Introducción a STM32Cube IDE con NUCLEO-F401RE

Tabla de contenido

Objetivos del curso

- Introducir a los estudiantes de 2º de ingeniería informática en el desarrollo con microcontroladores STM32.
- Familiarizar a los alumnos con el entorno STM32Cube IDE y la placa NUCLEO-F401RE.
- Enseñar conceptos básicos de programación de microcontroladores a través de proyectos prácticos.

Sesión 1

---

1. Introducción a los microcontroladores STM32

1.1 Arquitectura ARM Cortex-M

Breve historia de ARM y su importancia en el mercado de microcontroladores

ARM (Advanced RISC Machines) tiene sus orígenes en la década de 1980, cuando Acorn Computers Ltd. desarrolló el primer procesador ARM para sus computadoras personales. En 1990, se formó ARM Holdings como una empresa separada para licenciar la tecnología ARM.

Puntos clave de la historia de ARM:

- 1983: Inicio del proyecto ARM en Acorn Computers.
- 1985: Lanzamiento del primer procesador ARM1.
- 1990: Fundación de ARM Holdings.
- 2002: Introducción de la arquitectura ARMv7, base para los procesadores Cortex.
- 2004: Lanzamiento de la familia Cortex-M para microcontroladores.

La importancia de ARM en el mercado de microcontroladores se debe a varios factores:

1. Eficiencia energética: Los procesadores ARM son conocidos por su bajo consumo de energía, lo que los hace ideales para dispositivos móviles y embebidos.
2. Rendimiento escalable: ARM ofrece una amplia gama de núcleos, desde los más simples hasta los más potentes, permitiendo a los fabricantes elegir el más adecuado para cada aplicación.

3. Modelo de negocio basado en licencias: ARM no fabrica chips, sino que licencia su tecnología a otros fabricantes, lo que ha permitido una amplia adopción y competencia en el mercado.
4. Ecosistema robusto: Existe un gran ecosistema de herramientas de desarrollo, sistemas operativos y software compatibles con ARM.

Actualmente, los procesadores basados en ARM dominan el mercado de dispositivos móviles y tienen una presencia significativa en el mercado de microcontroladores, incluyendo la popular serie STM32 de STMicroelectronics.

Características principales de la arquitectura Cortex-M

La familia Cortex-M está diseñada específicamente para microcontroladores y dispositivos embebidos. Sus principales características incluyen:

Conjunto de instrucciones Thumb-2

Thumb-2 es una extensión del conjunto de instrucciones Thumb original, que combina instrucciones de 16 y 32 bits para ofrecer un equilibrio óptimo entre densidad de código y rendimiento.

Características clave de Thumb-2:

- Mezcla transparente de instrucciones de 16 y 32 bits.
- Mayor densidad de código que ARM de 32 bits puro.
- Mejor rendimiento que Thumb de 16 bits puro.
- Soporte para operaciones de bit y manejo de interrupciones más eficiente.

Modelo de programación simplificado

La arquitectura Cortex-M ofrece un modelo de programación más sencillo en comparación con procesadores ARM más complejos:

- Modos de operación reducidos: principalmente modo Thread y modo Handler.
- Conjunto de registros uniforme: 13 registros de propósito general (R0-R12), Stack Pointer (SP), Link Register (LR), y Program Counter (PC).
- Acceso simplificado a los periféricos mediante un mapa de memoria unificado.
- Sistema de interrupciones vectorizado con prioridades configurables.
- Instrucciones especiales para manejo eficiente de bits (bit-banding).

Soporte para sistemas operativos en tiempo real

La arquitectura Cortex-M está diseñada para facilitar la implementación de sistemas operativos en tiempo real (RTOS):

- Temporizador SysTick: proporciona una base de tiempo precisa para la planificación de tareas.

- Controlador de interrupciones anidadas vectorizadas (NVIC): permite una gestión eficiente de múltiples interrupciones con diferentes prioridades.
- Instrucciones atómicas: facilitan la implementación de mecanismos de sincronización.
- Modos de bajo consumo: permiten una gestión eficiente de la energía en sistemas basados en RTOS.
- Pila dual: permite una separación clara entre el espacio de pila del sistema operativo y el de las aplicaciones.

Estas características hacen que la arquitectura Cortex-M sea ideal para una amplia gama de aplicaciones embebidas, desde dispositivos IoT de bajo consumo hasta sistemas de control industrial más complejos.

1.2 Características principales de la familia STM32F4

La familia STM32F4 es una línea de microcontroladores de alto rendimiento dentro de la gama STM32 de STMicroelectronics. Estas son sus características principales:

Núcleo ARM Cortex-M4 con unidad de punto flotante (FPU)

- **Arquitectura**: ARM Cortex-M4 de 32 bits
- **Unidad de punto flotante (FPU)**:
    - Soporte para operaciones de punto flotante de precisión simple
    - Mejora significativa en el rendimiento para cálculos matemáticos complejos
- **DSP** (Digital Signal Processing):
    - Instrucciones DSP integradas
    - Ideal para aplicaciones de procesamiento de señales

Ventajas:

1. Mayor rendimiento en cálculos matemáticos complejos
2. Eficiencia en aplicaciones de control y procesamiento de señales
3. Código más limpio y eficiente al utilizar operaciones de punto flotante directamente

Frecuencias de reloj de hasta 180 MHz

- Rango de frecuencias:
    - Típicamente desde 84 MHz hasta 180 MHz, dependiendo del modelo específico

- La NUCLEO-F401RE funciona hasta 84 MHz
- **PLL** (Phase-Locked Loop) configurable para generación flexible de reloj
- Múltiples dominios de reloj para optimizar el consumo de energía

Ventajas:

1. Alto rendimiento para aplicaciones exigentes
2. Flexibilidad para ajustar el rendimiento y el consumo de energía
3. Capacidad de ejecutar algoritmos complejos en tiempo real

Memoria Flash y RAM integradas

- **Memoria Flash**:
  - Capacidad: desde 256 KB hasta 2 MB, dependiendo del modelo
  - La NUCLEO-F401RE tiene 512 KB de Flash
  - Programación y borrado rápidos
  - Protección de memoria configurable
- **RAM**:
  - SRAM: desde 64 KB hasta 384 KB
  - La NUCLEO-F401RE tiene 96 KB de SRAM
  - Algunos modelos incluyen CCMRAM (Core Coupled Memory) adicional para mayor rendimiento

Ventajas:

1. Amplio espacio para código de aplicación y datos
2. Ejecución rápida de código desde la memoria Flash
3. Suficiente RAM para aplicaciones complejas y multitarea

Amplia gama de periféricos integrados

Los STM32F4 ofrecen una rica variedad de periféricos, que incluyen:

1. **Comunicación**:
   - USART/UART
   - SPI
   - I2C
   - CAN
   - USB OTG

- Ethernet (en algunos modelos)

2. **Conversión analógica-digital y digital-analógica**:
   - ADCs de 12 bits
   - DACs de 12 bits
3. **Timers**:
   - Timers de propósito general
   - Timers avanzados para control de motores
   - Watchdog timers
4. **DMA** (Direct Memory Access):
   - Múltiples canales para transferencia eficiente de datos
5. **RTC** (Real-Time Clock):
   - Reloj en tiempo real con funciones de calendario
6. **GPIO** (General Purpose Input/Output):
   - Numerosos pines configurables para entrada/salida digital
7. **Interfaces de cámara y LCD** (en algunos modelos)
8. **Criptografía y CRC** (en algunos modelos):
   - Aceleración por hardware para algoritmos de seguridad

Ventajas:

1. Versatilidad para una amplia gama de aplicaciones
2. Reducción de componentes externos necesarios
3. Integración simplificada de múltiples funcionalidades en un solo chip

Conclusión

La familia STM32F4 ofrece un equilibrio excepcional entre alto rendimiento, eficiencia energética y rica integración de periféricos. Estas características la hacen ideal para una amplia gama de aplicaciones, desde dispositivos IoT y wearables hasta equipos industriales y médicos.

1.3 Ventajas de usar microcontroladores en proyectos

Los microcontroladores, como los de la familia STM32F4, ofrecen numerosas ventajas que los hacen ideales para una amplia gama de proyectos de ingeniería. Estas son las principales ventajas:

Bajo consumo de energía

Los microcontroladores están diseñados para ser altamente eficientes en términos de consumo de energía.

Características que contribuyen al bajo consumo:

- Múltiples modos de bajo consumo (sleep, deep sleep, standby)
- Capacidad de apagar periféricos no utilizados
- Voltajes de operación bajos (típicamente 3.3V o menos)
- Tecnología de fabricación optimizada para eficiencia energética

Beneficios:

1. **Mayor duración de la batería** en dispositivos portátiles
2. **Reducción de costos operativos** en sistemas alimentados por red
3. **Posibilidad de usar energía harvesting** en aplicaciones IoT
4. **Menor disipación de calor**, reduciendo la necesidad de sistemas de enfriamiento

Ejemplos de aplicación:

- Sensores inalámbricos alimentados por batería
- Dispositivos médicos implantables
- Wearables y dispositivos IoT

Tamaño reducido

Los microcontroladores integran múltiples componentes en un solo chip, lo que resulta en un tamaño muy compacto.

Factores que contribuyen al tamaño reducido:

- Integración de CPU, memoria y periféricos en un solo chip
- Tecnologías de fabricación avanzadas (28nm, 40nm, etc.)
- Encapsulados optimizados (QFN, BGA, etc.)

Beneficios:

1. **Diseños de productos más compactos**
2. **Reducción del tamaño de las placas de circuito impreso** (PCB)
3. **Menor peso** en aplicaciones donde es crítico (ej. drones, dispositivos portátiles)
4. **Mejor gestión térmica** debido a la menor superficie

Ejemplos de aplicación:

- Audífonos y dispositivos médicos miniatura
- Nodos de sensores para IoT
- Componentes electrónicos para automoción

Flexibilidad y capacidad de personalización

Los microcontroladores ofrecen una gran flexibilidad en su configuración y programación.

Aspectos que contribuyen a la flexibilidad:

- Amplia gama de periféricos integrados
- Pines GPIO configurables
- Soporte para múltiples protocolos de comunicación
- Capacidad de reprogramación en el campo (field-programmable)

Beneficios:

1. **Un mismo microcontrolador puede usarse en múltiples aplicaciones**
2. **Fácil adaptación a cambios en los requisitos del proyecto**
3. **Posibilidad de actualizaciones de firmware** para añadir funcionalidades o corregir errores
4. **Reutilización de diseños** en diferentes productos

Ejemplos de aplicación:

- Plataformas de desarrollo como Arduino o STM32 Nucleo
- Sistemas de control industrial reconfigurables
- Dispositivos de consumo actualizables

Costo-efectividad para producción a escala

Los microcontroladores ofrecen una excelente relación costo-beneficio, especialmente en producción a gran escala.

