Taller de Sistemas Embebidos
STM32 MCU – Flash Memory Management

Información relevante

Taller de Sistemas Embebidos

Asignatura correspondiente a la **actualización 2023** del Plan de Estudios 2020 y resoluciones modificatorias, de Ingeniería Electrónica de FIUBA

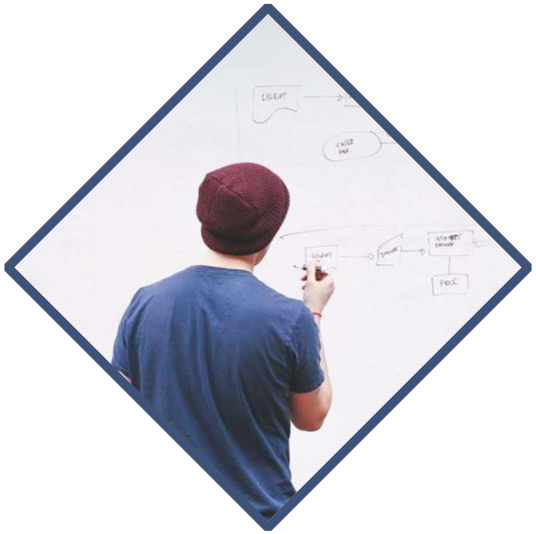
Estructura Curricular de la Carrera

El **Proyecto Intermedio** se desarrolla en la asignatura **Taller de Sistemas Embebidos**, la cual tiene un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**

Más información . . .

. . . sobre la **actualización 2023** . . . https://www.fi.uba.ar/grado/carreras/ingenieria-electronica/plan-de-estudios

. . . sobre el **Taller de Sistemas Embebidos** . . . https://campusgrado.fi.uba.ar/course/view.php?id=1217



¡Hola!

Soy Juan Manuel Cruz
Taller de Sistemas Embebidos
Consultas a: jcruz@fi.uba.ar

1

Introducción

Actualización 2023 del Plan de Estudios 2020 y resoluciones . . .

Conceptos básicos

- Referencia:
  - Mastering STM32 - A step-by-step guide to the most complete ARM Cortex-M platform, using a free and powerful development environment based on Eclipse and GCC - Carmine Noviello (Author)
  - Chapter 18: Flash Memory Mangement
    - La memoria flash es un periférico silencioso que utilizamos sin preocuparnos demasiado por ello. Una vez que estemos seguros de que la memoria flash tiene suficiente espacio para almacenar el firmware, cargamos la imagen binaria usando el depurador o una herramienta de actualización dedicada. Y lo olvidamos por completo.

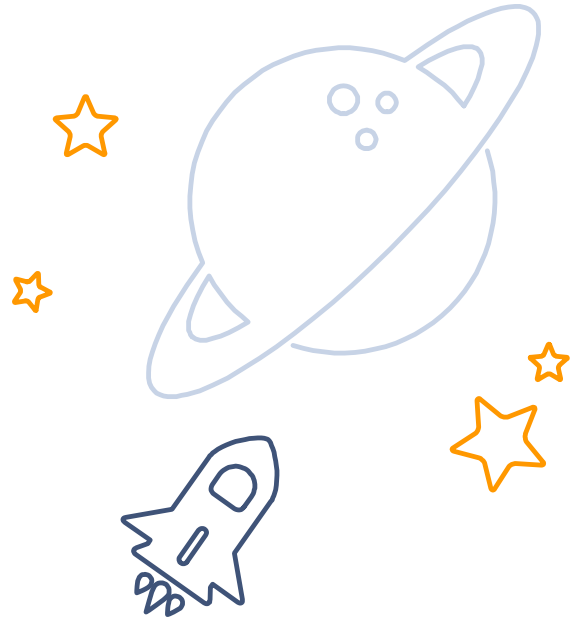
Conceptos básicos

- Sin embargo, la flash interna proporcionada por todos los microcontroladores STM32 funciona de la misma manera que otros periféricos.
- Se puede programar directamente desde el firmware configurando registros específicos, y esto nos permite actualizar el firmware usando el mismo código integrado o almacenar datos de configuración relevantes sin usar hardware externo dedicado (una EEPROM I^2C externa o un flash SPI).

