acceso.

Pruebas y Depuración

- **Pruebas unitarias:** Realiza pruebas unitarias para verificar el correcto funcionamiento de cada función del driver de forma aislada.
- **Pruebas de integración:** Integra el driver en tu aplicación y verifica que funciona correctamente en un entorno real.
- **Depuración:** Utiliza un depurador para identificar y corregir errores en el código.

Ejemplos de Errores Comunes

- **Configuración incorrecta del reloj:** Si el reloj del GPIO no está habilitado o está configurado con una frecuencia incorrecta, el periférico no funcionará correctamente.
- **Acceso a registros incorrecto:** Un acceso incorrecto a los registros del GPIO puede provocar un comportamiento inesperado o incluso dañar el hardware.
- **Conflictos de acceso:** Si múltiples tareas acceden al GPIO sin sincronización, pueden producirse errores de concurrencia.
- **Interrupciones no configuradas correctamente:** Si las interrupciones no están configuradas correctamente, el microcontrolador no responderá a los eventos del GPIO.

Aplicaciones desarrolladas:

Aplicación: App_1_1_Grupo_4_2024 MOD

Autor de la modificación: Coronel, Carlos

Observaciones: utilizar un driver en esta aplicación no solo simplifica la implementación, sino que también mejora la fiabilidad, la eficiencia y la escalabilidad del sistema. Al seleccionar el driver adecuado y diseñar un código bien estructurado, podrás crear un proyecto robusto y flexible.

Aplicación: App_1_2_Grupo_4_2024 MOD

Autor de la modificación: Palomo, Jose María

Observaciones: el uso de un driver en esta aplicación con pulsador te permite crear un sistema flexible y escalable, donde puedes controlar de manera precisa y eficiente la iluminación de los LED y alternar entre diferentes secuencias de forma sencilla.

Aplicación: App_1_3_Grupo_4_2024 MOD

Autor de la modificación: Yapura, Yain

Observaciones: el uso de un driver en esta aplicación con múltiples secuencias te permite crear efectos visuales más complejos y dinámicos. Al combinar el driver con una estructura de control adecuada y una buena organización del código, podrás implementar una amplia variedad de secuencias y personalizarlas según tus necesidades.

Aplicación: App_1_4_Grupo_4_2024 MOD

Autor de la modificación: Romano Sánchez, Marcos Joel

Observaciones: el uso de un driver en esta aplicación te permitirá crear un sistema de iluminación flexible y eficiente, donde puedes controlar la frecuencia de parpadeo de los LED de manera precisa y sencilla**.**

Link al repositorio grupal:

https://github.com/Joelsanchez00/Grupo-4-TDII-2024.git