Factores que contribuyen a la costo-efectividad:

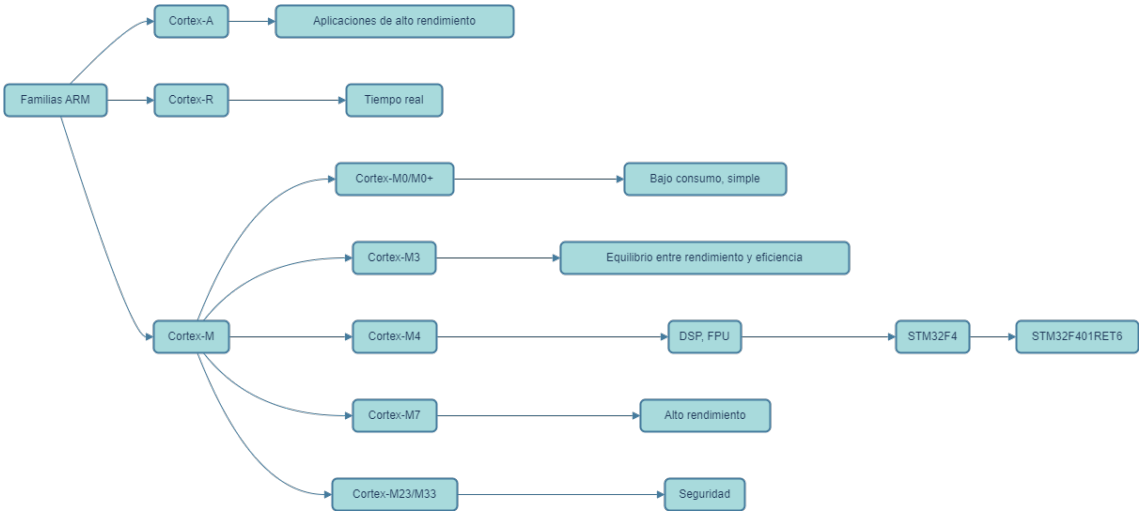
- Integración de múltiples componentes en un solo chip
- Producción en masa que reduce costos unitarios
- Reducción de componentes externos necesarios
- Menor complejidad en el diseño de PCB

Beneficios:

1. **Reducción del costo total de materiales** (BOM - Bill of Materials)
2. **Simplificación del proceso de ensamblaje**
3. **Menor tasa de fallos** debido a la reducción de componentes
4. **Economías de escala** en la producción

Ejemplos de aplicación:

- Electrodomésticos inteligentes
- Sistemas de control automotriz
- Dispositivos de consumo masivo (ej. juguetes electrónicos, relojes digitales)



2. Presentación de la placa NUCLEO-F401RE

2.1 Componentes y características

La placa NUCLEO-F401RE es una plataforma de desarrollo versátil y potente diseñada por STMicroelectronics. Está basada en el microcontrolador STM32F401RET6 y ofrece una serie de características que la hacen ideal para el desarrollo y prototipado de aplicaciones.

Microcontrolador STM32F401RET6

El corazón de la placa NUCLEO-F401RE es el microcontrolador STM32F401RET6, que pertenece a la familia STM32F4 de alto rendimiento.

Características principales:

- Núcleo: ARM Cortex-M4 con FPU (Unidad de Punto Flotante)
- Velocidad de reloj: Hasta 84 MHz

- Memoria Flash: 512 KB
- SRAM: 96 KB
- Encapsulado: LQFP64

Periféricos integrados:

- Timers avanzados y de propósito general
- Interfaces de comunicación: USART, I2C, SPI, USB OTG
- ADC de 12 bits
- RTC (Reloj en tiempo real)
- DMA (Acceso Directo a Memoria)

Programador/depurador ST-LINK/V2-1 integrado

Una de las características más destacadas de la placa NUCLEO-F401RE es su programador/depurador ST-LINK/V2-1 integrado.

Ventajas:

- Permite programar y depurar el microcontrolador sin hardware adicional
- Compatible con múltiples entornos de desarrollo (IDEs)
- Soporta SWD (Serial Wire Debug) para depuración avanzada
- Puede utilizarse para programar otros dispositivos STM32 externos

Funcionalidades:

- Programación de la memoria Flash
- Depuración paso a paso
- Puntos de interrupción y puntos de observación
- Acceso a registros y memoria durante la depuración

Alimentación flexible

La placa NUCLEO-F401RE ofrece múltiples opciones de alimentación, lo que la hace muy versátil para diferentes escenarios de uso.

Opciones de alimentación:

1. **USB**:
    - A través del conector USB mini-B del ST-LINK
    - Proporciona 5V y permite programación/depuración simultánea
2. **Batería externa**:

- Conector para batería de 3.3V
- Ideal para proyectos portátiles o autónomos

3. **Fuente de alimentación externa**:
   - Pines para conectar una fuente externa
   - Acepta voltajes entre 3.3V y 5V
4. **Arduino power**:
   - Puede ser alimentada a través de los pines de alimentación compatibles con Arduino

Características adicionales:

- Regulador de voltaje integrado para proporcionar 3.3V estables al microcontrolador
- Jumpers para seleccionar la fuente de alimentación y medir el consumo de corriente

Pines de expansión compatibles con Arduino

La placa NUCLEO-F401RE incluye headers compatibles con el estándar de pines de Arduino, lo que amplía significativamente sus posibilidades de expansión.

Características:

- Disposición de pines compatible con Arduino Uno R3
- Permite el uso de shields y módulos diseñados para Arduino
- Acceso directo a los pines GPIO del microcontrolador

Ventajas:

1. Reutilización de hardware existente para Arduino
2. Acceso a un amplio ecosistema de módulos y shields
3. Facilita la migración de proyectos de Arduino a STM32

Consideraciones:

- Algunos pines pueden tener funciones diferentes a las de Arduino
- Es importante verificar la compatibilidad de voltaje (3.3V vs 5V) al usar shields de Arduino

Características adicionales

- LED de usuario (LD2) conectado al pin PA5
- Botón de usuario (B1) conectado al pin PC13

- Conector UART para comunicación serie con un PC
- Oscilador de cristal de 8 MHz
- Botón de reset

2.2 Pines y periféricos disponibles

La placa NUCLEO-F401RE, basada en el microcontrolador STM32F401RET6, ofrece una amplia gama de pines y periféricos que la hacen versátil para múltiples aplicaciones. A continuación, se detallan los principales pines y periféricos disponibles:

GPIO (Entradas/Salidas de Propósito General)

Los pines GPIO son fundamentales para la interacción con el mundo exterior, permitiendo controlar y leer señales digitales.

Características principales:

- Total de pines GPIO: hasta 50 (dependiendo de la configuración)
- Voltaje de operación: 3.3V (tolerante a 5V en algunos pines)
- Capacidad de corriente: hasta 25mA por pin
- Resistencias pull-up y pull-down programables por software

Funcionalidades:

1. Configurables como entrada o salida
2. Interrupciones externas en casi todos los pines
3. Velocidad configurable (2MHz, 25MHz, 50MHz, 100MHz)
4. Función alternativa para periféricos (UART, SPI, I2C, etc.)

Ejemplos de uso:

- Control de LEDs
- Lectura de botones y switches
- Interfaz con sensores digitales
- Control de relés y actuadores

Comunicación Serie

La NUCLEO-F401RE ofrece varios protocolos de comunicación serie, esenciales para la interconexión con otros dispositivos y sensores.

UART (Universal Asynchronous Receiver-Transmitter)

- Número de interfaces: 3 (USART1, USART2, USART6)
- Velocidad: hasta 10.5 Mbits/s
- Soporte para comunicación full-duplex
- Control de flujo por hardware (RTS/CTS) en USART1

Usos comunes:

- Comunicación con el PC
- Depuración y logging
- Interfaz con módulos GPS, Bluetooth, etc.

SPI (Serial Peripheral Interface)

- Número de interfaces: 4 (SPI1, SPI2, SPI3, SPI4)
- Modo maestro y esclavo
- Velocidad: hasta 42 Mbits/s
- Tamaño de datos configurable: 8 a 16 bits

Aplicaciones:

- Comunicación con pantallas LCD
- Interfaz con sensores de alta velocidad
- Comunicación con memorias externas

I2C (Inter-Integrated Circuit)

- Número de interfaces: 3 (I2C1, I2C2, I2C3)
- Velocidad: hasta 400 kbit/s en modo rápido
- Soporte para modo maestro y esclavo
- Direccionamiento de 7 y 10 bits

Usos frecuentes:

- Comunicación con sensores de baja velocidad (temperatura, presión, etc.)
- Interfaz con EEPROMs
- Control de dispositivos como relojes en tiempo real (RTC)

ADC (Conversor Analógico-Digital)

El ADC permite medir señales analógicas, convirtiendo voltajes en valores digitales.

Características del ADC en la NUCLEO-F401RE:

- Resolución: 12 bits
- Número de canales: 16
- Velocidad de muestreo: hasta 2.4 MSPS
- Modos de conversión: simple, continuo, scan

Aplicaciones:

- Lectura de sensores analógicos (temperatura, luz, etc.)
- Monitoreo de voltaje de baterías
- Interfaz con potenciómetros para control de usuario

Timers

Los timers son esenciales para tareas que requieren temporización precisa o generación de señales.

Tipos de timers disponibles:

1. Timers de propósito general (TIM2, TIM3, TIM4, TIM5)
    - Resolución: 16/32 bits
    - Funciones: PWM, captura de entrada, comparación de salida
2. Timers avanzados (TIM1)
    - Resolución: 16 bits
    - Características adicionales para control de motores
3. Timers básicos (TIM6, TIM7)
    - Usados principalmente para temporizaciones y trigger de DAC

Aplicaciones comunes:

- Generación de señales PWM para control de motores o LEDs
- Medición de frecuencia y periodo de señales
- Implementación de delays precisos
- Trigger para conversiones ADC

Otros periféricos

- USB OTG FS: Para implementación de dispositivos o host USB
- RTC (Real-Time Clock): Para mantener la hora y fecha
- DMA (Direct Memory Access): Para transferencias de datos eficientes sin intervención de la CPU

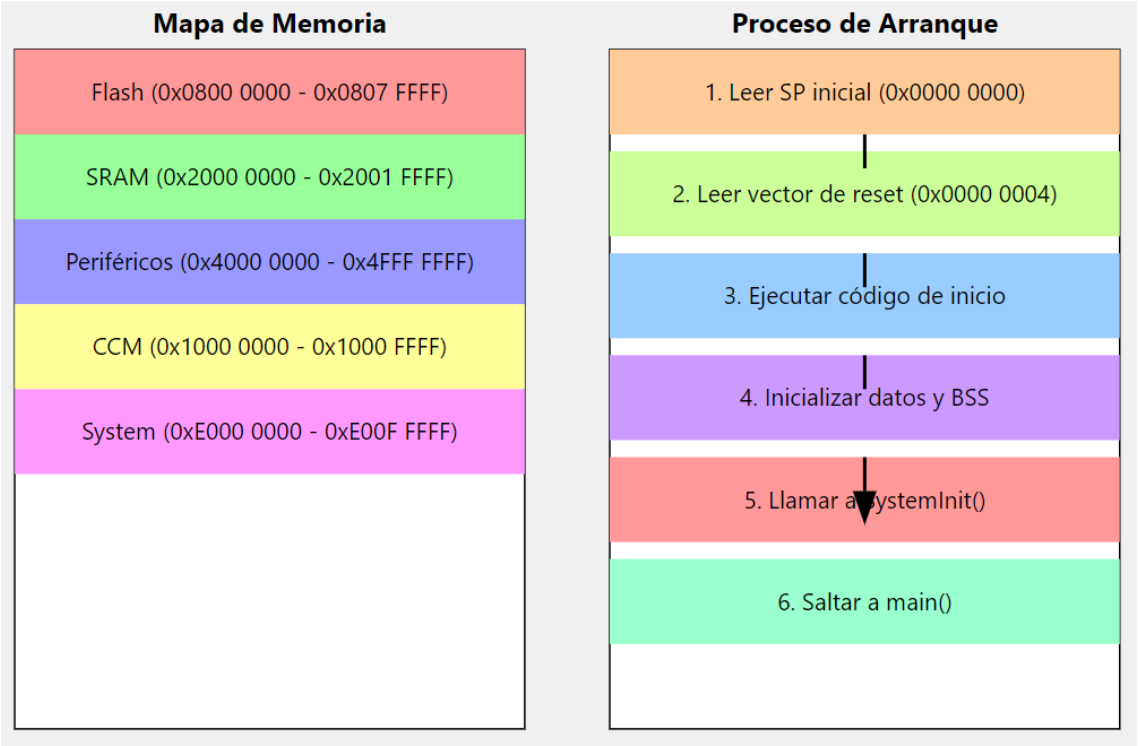
- Watchdog: Para asegurar la recuperación del sistema en caso de fallos

2.3 Mapeado de memoria y proceso de arranque del STM32F401RE

1. Mapa de memoria

El STM32F401RE tiene un espacio de direcciones de 32 bits, lo que permite direccionar hasta 4GB de memoria. Sin embargo, no todo este espacio está físicamente implementado. El mapa de memoria se divide en varias regiones:

```
0x0000 0000 - 0x0007 FFFF : Flash memory (512 KB)
0x1000 0000 - 0x1000 FFFF : CCM (Core Coupled Memory) data RAM (64 KB)
0x2000 0000 - 0x2001 FFFF : SRAM (96 KB)
0x4000 0000 - 0x4FFF FFFF : Peripherals
0x5000 0000 - 0x5FFF FFFF : External RAM
0x6000 0000 - 0x9FFF FFFF : External memory
0xE000 0000 - 0xE00F FFFF : System control space
```



2. Proceso de arranque

Cuando el microcontrolador se enciende o se reinicia, sigue estos pasos:

1. Lee la dirección 0x0000 0000 para obtener el valor inicial del Stack Pointer (SP).
2. Lee la dirección 0x0000 0004 para obtener la dirección del vector de reset (punto de entrada del programa).
3. Salta a la dirección del vector de reset y comienza la ejecución.