Conceptos básicos

- Este capítulo muestra cómo programar la memoria flash STM32 interna utilizando el módulo HAL_-FLASH dedicado de CubeHAL.
- Describe cómo suele organizarse la memoria flash en un microcontrolador STM32 típico, ilustrando brevemente las diferencias entre cada familia y los pasos necesarios para programar áreas específicas de esta memoria directamente desde el mismo microcontrolador.
- Finalmente, se describe el papel del Acelerador ART™, junto con las evoluciones de esta tecnología propia de ST en los microcontroladores STM32F7.

Solución Adecuada



. . . lo más simple posible, previa determinación del objetivo de excelencia a cumplir, obviamente contando con la documentación debida y recurriendo a la metodología de trabajo adecuada

2

Documentación debida

1er Cuatrimestre de 2024, dictado por primera vez . . .

“Mastering STM32 - A step-by-step guide to the most complete ARM Cortex-M platform, using a free and powerful development environment based on Eclipse and GCC - Carmine Noviello (Author)

Chapter 18: Flash Memory Mangement

Introduction to STM32 Flash Memory

- A diferencia de otras arquitecturas embebidas, todos los microcontroladores STM32 proporcionan una memoria flash dedicada para almacenar código de programa y datos constantes.
- Actualmente hay once tamaños de memoria, que van desde 16 KB hasta 2 MB. El último dígito del número de pieza de una MCU STM32 determinada define el tamaño de la memoria flash, como se muestra en la Tabla 1. Por ejemplo, una MCU STM32F401RE tiene 512 KB de memoria flash.

Table 1: The size of the flash memory given the last digit in an STM32 part number

Last digit in P/N	Flash memory size (KB)
4	16
6	32
8	64
B	128
Z	192
C	256
D	384

Table 1: The size of the flash memory given the last digit in an STM32 part number

Last digit in P/N	Flash memory size (KB)
E	512
F	768
G	1024
I	2048

Introduction to STM32 Flash Memory

- Dependiendo de la familia STM32, el tipo el empaque usado, la memoria flash de una MCU STM32 se puede organizar en:
  - Uno o dos bancos: la mayoría de los microcontroladores STM32 proporcionan sólo un banco de memoria flash, mientras que los de mayor rendimiento hasta dos bancos.
    - La arquitectura multibanco permite operaciones duales y simultáneas: mientras se programa o borra en un banco, es posible realizar operaciones de lectura en el otro.
    - Este enfoque proporciona una mayor flexibilidad para operaciones duales, especialmente para aplicaciones de alto rendimiento.
    - En algunas MCU STM32 más recientes, como la última STM32F7, el multibanco es una función programable que se puede habilitar opcionalmente y los tamaños de los bancos se pueden configurar según sea necesario.

Introduction to STM32 Flash Memory

- Cada banco está a su vez dividido en sectores: cada banco de memoria flash está dividido en varios subbloques, llamados sectores.
  - Algunas MCU STM32 proporcionan memoria flash con todos los sectores del mismo tamaño (normalmente igual a 1 KB o 2 KB).
  - Algunos otros proporcionan varios sectores con diferentes tamaños (normalmente los primeros sectores tienen un tamaño menor que los restantes).
- Cada sector se puede dividir en páginas: en algunas MCU STM32, un sector se divide en varias páginas más pequeñas.
  - A veces, esto sucede sólo para los primeros sectores, y esto permite borrar y luego programar sólo una fracción del sector.

Introduction to STM32 Flash Memory

- La Tabla 2 muestra cómo se organiza la memoria flash en algunos microcontroladores STM32F0.
  - Como puedes ver, pueden proporcionar hasta diecisiete sectores, cada uno a su vez dividido en cuatro páginas.
  - Además, un área dedicada, llamada Bloque de información, está asignada a otro rango de direcciones: esta memoria no volátil se utiliza para almacenar registros de configuración especiales (llamados bytes de opción) y algunos cargadores de arranque (bootloaders) preprogramados de fábrica, que estudiaremos en el siguiente capítulo.
  - En las MCU STM32 más potentes, la región del bloque de información también contiene la memoria programable una sola vez (OTP) (que puede oscilar entre 512 y 1024 bytes): esta es una memoria no volátil que se puede usar para almacenar parámetros de configuración relevantes del dispositivo.