3. Opciones de arranque

El STM32F401RE puede arrancar desde diferentes ubicaciones, dependiendo del estado de los pines BOOT0 y BOOT1:

- BOOT0 = 0, BOOT1 = X: Arranca desde la memoria Flash principal
- BOOT0 = 1, BOOT1 = 0: Arranca desde la memoria del sistema (bootloader)
- BOOT0 = 1, BOOT1 = 1: Arranca desde la SRAM interna

4. Código de inicio (Startup)

El código de inicio, generalmente en un archivo llamado startup_stm32f401xe.s, es crucial para la inicialización del microcontrolador. Este código:

1. Define la tabla de vectores de interrupción.
2. Inicializa el Stack Pointer.
3. Configura el reloj del sistema.
4. Inicializa la sección de datos en la RAM.
5. Inicializa la sección BSS (poniéndola a cero).
6. Llama a los constructores de C++ (si se usa).
7. Salta a la función main().

Ejemplo simplificado de una sección del código de inicio:

```
  .section .text.Reset_Handler
  .weak Reset_Handler
  .type Reset_Handler, %function
Reset_Handler:
  ldr   r0, =_estack
  mov   sp, r0          ; Inicializa el Stack Pointer

  ; Inicialización de datos y BSS
  bl    SystemInit      ; Llama a la inicialización del sistema
  bl    __libc_init_array ; Inicializa la biblioteca C
  bl    main            ; Salta a la función main
```

5. Secciones de memoria

El linker script (generalmente STM32F401RETx_FLASH.ld) define cómo se organizan las diferentes secciones de memoria:

- .text: Contiene el código ejecutable. Se almacena en Flash.
- .data: Variables inicializadas. Se almacenan en Flash y se copian a SRAM durante el arranque.
- .bss: Variables no inicializadas. Se reserva espacio en SRAM y se inicializa a cero durante el arranque.
- .stack: Espacio para la pila. Generalmente en la parte superior de la SRAM.

6. Gestión de la memoria Flash

La memoria Flash se organiza en sectores de diferentes tamaños:

- Sector 0: 16 KB
- Sector 1: 16 KB
- Sector 2: 16 KB
- Sector 3: 16 KB
- Sector 4: 64 KB
- Sector 5: 128 KB
- Sector 6: 128 KB
- Sector 7: 128 KB

Esta organización es importante para las operaciones de borrado y escritura de la Flash.

7. Acceso a la memoria

El STM32F401RE utiliza la arquitectura Harvard modificada, lo que significa que tiene buses separados para instrucciones y datos. Esto permite accesos simultáneos a la memoria de programa y de datos, mejorando el rendimiento.

8. Optimización del rendimiento

- La CCM (Core Coupled Memory) es una RAM de alta velocidad directamente acoplada al núcleo. Es ideal para datos críticos en términos de rendimiento.
- El controlador de DMA (Direct Memory Access) permite transferencias de datos entre memoria y periféricos sin intervención de la CPU.

9. Protección de memoria

El STM32F401RE ofrece varias características de protección de memoria:

- MPU (Memory Protection Unit): Permite definir regiones de memoria con diferentes permisos de acceso.
- Readout Protection: Previene la lectura no autorizada de la memoria Flash.
- Write Protection: Protege sectores de la Flash contra escrituras o borrados accidentales.

2.4 Acceso a periféricos en el STM32F401RE

En el STM32F401RE, como en muchos microcontroladores modernos, los periféricos se acceden mediante un mecanismo conocido como "memory-mapped I/O" (E/S mapeada en memoria). Esto significa que los registros de control y estado de los periféricos se tratan como si fueran ubicaciones de memoria.

Conceptos clave

1. **Mapeado de memoria**: Cada periférico tiene una dirección base asignada en el espacio de memoria.
2. **Registros de periféricos**: Son tratados como ubicaciones de memoria, pero controlan o reflejan el estado del hardware.
3. **Buses de sistema**: AHB (Advanced High-performance Bus) y APB (Advanced Peripheral Bus) conectan el núcleo del procesador con los periféricos.
4. **Relojes de periféricos**: Cada periférico tiene su propio reloj que debe ser habilitado antes de su uso.

Proceso de acceso a periféricos

1. **Habilitación del reloj**: Antes de usar un periférico, su reloj debe ser activado.
2. **Configuración**: Se escriben valores específicos en los registros de control del periférico.
3. **Operación**: Se leen o escriben los registros de datos o estado del periférico.
4. **Interrupciones**: Muchos periféricos pueden generar interrupciones para notificar eventos.

Ejemplo: Configuración de un pin GPIO

1. Habilitar el reloj del puerto GPIO (por ejemplo, GPIOA):

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
```

2. Configurar el modo del pin (por ejemplo, como salida):

```
GPIOA->MODER &= ~GPIO_MODER_MODER5_Msk;
GPIOA->MODER |= GPIO_MODER_MODER5_0;
```

3. Establecer el estado del pin:

```
GPIOA->BSRR = GPIO_BSRR_BS5; // Establece el pin 5 en alto
```

Uso de la biblioteca HAL

Aunque es posible acceder directamente a los registros de los periféricos, STMicroelectronics proporciona la biblioteca HAL (Hardware Abstraction Layer) que simplifica el proceso:

```
// Inicialización de GPIO usando HAL
GPIO_InitTypeDef GPIO_InitStruct = {0};
GPIO_InitStruct.Pin = GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

// Control de GPIO usando HAL
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
```

Este enfoque abstrae los detalles de bajo nivel y proporciona una interfaz más portable entre diferentes microcontroladores STM32.

2.5 Enfoque en LED y botón incorporados

LED verde de usuario (LD2) y Botón de usuario (B1) en la placa NUCLEO-F401RE

La placa NUCLEO-F401RE viene equipada con un LED y un botón de usuario que son excelentes para comenzar a experimentar con entradas y salidas digitales. Estos componentes están directamente conectados a pines específicos del microcontrolador STM32F401RET6.

LED verde de usuario (LD2)

El LED verde de usuario, etiquetado como LD2 en la placa, es un componente de salida simple pero muy útil.

Características:

- **Color**: Verde
- **Conexión**: Conectado al pin PA5 del microcontrolador
- **Configuración**: Conectado en configuración de ánodo común (se enciende cuando el pin está en estado alto)

Detalles técnicos:

- Voltaje de operación: 3.3V (el voltaje de operación del microcontrolador)
- Corriente típica: Alrededor de 7mA cuando está completamente encendido

Usos comunes:

1. Indicador de estado del programa
2. Depuración visual de código
3. Señalización de eventos o condiciones específicas
4. Primeras prácticas de programación de GPIO

Ejemplo de código para controlar LD2:

```
// Configuración del pin PA5 como salida
GPIO_InitTypeDef GPIO_InitStruct = {0};
GPIO_InitStruct.Pin = GPIO_PIN_5;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

// Encender el LED
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);

// Apagar el LED
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
```

Botón de usuario (B1)

El botón de usuario, etiquetado como B1 en la placa, proporciona una entrada digital simple pero efectiva.

Características:

- **Tipo**: Pulsador momentáneo
- **Conexión**: Conectado al pin PC13 del microcontrolador
- **Configuración**: Conectado a tierra cuando se presiona (lógica inversa)

Detalles técnicos:

- Estado en reposo: Alto (3.3V) debido a una resistencia de pull-up interna
- Estado cuando se presiona: Bajo (0V)
- Debouncing: No tiene hardware de debouncing, debe manejarse por software si es necesario

Usos comunes:

1. Entrada de usuario para cambiar modos o estados del programa
2. Activar o desactivar funciones específicas
3. Generar interrupciones para eventos controlados por el usuario
4. Prácticas de manejo de entradas digitales y detección de flancos

Ejemplo de código para leer B1:

```
// Configuración del pin PC13 como entrada
GPIO_InitTypeDef GPIO_InitStruct = {0};
GPIO_InitStruct.Pin = GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

// Leer el estado del botón
GPIO_PinState buttonState = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13);
if (buttonState == GPIO_PIN_RESET) {
    // El botón está presionado
    // Realizar acción deseada
}
```

Consideraciones adicionales

- **Debouncing**: El botón B1 puede necesitar debouncing por software para evitar lecturas falsas debido al rebote mecánico.
- **Interrupciones**: Para una respuesta más eficiente, se puede configurar una interrupción en el pin PC13 para detectar las pulsaciones del botón.
- **Modos de bajo consumo**: Tanto el LED como el botón pueden usarse en conjunto con los modos de bajo consumo del microcontrolador para prácticas de gestión de energía.

3. Introducción al STM32Cube IDE

3.1 Instalación y configuración inicial

1. Descarga desde la página oficial de STMicroelectronics

1. Abre tu navegador web y ve a la página oficial de STMicroelectronics: https://www.st.com/
2. En la barra de búsqueda, escribe "STM32CubeIDE" y presiona Enter.
3. Haz clic en el resultado que dice "STM32CubeIDE - Integrated Development Environment for STM32".
4. En la nueva página, desplázate hacia abajo hasta la sección "Get Software".
5. Haz clic en el botón "Get Software" junto a la versión más reciente de STM32CubeIDE.
6. Si no tienes una cuenta, se te pedirá que crees una. Sigue los pasos para crear tu cuenta de STMicroelectronics.
7. Una vez que hayas iniciado sesión, acepta el acuerdo de licencia y selecciona la versión apropiada para tu sistema operativo (Windows, macOS o Linux).
8. Haz clic en "Download" para iniciar la descarga del instalador.

2. Proceso de instalación paso a paso

Para Windows:

1. Una vez completada la descarga, localiza el archivo de instalación (normalmente en la carpeta "Descargas").
2. Haz doble clic en el archivo .exe para iniciar el instalador.

3. Si aparece una advertencia de seguridad de Windows, haz clic en "Sí" para permitir que el programa realice cambios en tu dispositivo.
4. En la ventana de bienvenida del instalador, haz clic en "Next".
5. Lee y acepta el acuerdo de licencia, luego haz clic en "Next".
6. Elige la ubicación de instalación o deja la ubicación predeterminada. Haz clic en "Next".
7. En la pantalla de selección de componentes, asegúrate de que todos los componentes estén seleccionados y haz clic en "Next".
8. Haz clic en "Install" para comenzar la instalación.
9. El proceso de instalación puede tardar varios minutos. Espera a que se complete.
10. Una vez finalizada la instalación, marca la casilla "Launch STM32CubeIDE" si deseas iniciar el IDE inmediatamente.
11. Haz clic en "Finish" para completar la instalación.

Para macOS:

1. Localiza el archivo .dmg descargado y haz doble clic para montarlo.
2. Arrastra el icono de STM32CubeIDE a la carpeta de Aplicaciones.
3. La primera vez que ejecutes STM32CubeIDE, es posible que macOS te advierta sobre una aplicación descargada de Internet. Haz clic en "Abrir" para permitir la ejecución.

Para Linux:

1. Abre una terminal y navega hasta el directorio donde se descargó el archivo de instalación.
2. Haz el archivo ejecutable con el comando: chmod +x nombre_del_archivo.sh
3. Ejecuta el instalador con: ./nombre_del_archivo.sh
4. Sigue las instrucciones en pantalla para completar la instalación.

3. Configuración inicial del entorno

1. Inicia STM32CubeIDE.
2. En la primera ejecución, se te pedirá que selecciones un directorio de trabajo (workspace). Elige una ubicación conveniente o acepta la predeterminada.
3. Una vez que se abra el IDE, verás una pantalla de bienvenida. Puedes explorar los recursos disponibles o cerrarla.

4. Ve a "Help" > "Check for Updates" para asegurarte de que tienes la versión más reciente del software y sus componentes.
5. Configura el compilador y las herramientas:
    - Ve a "Window" > "Preferences"
    - En el panel izquierdo, navega a "MCU" > "Global MCU Settings"
    - Verifica que las rutas al compilador ARM GCC y a las herramientas de construcción estén correctamente configuradas.
6. Configura el depurador ST-LINK:
    - En las preferencias, ve a "MCU" > "ST-LINK"
    - Asegúrate de que la opción "ST-LINK GDB server" esté seleccionada.
7. (Opcional) Personaliza el entorno:
    - Ajusta el tema de color: "Window" > "Preferences" > "General" > "Appearance"
    - Configura los atajos de teclado: "Window" > "Preferences" > "General" > "Keys"
8. Familiarízate con la interfaz:
    - Explorador de proyectos (izquierda)
    - Editor de código (centro)
    - Consola y ventanas de salida (abajo)
    - Perspectivas de depuración (arriba derecha)
9. Crea un nuevo proyecto para probar la instalación:
    - File > New > STM32 Project
    - Selecciona tu placa (NUCLEO-F401RE) o tu microcontrolador específico
    - Sigue el asistente para crear un proyecto básico
10. Compila el proyecto de prueba para verificar que todo esté funcionando correctamente.

Consejos adicionales

- Asegúrate de tener instalados los drivers ST-LINK para tu sistema operativo.
- Si encuentras problemas, consulta la documentación oficial o los foros de STMicroelectronics.

- Considera instalar alguna extensión de STM32CubeMX si planeas utilizarlo para la configuración gráfica de periféricos.

3.2 Visión general de la interfaz de usuario

Perspectivas y vistas principales

STM32CubeIDE utiliza el concepto de "perspectivas" y "vistas" para organizar su interfaz de usuario. Una perspectiva es un conjunto predefinido de vistas que están organizadas de una manera específica para facilitar un tipo particular de tarea.

Perspectivas principales:

1. **C/C++**: Esta es la perspectiva predeterminada para el desarrollo de código.
2. **Debug**: Se utiliza cuando estás depurando tu aplicación.
3. **CubeMX**: Se usa cuando estás configurando tu microcontrolador con la herramienta gráfica CubeMX.

Para cambiar entre perspectivas:

- Haz clic en el icono de perspectiva en la esquina superior derecha de la ventana.
- O ve a Window > Perspective > Open Perspective

Vistas comunes:

- Project Explorer
- Outline
- Problems
- Console
- Memory
- Registers
- SFR (Special Function Registers)

Para abrir una vista específica:

- Ve a Window > Show View y selecciona la vista deseada.

Explorador de proyectos

El Explorador de proyectos (Project Explorer) es una vista crucial que te permite navegar por la estructura de archivos de tus proyectos.

Características principales:

1. **Estructura de árbol**: Muestra tus proyectos y sus archivos en una estructura jerárquica.
2. **Filtros**: Puedes filtrar los tipos de archivos que se muestran.
3. **Acciones contextuales**: Haz clic derecho en cualquier elemento para acceder a acciones específicas.
4. **Organización de proyectos STM32**:
    - Core: Contiene el código fuente principal y los archivos de configuración.
    - Drivers: Incluye los drivers HAL y CMSIS.
    - Debug y Release: Carpetas de salida para las diferentes configuraciones de compilación.

Consejos:

- Utiliza la vista "Outline" junto con el explorador de proyectos para navegar rápidamente por la estructura de tus archivos de código.
- Puedes arrastrar y soltar archivos para reorganizarlos.

Editor de código

El editor de código es donde pasarás la mayor parte de tu tiempo escribiendo y editando código.

Características principales:

1. **Resaltado de sintaxis**: Colorea el código para mejorar la legibilidad.
2. **Autocompletado**: Sugiere completar funciones, variables y estructuras mientras escribes.
3. **Indicadores de error y advertencia**: Muestra líneas onduladas rojas (errores) o amarillas (advertencias) en el código.
4. **Plegado de código**: Permite colapsar y expandir bloques de código.
5. **Navegación rápida**: Ctrl+clic (o Cmd+clic en Mac) en un símbolo para ir a su definición.
6. **Múltiples pestañas**: Permite tener varios archivos abiertos simultáneamente.

Consejos:

- Utiliza Ctrl+Space para activar el autocompletado manualmente.
- Hover sobre variables o funciones para ver información adicional.
- Utiliza Ctrl+Shift+F para formatear tu código automáticamente.

Consola y ventana de depuración

Estas vistas son esenciales para monitorear la salida de tu programa y para la depuración.

Consola:

1. **Build console**: Muestra la salida del proceso de compilación.
2. **Error Log**: Muestra errores del IDE y otras informaciones de diagnóstico.
3. **STM32CubeIDE Console**: Proporciona información sobre las operaciones del IDE.

Ventanas de depuración:

1. **Debug**: Muestra la pila de llamadas y permite controlar la ejecución del programa.
2. **Variables**: Muestra el valor de las variables durante la depuración.
3. **Breakpoints**: Lista todos los puntos de interrupción establecidos.
4. **Registers**: Muestra el contenido de los registros del microcontrolador.
5. **Memory**: Permite examinar el contenido de la memoria.
6. **Disassembly**: Muestra el código desensamblado.

Consejos para depuración:

- Utiliza F5 para iniciar la depuración, F6 para step over, F5 para step into.
- Establece puntos de interrupción haciendo doble clic en el margen izquierdo del editor.
- Utiliza la vista "Expressions" para evaluar expresiones complejas durante la depuración.

Configuración y personalización

STM32CubeIDE es altamente personalizable:

1. Ve a Window > Preferences para acceder a las opciones de configuración.
2. Puedes personalizar atajos de teclado, colores y fuentes, y muchas otras opciones.
3. Considera instalar plugins adicionales para extender la funcionalidad del IDE.

3.3 Explicación de las herramientas principales

1. STM32CubeMX para configuración gráfica de pines y periféricos

STM32CubeMX es una herramienta gráfica integrada en STM32CubeIDE que permite configurar fácilmente los microcontroladores STM32 y generar el código de inicialización correspondiente.

Características principales:

- **Selección de microcontrolador**: Permite elegir el microcontrolador específico o la placa de desarrollo.
- **Configuración de pines**: Interfaz gráfica para asignar funciones a los pines del microcontrolador.
- **Configuración de periféricos**: Permite configurar los periféricos internos como UART, SPI, I2C, Timers, etc.
- **Configuración del reloj**: Herramienta visual para configurar el árbol de reloj del microcontrolador.
- **Generación de código**: Genera automáticamente el código de inicialización basado en la configuración.

Cómo usar STM32CubeMX:

1. Al crear un nuevo proyecto, selecciona "STM32 Project" y elige tu placa o microcontrolador.
2. En la vista de configuración de pines, haz clic en los pines para asignar funciones.
3. Utiliza las pestañas laterales para configurar periféricos, reloj, y otras opciones.
4. Haz clic en "Generate Code" para crear el código de inicialización.

Consejos:

- Utiliza la función de resolución de conflictos para detectar y resolver problemas de configuración.
- Explora las diferentes pestañas para familiarizarte con todas las opciones de configuración disponibles.

2. Compilador integrado

STM32CubeIDE incluye un compilador GCC (GNU Compiler Collection) optimizado para microcontroladores ARM.

Características principales:

- **Soporte para C y C++**: Permite desarrollar en ambos lenguajes.

- **Optimizaciones**: Ofrece diferentes niveles de optimización (-O0, -O1, -O2, -O3, -Os).
- **Opciones de depuración**: Genera información de depuración para uso con GDB.
- **Integración con el IDE**: Configuración sencilla a través de las propiedades del proyecto.

Cómo usar el compilador:

1. La compilación se inicia automáticamente al guardar cambios (si está configurado así).
2. Puedes compilar manualmente con el botón "Build" o presionando Ctrl+B (Cmd+B en Mac).
3. Los errores y advertencias aparecerán en la vista "Problems".

Configuración del compilador:

1. Haz clic derecho en el proyecto > Properties.
2. Navega a C/C++ Build > Settings.
3. Aquí puedes ajustar las opciones del compilador, como las banderas de optimización.

Consejos:

- Utiliza diferentes niveles de optimización para equilibrar el tamaño del código y el rendimiento.
- Habilita las advertencias del compilador para detectar posibles problemas en el código.

3. Herramientas de depuración

STM32CubeIDE proporciona un conjunto completo de herramientas de depuración para ayudarte a encontrar y corregir errores en tu código.

Características principales:

- **Depurador GDB**: Permite depuración a nivel de fuente.
- **Puntos de interrupción**: Puedes establecer breakpoints en el código.
- **Ejecución paso a paso**: Permite ejecutar el código línea por línea.
- **Visualización de variables**: Muestra el valor de las variables durante la ejecución.
- **Visualización de registros**: Permite ver y modificar los registros del microcontrolador.

- **Visualización de memoria**: Permite examinar y modificar la memoria del dispositivo.

Cómo usar las herramientas de depuración:

1. Establece puntos de interrupción haciendo doble clic en el margen izquierdo del editor.
2. Inicia la depuración con el botón "Debug" o presionando F11.
3. Utiliza los botones de control de depuración para navegar por el código (Step Over, Step Into, etc.).
4. Observa los valores de las variables en la vista "Variables" o añade expresiones de observación.

Consejos:

- Utiliza puntos de interrupción condicionales para situaciones más complejas.
- Familiarízate con los atajos de teclado para una depuración más eficiente.

4. Utilidad de programación de la placa

STM32CubeIDE incluye herramientas para programar (flashear) tu microcontrolador STM32 directamente desde el IDE.

Características principales:

- **Soporte para diversos programadores**: ST-LINK, J-LINK, etc.
- **Programación de la memoria Flash**: Carga tu programa en la memoria del microcontrolador.
- **Verificación**: Comprueba que el programa se ha cargado correctamente.
- **Opciones de borrado**: Permite borrar la memoria Flash antes de programar.

Cómo usar la utilidad de programación:

1. Conecta tu placa de desarrollo al ordenador.
2. Compila tu proyecto.
3. Haz clic en el botón "Run" (o presiona F11) para compilar, cargar y comenzar la depuración.
4. Alternativamente, usa el botón "Program" para solo cargar el programa sin iniciar la depuración.

Configuración de la programación:

1. Haz clic derecho en el proyecto > Properties.
2. Navega a C/C++ Build > Settings > Tool Settings > MCU Programming.

3. Aquí puedes configurar opciones como el programador a utilizar, opciones de borrado, etc.

Consejos:

- Asegúrate de que los drivers de tu programador (por ejemplo, ST-LINK) estén correctamente instalados.
- Si tienes problemas de conexión, verifica el cable y prueba diferentes puertos USB.

4. Creación de un proyecto básico: LED parpadeante

4.1 Configuración del proyecto para la NUCLEO-F401RE

1. Creación de un nuevo proyecto en STM32CubeIDE

1. Abre STM32CubeIDE.
2. En la barra de menú, Selecciona File > New > Project..., luego elige STM32 Project.
3. En la ventana "Target Selector", verás varias opciones para seleccionar tu microcontrolador o placa:
    - MCU/MPU Selector
    - Board Selector
    - MCU Selector
4. Dado que estamos usando la placa NUCLEO-F401RE, selecciona la pestaña "Board".
5. En el campo de búsqueda, escribe "NUCLEO-F401RE".
6. Selecciona la placa "NUCLEO-F401RE" de la lista de resultados.
7. Haz clic en "Next".

2. Selección de la placa NUCLEO-F401RE

8. En la siguiente pantalla, verás los detalles de la placa seleccionada y las opciones de proyecto:
    - Project Name: Dale un nombre descriptivo a tu proyecto.
    - Location: Elige dónde quieres guardar tu proyecto o deja la ubicación por defecto.
    - Targeted Language: Selecciona C++.

- Targeted Binary Type: Normalmente, "Executable" para un programa completo.
- Targeted Project Type: "STM32Cube" es la opción recomendada para usar todas las características de STM32CubeIDE.

9. Haz clic en "Finish".
10. Si aparece un diálogo preguntando si quieres cambiar a la perspectiva asociada, selecciona "Yes".

3. Configuración básica del reloj del sistema a 16 MHz

1. **Abrir la vista de configuración del reloj**:
   - Después de crear el proyecto, se abrirá automáticamente la vista de **STM32CubeMX**.
   - En el panel izquierdo, haz clic en **Clock Configuration**.
2. **Configurar la fuente del reloj**:
   - En el diagrama del árbol de reloj del microcontrolador, localiza la sección **PLL Source Mux**.
   - **Selecciona "HSI"** (High Speed Internal clock) como fuente del PLL.
3. **Configurar la frecuencia del HCLK**:
   - En lugar de configurar el HCLK a 84 MHz, lo vamos a configurar a **16 MHz**.
   - Introduce **16** en la casilla de **HCLK (MHz)**, el sistema ajustará automáticamente algunos valores para alcanzar esta frecuencia.
4. **Ajuste de los divisores del PLL**:
   - A continuación, configura los divisores y multiplicadores del PLL:
     - **PLL Source**: HSI
     - **PLLM**: 8 (divisor)
     - **PLLN**: 64 (multiplicador)
     - **PLLP**: 2 (divisor)
     - **PLLQ**: 7 (para las interfaces USB o I2S, si es necesario)

Estos valores deberían resultar en una frecuencia de **16 MHz** para el reloj del sistema.

5. **Verificación**:
   - Asegúrate de que todos los valores en el diagrama del reloj están en negro o azul. Si hay algún valor en rojo, deberás ajustar los divisores **/M**, **/N**, **/P** o **/Q** hasta que la configuración sea válida.

Generar el Código:

Una vez que hayas configurado correctamente el reloj, haz clic en **Project > Generate Code** para que **STM32CubeIDE** genere el código de inicialización de la MCU.

4. Exploración del proyecto generado

Después de generar el código, verás la estructura de tu proyecto en el "Project Explorer":

- Core: Contiene los archivos principales de tu proyecto, incluyendo main.c.
- Drivers: Contiene los drivers HAL y CMSIS necesarios.
- Core/Inc y Core/Src: Contienen los archivos de cabecera y fuente respectivamente.

El archivo main.c ya incluirá la inicialización básica del sistema y un bucle infinito donde puedes comenzar a añadir tu código.

5. Compilación y carga del proyecto

16. Para compilar el proyecto, Selecciona **Project > Build All**.
17. Si la compilación es exitosa, puedes cargar el programa en tu placa NUCLEO-F401RE haciendo clic en el icono "Run" (triángulo verde) o seleccionando Run > Run.

Consejos adicionales

- Siempre que hagas cambios en la configuración a través de STM32CubeMX, asegúrate de regenerar el código.
- Explora las diferentes pestañas en la vista de configuración para familiarizarte con las opciones disponibles.
- Si encuentras errores, revisa la consola y la vista de problemas para obtener más información.

4.2 Explicación de la estructura del proyecto

Cuando creas un nuevo proyecto en STM32CubeIDE, se genera automáticamente una estructura de directorios y archivos. Comprender esta estructura es crucial para navegar eficientemente por tu proyecto y entender dónde añadir tu propio código.

1. Archivos generados automáticamente

STM32CubeIDE genera varios archivos automáticamente basados en tu configuración inicial. Algunos de los más importantes son:

1. **main.c**: Este archivo contiene la función main() y es donde comenzarás a escribir tu código principal.
2. **stm32f4xx_hal_msp.c**: Contiene las funciones de inicialización de bajo nivel para los periféricos HAL (Hardware Abstraction Layer).
3. **stm32f4xx_it.c**: Aquí se definen los manejadores de interrupciones.
4. **syscalls.c**: Implementa las llamadas al sistema necesarias para ciertas funciones de la biblioteca estándar de C.
5. **sysmem.c**: Implementa funciones relacionadas con la gestión de memoria dinámica.
6. **system_stm32f4xx.c**: Contiene la inicialización del sistema y la configuración del reloj.

2. Estructura de directorios

La estructura típica de un proyecto STM32CubeIDE se organiza de la siguiente manera:

```
NombreDelProyecto/
|
├─── Core/
|    ├─── Inc/
|    |    ├─── main.h
|    |    ├─── stm32f4xx_hal_conf.h
|    |    ├─── stm32f4xx_it.h
|    |    └─── ...
|    |
|    └─── Src/
|         ├─── main.c
|         ├─── stm32f4xx_hal_msp.c
|         ├─── stm32f4xx_it.c
|         ├─── syscalls.c
|         ├─── sysmem.c
|         └─── system_stm32f4xx.c
|
├─── Drivers/
|    ├─── CMSIS/
|    |    └─── ...
|    |
|    └─── STM32F4xx_HAL_Driver/
|         ├─── Inc/
|         └─── Src/
|
├─── Debug/
|
└─── Release/
```

Descripción de los directorios principales:

- **Core**: Contiene el código específico de tu aplicación.
    - **Inc**: Archivos de cabecera (.h)

    - **Src**: Archivos de código fuente (.c)
- **Drivers**: Contiene los drivers necesarios para el microcontrolador.
    - **CMSIS**: Core Microcontroller Software Interface Standard, proporciona una capa de abstracción para los núcleos Cortex-M.
    - **STM32F4xx_HAL_Driver**: Hardware Abstraction Layer para la familia STM32F4.
- **Debug/Release**: Directorios de salida para las diferentes configuraciones de compilación.

3. Archivos de configuración y Makefile

STM32CubeIDE utiliza varios archivos de configuración y un Makefile para gestionar la compilación y enlazado del proyecto.

Archivos de configuración clave:

1. **stm32f4xx_hal_conf.h**:
    - Ubicación: Core/Inc/
    - Propósito: Configura qué módulos HAL se incluirán en el proyecto.
2. **.cproject**:
    - Ubicación: Raíz del proyecto
    - Propósito: Archivo de configuración XML para el proyecto C/C++. Contiene configuraciones de compilación, rutas de inclusión, etc.
3. **.project**:
    - Ubicación: Raíz del proyecto
    - Propósito: Archivo de configuración XML general del proyecto Eclipse.
4. **STM32F401RETx_FLASH.ld**:
    - Ubicación: Raíz del proyecto
    - Propósito: Script de enlazado que define la disposición de memoria para el microcontrolador.

Makefile

Aunque STM32CubeIDE utiliza internamente un Makefile para la compilación, este no es visible directamente en la estructura del proyecto. En su lugar, la configuración de compilación se gestiona a través de la interfaz gráfica del IDE:

1. Haz clic derecho en el proyecto.
2. Selecciona "Properties".

3. Navega a "C/C++ Build" > "Settings".

Aquí puedes configurar:

- Opciones del compilador
- Opciones del enlazador
- Rutas de inclusión
- Símbolos de preprocesador
- Y más...

Estas configuraciones se traducen internamente en reglas de Makefile cuando se compila el proyecto.

Consejos para trabajar con la estructura del proyecto

1. **Añadir nuevo código**: Coloca tus nuevos archivos .c en Core/Src/ y los .h correspondientes en Core/Inc/.
2. **Modificar la configuración HAL**: Edita stm32f4xx_hal_conf.h para habilitar o deshabilitar módulos HAL según sea necesario.
3. **Cambiar la configuración de memoria**: Si necesitas modificar la disposición de memoria, edita el archivo STM32F401RETx_FLASH.ld.
4. **Añadir librerías externas**: Crea nuevas carpetas en la raíz del proyecto para tus librerías y añade las rutas de inclusión en las propiedades del proyecto.
5. **Regenerar código**: Después de modificar la configuración en CubeMX, siempre regenera el código para asegurar que todos los archivos estén actualizados.

4.3 Programación de un LED parpadeante (Blink)

En este ejercicio, crearemos un programa simple que hará parpadear el LED incorporado en la placa NUCLEO-F401RE, que está conectado al pin PA5.

1. Configuración del pin PA5 como salida

Primero, necesitamos configurar el pin PA5 como una salida digital.

1. Abre tu proyecto en STM32CubeIDE.
2. En el panel de la izquierda, haz clic en el icono de tu microcontrolador para abrir la vista de configuración de pines.
3. Busca el pin PA5 en el diagrama o en la lista de pines.
4. Haz clic en PA5 y en el menú desplegable, selecciona "GPIO_Output".
5. En el panel de configuración de GPIO (abajo a la izquierda), puedes dejar la configuración por defecto:

- GPIO output level: Low
- GPIO mode: Output Push Pull
- GPIO Pull-up/Pull-down: No pull-up and no pull-down
- Maximum output speed: Low
- User Label: Puedes dejarlo vacío o poner "LD2" (el nombre del LED en la placa)

6. Haz clic en Project > Generate Code en la barra de herramientas superior para actualizar el código con esta configuración.

2. Escritura del código para alternar el estado del LED

Ahora, escribiremos el código para alternar el estado del LED.

1. Abre el archivo main.c en el directorio Core/Src.
2. Busca la función main(). Debería verse algo así:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN 3 */
  while (1)
  {
    /* USER CODE END 3 */
  }
}
Dentro del bucle infinito while(1), añade el código para alternar el estado del LED:
c
Copy
/* Infinite loop */
/* USER CODE BEGIN 3 */
while (1)
{
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  HAL_Delay(500);  // Espera 500 ms
}
/* USER CODE END 3 */
```

3. Uso de la función de retardo para crear el parpadeo

En el código anterior, ya hemos utilizado la función HAL_Delay() para crear el efecto de parpadeo. Esta función introduce un retraso en milisegundos.

```
HAL_Delay(500); crea un retraso de 500 ms, o medio segundo.
Puedes ajustar este valor para cambiar la velocidad de parpadeo. Por ejemplo,
HAL_Delay(1000); hará que el LED parpadee una vez por segundo.
```

4. Compilación y carga del programa en la placa

Ahora que hemos escrito nuestro código, es hora de compilarlo y cargarlo en la placa.

1. **Compilar el programa**:
    - Haz clic en el icono de martillo en la barra de herramientas, o
    - Ve a Project > Build All
2. Observa la consola en la parte inferior de la IDE. Deberías ver mensajes indicando que la compilación fue exitosa.
3. **Cargar el programa**:
    - Asegúrate de que tu placa NUCLEO-F401RE esté conectada a tu computadora vía USB.
    - Haz clic en el icono de "Run" (un botón de play verde) en la barra de herramientas, o
    - Ve a Run > Run
4. Si es la primera vez que cargas un programa, es posible que se te pida configurar una configuración de depuración:
    - Selecciona "STM32 Cortex-M C/C++ Application"
    - Haz clic en "Search Project" para encontrar el archivo ejecutable
    - Haz clic en "OK" para crear la configuración
5. El programa se cargará en la placa y comenzará a ejecutarse inmediatamente.
6. Deberías ver el LED en tu placa NUCLEO-F401RE parpadeando cada medio segundo.

Solución de problemas comunes

- Si el LED no parpadea, verifica que hayas seleccionado el pin correcto (PA5) en la configuración.
- Asegúrate de que la placa esté correctamente conectada y reconocida por tu computadora.

- Verifica que no haya errores de compilación en la consola.
- Si el LED parpadea muy rápido o muy lento, ajusta el valor en HAL_Delay().