Introduction to STM32 Flash Memory

- ¿Por qué tener tal organización de la memoria? Antes de que podamos responder a esta pregunta, debemos introducir algunos conceptos fundamentales sobre las tecnologías de memoria flash. Sin entrar en detalles específicos de implementación, existen dos familias principales de memorias flash: NAND y NOR.
- Las memorias NAND-flash ofrecen una arquitectura física más compacta, permitiendo almacenar más celdas de memoria en la misma área de silicio.
- Las memorias NAND están disponibles en mayores densidades de almacenamiento y a menores costos por bit que las NOR-flash (recuerde que en electrónica, aparte de los costos de I+D, el costo de producción de un IC tiene que ver con el tamaño del die).
- Las memorias NAND también tienen hasta diez veces la durabilidad de las memorias NOR-flash. NAND es más adecuado como medio de almacenamiento para archivos grandes, incluidos vídeo y audio. Las memorias USB, las tarjetas SD y las tarjetas MMC son de tipo NAND.

Introduction to STM32 Flash Memory

Flash area	Flash memory addresses	Size (byte)	Name	Description(1)
Main Flash memory	0x0800 0000 - 0x0800 03FF	1 Kbyte	Page 0	Sector 0
	0x0800 0400 - 0x0800 07FF	1 Kbyte	Page 1	
	0x0800 0800 - 0x0800 0BFF	1 Kbyte	Page 2	
	0x0800 0C00 - 0x0800 0FFF	1 Kbyte	Page 3	
	. . .	. . .	. . .	. . .
	0x0800 7000 - 0x0800 73FF	1 Kbyte	Page 28	Sector 7 (1)
	0x0800 7400 - 0x0800 77FF	1 Kbyte	Page 29	
	0x0800 7800 - 0x0800 7BFF	1 Kbyte	Page 30	
	0x0800 7C00 - 0x0800 7FFF	1 Kbyte	Page 31	
	. . .	. . .	. . .	. . .
	0x0800 F000 - 0x0800 F3FF	1 Kbyte	Page 60	Sector 15
	0x0800 F400 - 0x0800 F7FF	1 Kbyte	Page 61	
	0x0800 F800 - 0x0800 FBFF	1 Kbyte	Page 62	
	0x0800 FC00 - 0x0800 FFFF	1 Kbyte	Page 63	
Information block	0x1FFF EC00 - 0x1FFF F7FF	3 Kbyte(2)	-	System memory
	0x1FFF C400 -0x1FFF F7FF	13 Kbyte(3)	-	System memory
	0x1FFF F800 - 0x1FFF F80F	2 x 8 byte	-	Option byte

1. On STM32F030x4 devices, the main Flash memory space is limited to sector 3. On STM32F030x6 and STM32F070x6 devices, the main Flash memory is limited to sector 7.
2. STM32F030x4, STM32F030x6 and STM32F030x8 devices
3. STM32F070x6 devices

Table 2: Flash memory organization in F030x4, F030x6, F070x6 and F030x8 devices

Introduction to STM32 Flash Memory

- NAND-flash no proporciona un bus de direcciones externo de acceso aleatorio, por lo que los datos deben leerse por bloques, donde cada bloque contiene cientos o miles de bits, lo que se asemeja a una especie de acceso secuencial a datos.
  - Esto hace que la tecnología NAND-flash no sea adecuada para microcontroladores embebidos, porque la mayoría de los microprocesadores y microcontroladores requieren acceso aleatorio a nivel de bytes.
  - Una cosa importante que hay que saber sobre las tecnologías de memoria flash es que una operación de escritura en cualquier tipo de dispositivo flash sólo se puede realizar en una unidad vacía o borrada.
  - Por tanto, en la mayoría de los casos una operación de escritura debe ir precedida de una operación de borrado.

Introduction to STM32 Flash Memory

- Si bien la operación de borrado es bastante sencilla en el caso de dispositivos flash NAND, en flash NOR es obligatorio que todos los bytes del bloque de destino se escriban con ceros antes de poder borrarlos.
- Por el contrario, las memorias NOR-flash ofrecen buses completos de direcciones y datos para acceder aleatoriamente a cualquiera de sus ubicaciones de memoria (direccionables a cada byte).

- Esto los hace adecuados para almacenar código y datos constantes, porque rara vez necesitan actualizarse.
- La resistencia de las memorias NOR es de 10.000 a 100.000 ciclos de borrado.
  - Las memorias flash NOR son más lentas en operaciones de borrado y escritura en comparación con las memorias flash NAND.
  - Eso significa que la NAND-flash tiene tiempos de borrado y escritura más rápidos.

Introduction to STM32 Flash Memory

- Además, NAND tiene unidades de borrado más pequeñas.
- Por lo tanto, se necesitan menos borrados y esto los hace más adecuados para almacenar sistemas de archivos.
- La memoria flash NOR puede leer datos un poco más rápido que NAND.

Los dispositivos NOR-flash se dividen en unidades de borrado, también llamadas bloques, páginas o sectores.

- Esta división es necesaria para reducir los precios y superar las limitaciones físicas.
- La escritura de información en un bloque específico solo se puede realizar si ese bloque está vacío/borrado, como se dijo antes.
- En la mayoría de las memorias NOR-flash, después de un ciclo de borrado, una celda individual contiene el valor “1”, y una operación de escritura permite cambiar su valor a “0”.

Introduction to STM32 Flash Memory

- Esto significa que una ubicación de memoria de palabras se establece en 0xFFFF FFFF después de un borrado.
- Sin embargo, existen algunas memorias NOR-flash donde el valor predeterminado de la celda después de un borrado es “0”, y podemos establecerlo en “1” con una operación de escritura.

Particionar la memoria flash en varios bloques nos da una ventaja indirecta: podemos borrar y luego reprogramar sólo pequeñas fracciones de la memoria flash.

- Esto es especialmente útil cuando utilizamos la memoria flash para almacenar parámetros de configuración no volátiles, sin utilizar memorias EEPROM dedicadas y externas.

Introduction to STM32 Flash Memory

- Para evitar por completo escrituras no deseadas en la memoria no volátil (NVM), la memoria flash en todas las MCU STM32 está protegida contra escritura y existe una secuencia de desbloqueo específica a seguir para deshabilitarla: se proporcionan dos registros de clave dedicados en la región Option Bytes, que permiten deshabilitar la protección contra escritura flash escribiendo un valor específico dentro de ellos.
  - En algunas MCU STM32, la protección contra escritura debe desactivarse individualmente para cada sector. Dependiendo de la familia STM32, el acceso de escritura se realiza en 8, 16, 32 o 64 bits.
- Para proteger la propiedad intelectual, la memoria flash puede protegerse contra el acceso externo desde la interfaz de depuración (claramente, el acceso de lectura aún está permitido desde el núcleo Cortex-M y los controladores DMA).

Introduction to STM32 Flash Memory

- Esto evita que otros usuarios malintencionados puedan guardar el contenido de la memoria flash para desensamblarla o replicarla en dispositivos falsificados.
- Analizaremos este tema más adelante.

- Dependiendo de la familia STM32, la memoria flash puede realizar varias operaciones de programación/borrado en paralelo, lo que permite escribir más bytes a la vez.
  - Se deben cumplir condiciones particulares para llevar a cabo operaciones del programa en paralelo.
  - Generalmente, se requiere un voltaje VDD determinado para alcanzar el paralelismo máximo.
  - Consulta siempre el manual de referencia de tu MCU para descubrir más sobre esto.

The HAL_FLASH Module

- Como todos los demás periféricos STM32, incluso la memoria flash, proporciona varios registros que se utilizan para manipular su configuración, como se dijo antes.
- El módulo HAL_FLASH, junto con el módulo HAL_FLASHEx relacionado, permite borrar y reprogramar fácilmente la memoria NVM sin tener que lidiar demasiado con los detalles de su implementación.
- Los siguientes subpárrafos presentan las funciones más relevantes de esos módulos.