Sesión 2

1. Repaso rápido de la sesión anterior

Recapitulación de los conceptos clave vistos en la Sesión 1

1. Introducción a los microcontroladores STM32

- Arquitectura ARM Cortex-M
- Características principales de la familia STM32F4
- Ventajas de usar microcontroladores en proyectos

2. Placa NUCLEO-F401RE

- Componentes principales: Microcontrolador STM32F401RET6, ST-LINK/V2-1, LED y botón de usuario
- Pines y periféricos disponibles
- Opciones de alimentación

3. Entorno de desarrollo STM32CubeIDE

- Instalación y configuración inicial
- Interfaz de usuario: perspectivas, vistas, editor de código
- Herramientas principales: STM32CubeMX, compilador, depurador

4. Creación de un proyecto básico

- Proceso de creación de un nuevo proyecto
- Configuración del reloj del sistema
- Estructura de archivos y directorios del proyecto

5. Programación básica: LED parpadeante

- Configuración de GPIO
- Escritura de código para controlar el LED
- Uso de funciones HAL (Hardware Abstraction Layer)
- Compilación y carga del programa en la placa

Resolución de dudas surgidas entre sesiones

1. ¿Por qué usamos la familia STM32F4 en particular?

La familia STM32F4 ofrece un buen equilibrio entre rendimiento y facilidad de uso. Tiene suficiente potencia para aplicaciones complejas, pero no es tan abrumadora como algunas de las series más avanzadas. Además, la placa NUCLEO-F401RE es asequible y ampliamente disponible, lo que la hace ideal para aprendizaje y prototipado.

2. ¿Cómo funciona exactamente el ST-LINK/V2-1?

El ST-LINK/V2-1 es un programador/depurador integrado en la placa. Actúa como un puente entre tu computadora y el microcontrolador STM32. Permite cargar programas en la memoria flash del microcontrolador y facilita la depuración en tiempo real del código en ejecución.

3. ¿Por qué necesitamos configurar el reloj del sistema?

La configuración del reloj del sistema es crucial porque determina la velocidad a la que funciona el microcontrolador. Afecta directamente al rendimiento, consumo de energía y al timing de las operaciones. Una configuración adecuada asegura que el microcontrolador funcione de manera óptima para tu aplicación específica.

4. ¿Qué es exactamente HAL y por qué lo usamos?

HAL significa Hardware Abstraction Layer (Capa de Abstracción de Hardware). Es una biblioteca que proporciona una interfaz de programación de alto nivel para interactuar con el hardware del microcontrolador. Usamos HAL porque simplifica la programación, mejora la portabilidad del código entre diferentes microcontroladores STM32 y reduce la curva de aprendizaje.

5. ¿Cómo maneja STM32CubeIDE las dependencias y bibliotecas?

STM32CubeIDE gestiona automáticamente las dependencias y bibliotecas necesarias para tu proyecto. Cuando configuras tu proyecto usando STM32CubeMX, selecciona y configura automáticamente las bibliotecas HAL y CMSIS necesarias. Puedes ver estas dependencias en la estructura de carpetas del proyecto, específicamente en la carpeta "Drivers".

6. ¿Qué significa el polling en el contexto de nuestro programa de LED parpadeante?

En nuestro programa de LED parpadeante, usamos un enfoque de polling, lo que significa que nuestro programa está constantemente verificando (en un bucle) si es tiempo de cambiar el estado del LED. Es un método simple, pero no siempre el más eficiente. En futuras sesiones, exploraremos métodos basados en interrupciones que pueden ser más eficientes para ciertas aplicaciones.

7. ¿Cómo podemos depurar nuestro programa si algo no funciona como esperamos?

STM32CubeIDE ofrece potentes herramientas de depuración. Puedes establecer puntos de interrupción en tu código, ejecutar el programa paso a paso, y observar los valores de las variables en tiempo real. También puedes utilizar la vista de registros para ver el estado interno del microcontrolador. En futuras sesiones, profundizaremos en estas técnicas de depuración.

2. Configuración de periféricos con STM32CubeMX

2.1 Uso de la herramienta gráfica para configurar pines

1. Apertura de STM32CubeMX desde el proyecto existente

1. Abre tu proyecto en STM32CubeIDE.
2. En el explorador de proyectos, busca el archivo con extensión .ioc. Para la placa NUCLEO-F401RE, probablemente se llamará NUCLEO-F401RE.ioc.
3. Haz doble clic en este archivo. Esto abrirá la interfaz gráfica de STM32CubeMX dentro de STM32CubeIDE.

Nota: Si por alguna razón no ves el archivo .ioc, puedes crear uno nuevo:

- Haz clic derecho en tu proyecto
- Selecciona "New" > "Other"
- Busca y selecciona "STM32 Project from STM32CubeMX"
- Sigue el asistente para crear un nuevo archivo .ioc

2. Navegación por la interfaz gráfica

Una vez que STM32CubeMX se abra, verás varias áreas principales en la interfaz:

a) Barra de herramientas superior

Contiene opciones como "Generate Code", "Pinout", "Clock Configuration", y "Project Manager".

b) Panel de Pinout & Configuration (izquierda)

Muestra categorías de periféricos y configuraciones disponibles.

c) Vista de Pinout (centro)

Muestra un diagrama del microcontrolador con todos sus pines.

d) Panel de configuración (abajo)

Muestra opciones detalladas para el pin o periférico seleccionado.

e) Panel de mensajes (derecha)

Muestra advertencias, errores y sugerencias.

3. Selección y configuración de pines

Selección de pines:

1. En la vista de Pinout, haz clic en el pin que deseas configurar.
2. Aparecerá un menú desplegable con las funciones disponibles para ese pin.
3. Selecciona la función deseada (por ejemplo, GPIO_Output para un LED).

Configuración de pines:

1. Después de seleccionar una función para un pin, el panel de configuración en la parte inferior se actualizará.
2. Aquí puedes ajustar configuraciones específicas como:
    - Modo de GPIO (Input, Output, Analog, etc.)
    - Pull-up/Pull-down
    - Velocidad
    - Nivel de salida inicial

Ejemplo: Configurar un nuevo LED en PB10

1. Localiza el pin PB10 en el diagrama de pinout.
2. Haz clic en PB10 y selecciona "GPIO_Output" del menú desplegable.
3. En el panel de configuración inferior:
    - GPIO mode: Output Push Pull
    - GPIO Pull-up/Pull-down: No pull-up and no pull-down
    - Maximum output speed: Low
    - User Label: LED2 (o el nombre que prefieras)

Configuración de periféricos:

1. En el panel izquierdo, expande las categorías de periféricos (como Timers, Communication, Analog, etc.).
2. Selecciona el periférico que deseas configurar.
3. En el panel de configuración, ajusta los parámetros según sea necesario.

Resolución de conflictos:

Si ves un pin en rojo, significa que hay un conflicto:

1. Haz clic en el pin para ver detalles del conflicto.

2. STM32CubeMX a menudo sugerirá soluciones. Puedes aceptar una sugerencia o resolver manualmente el conflicto cambiando la configuración del pin.

4. Generación de código

Después de hacer tus cambios:

1. Haz clic en "Generate Code" en la barra de herramientas superior.
2. Revisa los cambios en la ventana emergente y haz clic en "Generate".
3. STM32CubeMX actualizará los archivos de tu proyecto con la nueva configuración.

5. Consejos adicionales

- Usa la función de búsqueda en la parte superior izquierda para encontrar rápidamente pines o periféricos específicos.
- Explora las pestañas "Clock Configuration" y "Project Manager" para ajustes adicionales del sistema.
- Siempre revisa el panel de mensajes (derecha) para ver advertencias o sugerencias importantes.
- Después de generar código, asegúrate de revisar los cambios en tus archivos de código, especialmente main.c.

Recuerda, cualquier cambio manual que hayas hecho en áreas marcadas con comentarios como /* USER CODE BEGIN */ y /* USER CODE END */ se preservará cuando regeneres el código. Sin embargo, es una buena práctica hacer una copia de seguridad de tu proyecto antes de hacer cambios significativos.

2.2 Configuración del LED incorporado

Configuración del pin PA5 como salida GPIO en STM32CubeMX

El pin PA5 en la placa NUCLEO-F401RE está conectado al LED incorporado (LD2). Configurarlo correctamente es esencial para controlar este LED en tus proyectos. Sigue esta guía paso a paso para configurar PA5 como una salida GPIO y ajustar sus propiedades.

1. Abrir STM32CubeMX

1. En STM32CubeIDE, abre tu proyecto.
2. Haz doble clic en el archivo .ioc de tu proyecto para abrir STM32CubeMX.

2. Localizar el pin PA5

1. En la vista de Pinout, busca el pin PA5.

- Puedes usar la función de búsqueda en la parte superior izquierda y escribir "PA5".
- O puedes buscarlo manualmente en el diagrama del chip.

3. Configurar PA5 como salida GPIO

1. Haz clic en el pin PA5.
2. En el menú desplegable que aparece, selecciona "GPIO_Output".
    - Notarás que el pin cambia de color, generalmente a verde, indicando que ahora está configurado como una salida GPIO.

4. Ajustar las propiedades del pin

Después de configurar PA5 como salida, necesitamos ajustar sus propiedades. Estas se encuentran en el panel de configuración en la parte inferior de la interfaz.

1. En el panel de configuración, busca la sección "GPIO".
2. Ajusta las siguientes propiedades: a) **GPIO output level**:
    - Opciones: Low o High
    - Recomendación: Selecciona "Low" para que el LED esté apagado inicialmente.

b) **GPIO mode**:

- Opciones: Push-pull o Open drain
- Recomendación: Selecciona "Push-pull" para un control directo del LED.

c) **GPIO Pull-up/Pull-down**:

- Opciones: No pull-up and no pull-down, Pull-up, o Pull-down
- Recomendación: Selecciona "No pull-up and no pull-down" ya que el LED tiene su propia resistencia en serie.

d) **Maximum output speed**:

- Opciones: Low, Medium, High, Very High
- Recomendación: Para un simple LED, "Low" es suficiente. Velocidades más altas pueden ser necesarias para aplicaciones que requieren conmutación rápida.

e) **User Label**:

- Puedes asignar un nombre descriptivo, como "LD2" o "LED_GREEN".
- Esto facilitará la referencia al pin en tu código.

5. Entender las configuraciones

- **Push-pull vs Open drain**:
  - Push-pull puede establecer activamente el pin en alto o bajo.
  - Open drain solo puede establecer el pin en bajo o dejarlo en alta impedancia.
- **Pull-up/Pull-down**:
  - Para una salida, generalmente no se necesita. Se usa más comúnmente para entradas.
- **Velocidad de salida**:
  - Afecta la tasa de cambio de la señal y el consumo de energía.
  - Velocidades más altas pueden causar más ruido y consumir más energía.

6. Generar el código

1. Una vez configurado el pin, haz clic en "Generate Code" en la barra de herramientas superior.
2. STM32CubeMX actualizará los archivos de tu proyecto con la nueva configuración.

7. Uso en el código

Después de generar el código, puedes controlar el LED en tu archivo main.c usando funciones HAL:

```
// Encender el LED
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);

// Apagar el LED
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

// Alternar el estado del LED
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
```

Si has usado un User Label, podrías ver definiciones como:

```
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
```

Lo que te permitiría usar:

```
HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
```

2.3 Configuración del botón incorporado

Configuración del pin PC13 como entrada GPIO en STM32CubeMX

El pin PC13 en la placa NUCLEO-F401RE está conectado al botón de usuario (B1). Configurarlo correctamente es esencial para detectar las pulsaciones del botón en tus proyectos. Sigue esta guía paso a paso para configurar PC13 como una entrada GPIO y ajustar sus propiedades.

1. Abrir STM32CubeMX

1. En STM32CubeIDE, abre tu proyecto.
2. Haz doble clic en el archivo .ioc de tu proyecto para abrir STM32CubeMX.

2. Localizar el pin PC13

1. En la vista de Pinout, busca el pin PC13.
    - Puedes usar la función de búsqueda en la parte superior izquierda y escribir "PC13".
    - O puedes buscarlo manualmente en el diagrama del chip.