Flash Memory Unlocking

- La memoria flash está protegida contra escritura de forma predeterminada, para evitar escrituras accidentales causadas por perturbaciones eléctricas o mal funcionamiento del programa.
  - Para habilitar el modo de escritura se debe realizar una secuencia de operaciones, y esto es específico de la familia STM32 dada.
  - Para realizar esta tarea, CubeHAL proporciona la función:
    - HAL_StatusType Def HAL_FLASH_Unlock(void);
- lo que nos permite ignorar por completo la arquitectura de memoria flash específica. Una vez que la protección contra escritura/borrado de la memoria flash está desactivada, podemos realizar una operación de borrado o escritura.
  - El procedimiento inverso al desbloqueo se realiza utilizando la función:
    - HAL_StatusType Def HAL_FLASH_Lock(void);

Flash Memory Unlocking

- La protección contra escritura se establece automáticamente al reiniciar el sistema.
  - Sin embargo, se recomienda volver a bloquear explícitamente la memoria cuando se completen todas las operaciones de escritura.
  - Esto evita cualquier escritura accidental causada por un mal funcionamiento del firmware o inestabilidad de energía.

Flash Memory Erasing

- Antes de que podamos cambiar el contenido de una ubicación de memoria flash, debemos restablecer sus bits al valor predeterminado (“0” o “1” según el tipo de flash NOR)
  - Esto se realiza mediante una operación de borrado en la granularidad del sector/página.
  - Alternativamente, se puede realizar un borrado masivo de todo el banco: esto significa que en aquellas MCU STM32 que proporcionan dos bancos podemos borrar en masa cada banco a la vez.
- En la mayoría de los microcontroladores STM32, las celdas individuales de un bloque de memoria flash (sector o página) se configuran en "1" después de una operación de borrado, con solo dos excepciones notables: los microcontroladores STM32L0 y STM32L1, cuyo valor predeterminado es "0". .
- CubeHAL proporciona dos formas de realizar una operación de borrado flash: borrado flash en modo de polling e interrupción.

Flash Memory Erasing

- La función:
  - HAL_StatusTypeDef HAL_FLASHEx_Erase(FLASH_EraseInitTypeDef *pEraseInit, uint32_t *SectorError);
- permite realizar un borrado flash en modo polling.
  - Acepta un puntero a una instancia de la estructura FLASH_EraseInitTypeDef, que veremos más adelante, y un puntero a la variable (SectorError) que devuelve la identificación de los sectores/páginas defectuosas en caso de error durante el procedimiento de borrado (por ejemplo , si el procedimiento de borrado falla en la cuarta página, el parámetro SectorError contendrá el valor 3).
- La estructura FLASH_EraseInitTypeDef difiere mucho entre cada familia STM32. Por este motivo, eche un vistazo al archivo stm32XXxx_hal_flash_ex.h de CubeHAL para su MCU. Aquí, vamos a considerar la implementación que se encuentra en CubeHAL para las MCU STM32 de mayor rendimiento, como las F2/F4/F7.

Flash Memory Erasing

```
typedef struct {
  uint32_t TypeErase;   /* Mass erase or sector Erase */
  uint32_t Banks;       /* Select banks to erase when Mass erase is enabled */
  uint32_t Sector;      /* Initial FLASH sector to erase when Mass erase is disabled */
  uint32_t NbSectors;   /* Number of sectors to be erased */
  uint32_t VoltageRange;/* The device voltage range which defines the erase parallelism */
} FLASH_EraseInitTypeDef;
```

- TypeErase: especifica si estamos realizando un borrado masivo de todo el banco o un borrado de sector/página. Puede asumir los valores FLASH_TYPEERASE_SECTORS o FLASH_TYPEERASE_MASSERASE.
- Bancos: este parámetro, que está disponible sólo en aquellas series STM32 que proporcionan una memoria flash interna de múltiples bancos, especifica el banco involucrado en un borrado masivo. Puede asumir los valores FLASH_BANK_1, FLASH_BANK_2 o FLASH_BANK_BOTH para eliminar ambos bancos.

Flash Memory Erasing

```
typedef struct {
  uint32_t TypeErase;   /* Mass erase or sector Erase */
  uint32_t Banks;       /* Select banks to erase when Mass erase is enabled */
  uint32_t Sector;      /* Initial FLASH sector to erase when Mass erase is disabled */
  uint32_t NbSectors;   /* Number of sectors to be erased */
  uint32_t VoltageRange;/* The device voltage range which defines the erase parallelism */
} FLASH_EraseInitTypeDef;
```

- Sector(Página): este campo se refiere a la identificación del sector involucrado en un borrado basado en sectores. Puede asumir el valor FLASH_SECTOR_0, FLASH_SECTOR_1 y así sucesivamente (el número máximo de sectores depende del microcontrolador específico). En aquellas MCU STM32 que proporcionan una memoria flash con granularidad de página, este campo se reemplaza por la primera dirección de la página involucrada en un procedimiento de borrado. Consulte el código fuente de CubeHAL para obtener más información sobre esto.
- NbSectors(NbPages): el número de sectores (páginas) que se borrarán a partir del Sector especificado.

Flash Memory Erasing

```
typedef struct {
  uint32_t TypeErase;   /* Mass erase or sector Erase */
  uint32_t Banks;       /* Select banks to erase when Mass erase is enabled */
  uint32_t Sector;      /* Initial FLASH sector to erase when Mass erase is disabled */
  uint32_t NbSectors;   /* Number of sectors to be erased */
  uint32_t VoltageRange;/* The device voltage range which defines the erase parallelism */
} FLASH_EraseInitTypeDef;
```

- VoltajeRange: incluso si estamos borrando un sector (o página) completo, en realidad el procedimiento de borrado recorre un subconjunto del mismo (normalmente dos bytes). Las MCU STM32 de mayor rendimiento permiten borrar varios bytes a la vez. Esta característica se llama paralelismo flash y está relacionada con el voltaje de funcionamiento de la MCU: cuanto mayor es VDD, más bytes se borran a la vez. Este campo puede asumir un valor de la Tabla 3. Sin embargo, consulte siempre el manual de referencia de su MCU para obtener más información sobre esto.

Flash Memory Erasing

Table 3: Program/erase parallelism depending on the voltage range

VoltageRange	Voltage range	*Parallelism*
FLASH_VOLTAGE_RANGE_1	1.7 - 2.1 V	8 bits at a time
FLASH_VOLTAGE_RANGE_2	2.1 - 2.4 V	16 bits at a time
FLASH_VOLTAGE_RANGE_3	2.4 - 3.6 V	32 bits at a time
FLASH_VOLTAGE_RANGE_4	2.7 - 3.6 V with External VPP	64 bits at a time

- HAL_FLASHEx_Erase() es una función de bloqueo: esperará hasta que se haya completado el procedimiento de borrado. Este puede ser un procedimiento bastante “largo”, dependiendo de la familia STM32, la velocidad HCLK, la cantidad de sectores/páginas involucradas en el borrado y el voltaje VDD en aquellas MCU STM32 que proporcionan paralelismo de programa/borrado.
  - Para evitar bloquear las actividades del firmware durante este procedimiento, HAL proporciona la función:
    - HAL_StatusTypeDef HAL_FLASHEx_Erase_IT(FLASH_EraseInitTypeDef *pEraseInit,uint32_t *SectorError);

Flash Memory Erasing

Table 3: Program/erase parallelism depending on the voltage range

VoltageRange	Voltage range	*Parallelism*
FLASH_VOLTAGE_RANGE_1	1.7 - 2.1 V	8 bits at a time
FLASH_VOLTAGE_RANGE_2	2.1 - 2.4 V	16 bits at a time
FLASH_VOLTAGE_RANGE_3	2.4 - 3.6 V	32 bits at a time
FLASH_VOLTAGE_RANGE_4	2.7 - 3.6 V with External VPP	64 bits at a time

- que realiza un procedimiento de borrado en modo interrupción. Podemos ser notificados del final del procedimiento de borrado habilitando la interrupción FLASH_IRQn e implementando el ISR correspondiente.

- Precaución: Se debe tener especial cuidado en caso de que estemos borrando la ubicación de la memoria flash que contiene el código del programa, especialmente si estamos borrando el primer sector/página que contiene la tabla de vectores (esto siempre es cierto si estamos realizando un borrado masivo). Si este es el caso, entonces necesitamos mover el código del programa y reubicar toda la tabla de vectores dentro de la SRAM, como se muestra en el Capítulo 15; de lo contrario, se producirá una falla una vez que se active la interrupción.

Flash Memory Programming

- Una vez borrado un sector/página, podemos proceder a programar su contenido.
- En teoría, es perfectamente posible acceder directamente a una ubicación flash para cambiar su contenido[6] escribiendo un código C como el siguiente:
  - . . .
  - *(volatile uint16_t*)0x0800AA00 = Data;
  - . . .
- Sin embargo, esto básicamente no es conveniente por dos razones principales.
  - En primer lugar, en algunas MCU STM32 es posible que se requieran operaciones preliminares (como configurar registros específicos) antes de que podamos programar una ubicación flash.