3. Configurar PC13 como entrada GPIO

1. Haz clic en el pin PC13.
2. En el menú desplegable que aparece, selecciona "GPIO_Input".
    - Notarás que el pin cambia de color, generalmente a verde, indicando que ahora está configurado como una entrada GPIO.

4. Ajustar las propiedades del pin

Después de configurar PC13 como entrada, necesitamos ajustar sus propiedades. Estas se encuentran en el panel de configuración en la parte inferior de la interfaz.

1. En el panel de configuración, busca la sección "GPIO".
2. Ajusta las siguientes propiedades: a) **GPIO mode**:
    - Opción: Input mode
    - Esta es la configuración por defecto para una entrada GPIO.

b) **GPIO Pull-up/Pull-down**:

- Opciones: No pull-up and no pull-down, Pull-up, o Pull-down
- Recomendación: Selecciona "Pull-up" para este botón específico.
- Explicación: El botón en la NUCLEO-F401RE está conectado a tierra cuando se presiona. Una resistencia pull-up asegura un estado alto cuando el botón no está presionado.

c) **User Label**:

- Puedes asignar un nombre descriptivo, como "B1" o "USER_BTN".
- Esto facilitará la referencia al pin en tu código.

5. Entender la configuración Pull-up

La configuración pull-up es crucial para el correcto funcionamiento del botón:

- Cuando el botón no está presionado, la resistencia pull-up mantiene el pin en estado alto (1 lógico).
- Cuando se presiona el botón, conecta el pin a tierra, llevándolo a estado bajo (0 lógico).
- Sin pull-up, el pin estaría en un estado "flotante" cuando el botón no está presionado, lo que podría llevar a lecturas inconsistentes.

6. Generar el código

1. Una vez configurado el pin, haz clic en "Generate Code" en la barra de herramientas superior.
2. STM32CubeMX actualizará los archivos de tu proyecto con la nueva configuración.

7. Uso en el código

Después de generar el código, puedes leer el estado del botón en tu archivo main.c usando funciones HAL:

```
// Leer el estado del botón
GPIO_PinState buttonState = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13);

if (buttonState == GPIO_PIN_RESET)
{
    // El botón está presionado
    // Realiza alguna acción
}
else
{
    // El botón no está presionado
}
```

Si has usado un User Label, podrías ver definiciones como:

```
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
```

Lo que te permitiría usar:

```
if (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET)
{
    // El botón está presionado
}
```

8. Consideraciones adicionales

- **Debouncing**: Los botones mecánicos pueden producir múltiples transiciones cuando se presionan o sueltan. Considera implementar un mecanismo de debounce en software.
- **Interrupciones**: Para una respuesta más eficiente, podrías configurar una interrupción externa en este pin. Esto se hace en la pestaña "System Core" > "GPIO" en STM32CubeMX.
- **Modo de bajo consumo**: PC13 tiene algunas características especiales relacionadas con el modo de bajo consumo y el RTC (Real-Time Clock). Si planeas usar estos modos, asegúrate de consultar el manual de referencia del microcontrolador.

2.4 Generación de código inicial

Guía de generación y análisis de código en STM32CubeMX

1. Proceso de generación de código

Después de configurar los pines y periféricos en STM32CubeMX, el siguiente paso es generar el código correspondiente. Este proceso crea o actualiza los archivos necesarios para tu proyecto basándose en la configuración que has realizado.

Pasos para generar código:

1. En la interfaz de STM32CubeMX, haz clic en el botón "Generate Code" en la barra de herramientas superior.
2. Si es la primera vez que generas código para este proyecto, se te pedirá que confirmes algunas opciones:
    - Nombre del proyecto
    - Ubicación del proyecto
    - Toolchain/IDE (normalmente STM32CubeIDE)
3. Haz clic en "Generate" para iniciar el proceso.
4. Aparecerá una ventana de progreso mostrando el estado de la generación de código.
5. Una vez completado, verás un mensaje indicando que la generación de código ha sido exitosa.

Notas importantes:

- Si ya has generado código anteriormente y has hecho cambios en la configuración, STM32CubeMX te preguntará si deseas sobrescribir los archivos existentes. Generalmente, debes aceptar para asegurar que tu código refleje la configuración más reciente.

- **STM32CubeMX preservará el código que hayas escrito dentro de las secciones marcadas con /* USER CODE BEGIN */ y /* USER CODE END */.**

2. Análisis del código generado

Después de la generación, es importante entender la estructura y el contenido del código generado. Los archivos principales a revisar son:

main.c

Ubicación: Core/Src/main.c

Este es el archivo principal de tu programa. Contiene:

1. Inclusiones de cabeceras necesarias
2. Definiciones de variables privadas
3. Prototipos de funciones privadas
4. La función main()
5. Funciones de inicialización de periféricos

Estructura típica de main():

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
```

```
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

stm32f4xx_hal_msp.c

Ubicación: Core/Src/stm32f4xx_hal_msp.c

Este archivo contiene las funciones de inicialización de bajo nivel para los periféricos HAL (Hardware Abstraction Layer). Aquí encontrarás la configuración de pines y clocks para los periféricos que hayas habilitado.

stm32f4xx_it.c

Ubicación: Core/Src/stm32f4xx_it.c

Contiene los manejadores de interrupciones para tu microcontrolador. Si has configurado interrupciones en STM32CubeMX, encontrarás aquí las funciones correspondientes.

3. Ubicación de los archivos de configuración

Los archivos de configuración generados por STM32CubeMX se distribuyen en varias ubicaciones dentro de la estructura del proyecto:

Archivos de cabecera principales:

- main.h: Core/Inc/main.h
- stm32f4xx_hal_conf.h: Core/Inc/stm32f4xx_hal_conf.h

Archivos de código fuente principales:

- main.c: Core/Src/main.c
- stm32f4xx_hal_msp.c: Core/Src/stm32f4xx_hal_msp.c
- stm32f4xx_it.c: Core/Src/stm32f4xx_it.c

Archivos de configuración del sistema:

- system_stm32f4xx.c: Core/Src/system_stm32f4xx.c

Archivo de descripción del proyecto:

- .ioc: En la raíz del proyecto (por ejemplo, NUCLEO-F401RE.ioc)

Drivers HAL y CMSIS:

- Drivers/STM32F4xx_HAL_Driver/
- Drivers/CMSIS/

Consejos para trabajar con el código generado:

1. Siempre escribe tu código dentro de las secciones marcadas con /* USER CODE BEGIN */ y /* USER CODE END */ para evitar que sea sobrescrito en futuras generaciones.
2. Si necesitas modificar la configuración de un periférico, es mejor hacerlo en STM32CubeMX y regenerar el código en lugar de modificar directamente los archivos generados.
3. Familiarízate con la estructura de main.c y cómo se inicializan los periféricos. Esto te ayudará a entender el flujo de ejecución de tu programa.
4. Revisa stm32f4xx_hal_msp.c si necesitas entender cómo se están configurando los pines y clocks para un periférico específico.
5. Si añades nuevas interrupciones, asegúrate de revisar stm32f4xx_it.c para ver dónde puedes añadir tu código de manejo de interrupciones.

Entender el proceso de generación de código y la estructura de los archivos generados es crucial para desarrollar proyectos eficientes y mantenibles con STM32CubeMX y STM32CubeIDE.

3. Programación de entrada/salida digital

3.1 Profundización en el control del LED

Las principales funciones HAL para controlar GPIO son:

```
HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

Ejemplos de encendido, apagado y conmutación del LED

Asumiendo que el LED está conectado al pin PA5:

```
// Encender el LED
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);

// Apagar el LED
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

// Conmutar el LED
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
```

Creación de funciones personalizadas para control del LED

```
#define LED_GPIO_Port GPIOA
#define LED_Pin GPIO_PIN_5

void LED_On(void)
```

```
{
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
}

void LED_Off(void)
{
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
}

void LED_Toggle(void)
{
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
}

void LED_Blink(uint32_t times, uint32_t delayMs)
{
    for (uint32_t i = 0; i < times; i++)
    {
        LED_Toggle();
        HAL_Delay(delayMs);
        LED_Toggle();
        HAL_Delay(delayMs);
    }
}
```

3.2 Lectura del estado del botón (polling)

Funciones HAL para leer entradas digitales

La función principal para leer entradas digitales es:

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

Implementación de un bucle de polling para detectar pulsaciones

Asumiendo que el botón está conectado al pin PC13:

```
#define BUTTON_GPIO_Port GPIOC
#define BUTTON_Pin GPIO_PIN_13

void CheckButton(void)
{
    if (HAL_GPIO_ReadPin(BUTTON_GPIO_Port, BUTTON_Pin) == GPIO_PIN_RESET)
    {
        // El botón está presionado
        // Realizar acción deseada
    }
}

// En el bucle principal
while (1)
{
    CheckButton();
    HAL_Delay(10); // Pequeño retardo para no saturar el CPU
}
```

Manejo del rebote del botón (debouncing)

El debouncing es crucial para evitar lecturas falsas debido a rebotes mecánicos del botón. Aquí hay una implementación simple:

```
#define DEBOUNCE_DELAY 50 // ms

uint32_t lastDebounceTime = 0;
GPIO_PinState lastButtonState = GPIO_PIN_SET;
GPIO_PinState buttonState;

void CheckButtonDebounced(void)
{
    GPIO_PinState reading = HAL_GPIO_ReadPin(BUTTON_GPIO_Port, BUTTON_Pin);

    if (reading != lastButtonState)
    {
        lastDebounceTime = HAL_GetTick();
    }

    if ((HAL_GetTick() - lastDebounceTime) > DEBOUNCE_DELAY)
    {
        if (reading != buttonState)
        {
            buttonState = reading;
            if (buttonState == GPIO_PIN_RESET)
            {
                // El botón ha sido presionado (después del debounce)
                // Realizar acción deseada
            }
        }
    }

    lastButtonState = reading;
}
```

3.3 Creación de un proyecto que controle el LED con el botón

Integración de la lectura del botón y el control del LED

Combinando las funciones anteriores, podemos crear un proyecto que controle el LED con el botón:

```
void HandleButton(void)
{
    static uint32_t lastDebounceTime = 0;
    static GPIO_PinState lastButtonState = GPIO_PIN_SET;
    static GPIO_PinState buttonState = GPIO_PIN_SET;

    GPIO_PinState reading = HAL_GPIO_ReadPin(BUTTON_GPIO_Port, BUTTON_Pin);

    if (reading != lastButtonState)
    {
        lastDebounceTime = HAL_GetTick();
    }

    if ((HAL_GetTick() - lastDebounceTime) > DEBOUNCE_DELAY)
    {
```

```
        if (reading != buttonState)
        {
            buttonState = reading;
            if (buttonState == GPIO_PIN_RESET)
            {
                LED_Toggle(); // Conmuta el LED cuando se presiona el botón
            }
        }
    }

    lastButtonState = reading;
}

int main(void)
{
    // ... inicialización ...

    while (1)
    {
        HandleButton();
        HAL_Delay(10);
    }
}
```

3.2 Implementación de diferentes comportamientos (toggle, press-and-hold)

Podemos expandir la funcionalidad para incluir diferentes comportamientos:

```
void HandleButtonAdvanced(void)
{
    static uint32_t lastDebounceTime = 0;
    static GPIO_PinState lastButtonState = GPIO_PIN_SET;
    static GPIO_PinState buttonState = GPIO_PIN_SET;
    static uint32_t pressStartTime = 0;
    static bool isLongPress = false;

    GPIO_PinState reading = HAL_GPIO_ReadPin(BUTTON_GPIO_Port, BUTTON_Pin);

    if (reading != lastButtonState)
    {
        lastDebounceTime = HAL_GetTick();
    }

    if ((HAL_GetTick() - lastDebounceTime) > DEBOUNCE_DELAY)
    {
        if (reading != buttonState)
        {
            buttonState = reading;
            if (buttonState == GPIO_PIN_RESET)
            {
                // Botón presionado
                pressStartTime = HAL_GetTick();
                isLongPress = false;
            }
            else
            {
                // Botón liberado
                if (!isLongPress)
                {
                    LED_Toggle(); // Toggle en pulsación corta
                }
            }
```

```
        }
        else if (buttonState == GPIO_PIN_RESET)
        {
            // Botón mantenido presionado
            if (!isLongPress && (HAL_GetTick() - pressStartTime > 1000))
            {
                isLongPress = true;
                LED_Blink(3, 100); // Parpadeo rápido en pulsación larga
            }
        }
    }

    lastButtonState = reading;
}
```

Demostración del proyecto funcionando en la placa

Para demostrar el proyecto en la placa:

1. Configura los pines PA5 (LED) y PC13 (Botón) en STM32CubeMX.
2. Genera el código e implementa las funciones anteriores en main.c.
3. Compila y carga el programa en la placa NUCLEO-F401RE.
4. Observa el comportamiento:
    - Pulsación corta del botón: Conmuta el LED.
    - Pulsación larga del botón (>1 segundo): El LED parpadea rápidamente 3 veces.

4. Cierre y proyecto práctico

4.1 Asignación de un mini proyecto

Descripción del proyecto: "Secuencia de luces controlada por botón"

En este proyecto, implementarás un sistema que controla una secuencia de parpadeo del LED incorporado en la placa NUCLEO-F401RE. El botón de usuario se utilizará para cambiar entre diferentes patrones de parpadeo.

Requisitos:

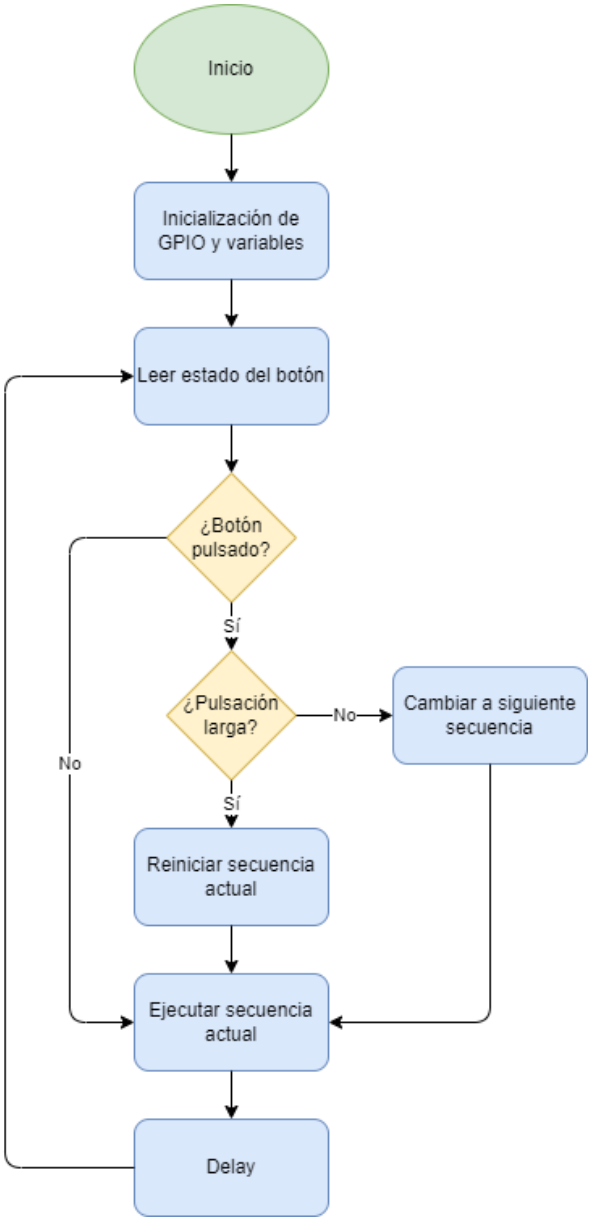
1. Utilizar el LED incorporado (PA5) y el botón de usuario (PC13) de la placa NUCLEO-F401RE.
2. Implementar al menos tres secuencias de parpadeo diferentes.
3. Utilizar el botón para cambiar entre las secuencias.

4. Implementar debouncing para el botón.
5. Utilizar funciones HAL para el control del GPIO.

Funcionamiento:

1. Al iniciar, el sistema comienza con la primera secuencia de parpadeo.
2. Cada pulsación corta del botón cambia a la siguiente secuencia.
3. Una pulsación larga (más de 2 segundos) reinicia la secuencia actual.

Diagrama de flujo



Ejemplo de implementación:

```
#include "main.h"

#define LED_PIN GPIO_PIN_5
#define LED_PORT GPIOA
#define BUTTON_PIN GPIO_PIN_13
#define BUTTON_PORT GPIOC

#define DEBOUNCE_DELAY 50
#define LONG_PRESS_DURATION 2000

typedef enum {
    SEQUENCE_1,
    SEQUENCE_2,
    SEQUENCE_3,
    SEQUENCE_COUNT
} LedSequence;

void LED_On(void) {
    HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_SET);
}

void LED_Off(void) {
    HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_RESET);
}

void LED_Toggle(void) {
    HAL_GPIO_TogglePin(LED_PORT, LED_PIN);
}

void RunSequence(LedSequence sequence) {
    switch (sequence) {
        case SEQUENCE_1:
            LED_Toggle();
            HAL_Delay(500);
            break;
        case SEQUENCE_2:
            LED_On();
            HAL_Delay(200);
            LED_Off();
            HAL_Delay(800);
            break;
        case SEQUENCE_3:
            LED_On();
            HAL_Delay(100);
            LED_Off();
            HAL_Delay(100);
            LED_On();
            HAL_Delay(100);
            LED_Off();
            HAL_Delay(700);
            break;
    }
}

int main(void) {
    HAL_Init();
    SystemClock_Config();
```

```
    MX_GPIO_Init();

    LedSequence currentSequence = SEQUENCE_1;
    uint32_t lastDebounceTime = 0;
    uint32_t buttonPressStartTime = 0;
    GPIO_PinState lastButtonState = GPIO_PIN_SET;
    GPIO_PinState buttonState = GPIO_PIN_SET;
    bool longPressDetected = false;

    while (1) {
        GPIO_PinState reading = HAL_GPIO_ReadPin(BUTTON_PORT, BUTTON_PIN);

        if (reading != lastButtonState) {
            lastDebounceTime = HAL_GetTick();
        }

        if ((HAL_GetTick() - lastDebounceTime) > DEBOUNCE_DELAY) {
            if (reading != buttonState) {
                buttonState = reading;
                if (buttonState == GPIO_PIN_RESET) {
                    buttonPressStartTime = HAL_GetTick();
                    longPressDetected = false;
                } else {
                    if (!longPressDetected) {
                        currentSequence = (currentSequence + 1) % SEQUENCE_COUNT;
                    }
                }
            } else if (buttonState == GPIO_PIN_RESET) {
                if (!longPressDetected && (HAL_GetTick() - buttonPressStartTime >
LONG_PRESS_DURATION)) {
                    longPressDetected = true;
                    // Reiniciar la secuencia actual
                }
            }
        }

        lastButtonState = reading;

        RunSequence(currentSequence);
    }
}
```

4.2 Variaciones y mejoras

Una mejora significativa para este proyecto es la incorporación de la placa LED&KEY, que utiliza el módulo TM1638, que veremos en la siguiente práctica. Esta adición amplía enormemente las posibilidades de interacción y visualización del proyecto.

Uso de la placa LED&KEY con módulo TM1638

El módulo TM1638 ofrece las siguientes características:

- 8 displays de 7 segmentos
- 8 LEDs rojos de 3mm
- 8 pulsadores
- Control de brillo de displays y LEDs

- Detección de pulsación múltiple de botones
- Control mediante solo 3 líneas de comunicación

Conexión del módulo TM1638 a la placa NUCLEO-F401RE:

1. VCC: Conectar a 3.3V de la NUCLEO-F401RE
2. GND: Conectar a GND de la NUCLEO-F401RE
3. STB (Strobe): Conectar a un pin GPIO, por ejemplo, PB6
4. CLK (Clock): Conectar a un pin GPIO, por ejemplo, PB7
5. DIO (Data I/O): Conectar a un pin GPIO, por ejemplo, PB8

4.3 Recursos adicionales para seguir aprendiendo

Esta guía proporciona una lista de recursos para que puedas continuar tu aprendizaje y desarrollo con microcontroladores STM32.

Documentación oficial de STMicroelectronics

1. STM32 MCU & MPU Portfolio
    - Visión general de toda la gama de microcontroladores STM32.
    - Útil para comparar diferentes familias y seleccionar el microcontrolador adecuado para futuros proyectos.
2. STM32F4 Series Reference Manual
    - Documento exhaustivo que cubre todos los periféricos y funcionalidades del STM32F4.
    - Esencial para entender los detalles de bajo nivel del microcontrolador.
3. STM32CubeF4 User Manual
    - Guía detallada sobre el uso de las bibliotecas HAL y LL para STM32F4.
    - Importante para comprender cómo utilizar eficientemente las funciones HAL en tus proyectos.
4. STM32CubeIDE User Manual
    - Instrucciones completas sobre el uso del entorno de desarrollo integrado.
    - Útil para dominar las herramientas de desarrollo y depuración.
5. STM32 Application Notes

- Colección de documentos que abordan aplicaciones específicas y técnicas de implementación.
- Excelente para aprender mejores prácticas y soluciones a problemas comunes.

Tutoriales y proyectos en línea

Estos recursos ofrecen aprendizaje práctico y ejemplos de aplicaciones del mundo real.

1. STM32 Tutorials by Shawn Hymel
   - Serie de tutoriales que cubren desde los conceptos básicos hasta aplicaciones avanzadas.
   - Incluye ejemplos de código y explicaciones detalladas.
2. Carmine Noviello's Mastering STM32 Book
   - Libro completo disponible online que cubre programación de STM32 en profundidad.
   - Incluye ejemplos prácticos y explicaciones detalladas de conceptos avanzados.
3. DigiKey's Introduction to STM32 Series
   - Serie de videos educativos que cubren los fundamentos de la programación de STM32.
   - Excelente para reforzar conceptos básicos y ver demostraciones prácticas.
4. STM32 Projects on Hackster.io
   - Plataforma con numerosos proyectos de la comunidad utilizando STM32.
   - Inspiración para proyectos y ejemplos de aplicaciones prácticas.
5. Embedded Systems Course by FastBit
   - Curso en línea que cubre sistemas embebidos con un enfoque en STM32.
   - Ofrece una comprensión profunda de los conceptos de sistemas embebidos.

Foros y comunidades de desarrolladores STM32

Estos foros y comunidades son excelentes para obtener ayuda, compartir conocimientos y mantenerse actualizado.

1. STM32 Forum Oficial

- Foro oficial de STMicroelectronics para discusiones sobre STM32.
- Lugar ideal para obtener soporte técnico y discutir con expertos de ST.

2. STM32 Subreddit
   - Comunidad activa en Reddit dedicada a STM32.
   - Bueno para discusiones informales, compartir proyectos y obtener consejos rápidos.
3. Stack Overflow - STM32 Tag
   - Plataforma de preguntas y respuestas para problemas específicos de programación.
   - Excelente para resolver problemas técnicos concretos.
4. EmbeddedRelated.com STM32 Forum
   - Foro especializado en sistemas embebidos con una sección dedicada a STM32.
   - Discusiones de nivel avanzado y compartición de experiencias.
5. STM32 Discord Server
   - Servidor de Discord dedicado a STM32 y desarrollo de sistemas embebidos.
   - Ideal para chatear en tiempo real con otros desarrolladores y obtener ayuda rápida.

Evaluación y seguimiento

- Los estudiantes deben completar el mini proyecto asignado
- Se valorará la creatividad en la implementación de funcionalidades adicionales