Flash Memory Programming

- En segundo lugar, dependiendo de la serie STM32 específica y el rango de voltaje VDD, la cantidad de bytes que se pueden transferir simultáneamente al flash puede diferir significativamente.
- Por estos motivos, HAL define la función:
  - HAL_StatusTypeDef HAL_FLASH_Program(uint32_t TypeProgram, uint32_t Address, uint64_t Data);
- que está diseñado para abstraer todos los detalles de implementación específicos. Analicemos los argumentos de la función:
  - TypeProgram: indica cuántos bytes se transfieren durante la operación de escritura, y puede asumir los valores FLASH_TYPEPROGRAM_HALFWORD, FLASH_TYPEPROGRAM_WORD y FLASH_- TYPEPROGRAM_DOUBLEWORD.

Flash Memory Programming

- Tenga en cuenta que este parámetro especifica solo la cantidad de datos transferidos utilizando la función HAL_FLASH_Program(). La cantidad efectiva de bytes transferidos en una sola transacción depende de la familia STM32 y del grado de paralelismo, si está disponible.
- Address : es la dirección de memoria inicial donde se empieza a colocar el contenido.
- Data : son los datos a almacenar dentro de la ubicación de la memoria flash (representados como una variable de doble palabra).

Al igual que el procedimiento de borrado visto anteriormente, es posible realizar un procedimiento de programación flash en modo interrupción utilizando la función:

- HAL_StatusTypeDef HAL_FLASH_Program_IT(uint32_t TypeProgram, uint32_t Address, uint64_t Data);

Flash Read Access During Programming and Erasing

- Un acceso de lectura a la memoria flash mientras se realiza una operación de borrado o escritura provocará una parada del bus, al menos en la mayoría de los microcontroladores STM32.
  - Esto significa que, si necesita realizar otras operaciones en paralelo, deberá reubicar el código a SRAM para ejecutarlo durante una operación de programación flash.
  - Un escenario típico está representado por un gestor de arranque personalizado (bootloader): podemos programar nuestro código para intercambiar el nuevo firmware para flashear usando el UART en modo interrupción o DMA.
  - Si este es el caso, no podemos perder eventos asincrónicos (por ejemplo, una interrupción que nos notifica una transferencia de datos) porque la MCU se detiene esperando la operación en curso.
  - Si es así, es mejor reubicar el código en SRAM (y eventualmente reubicar también la tabla de vectores).

Option Bytes

- Los Option Bytes son dos o más bytes cuyos bits son valores de configuración especiales.
  - El concepto de bytes de opción es similar al que se encuentra en otras arquitecturas de microcontroladores, como los fusibles de la serie AVR de Atmel o los Bits de Configuración que se encuentran en los microcontroladores PIC de Microchip.
  - Cada bit individual de estos bytes especiales en la región del Bloque de Información tiene un significado especial.
- El número y el tipo de parámetros de configuración dependen de la MCU STM32 específica.
  - Los parámetros de configuración más comunes están relacionados con:
    - BOOT: en la mayoría de microcontroladores STM32, dos bits de opción permiten seleccionar el origen del arranque (boot) (FLASH, Memoria del sistema o SRAM).

Option Bytes

- RDP: estos bits configuran el nivel de protección de lectura de la memoria flash y los analizaremos más en profundidad más adelante en este capítulo.
- BOR_LEVEL: estos bits contienen el umbral de nivel de suministro que activa/libera el reset. Se pueden escribir para programar un nuevo nivel BOR. De forma predeterminada, BOR está desactivado. Cuando la tensión de alimentación (VDD) cae por debajo del nivel BOR seleccionado, se genera un Reset del dispositivo.
- MCU behaviour when entering in some low-power modes: en casi todos los microcontroladores STM32 es posible configurar el MCU para que genere un Reset al entrar en los modos de bajo consumo stop o sleep.
- Hardware watchdog: en algunas MCU STM32, existen uno o dos bits que se utilizan para configurar WWDG e IWDG en "modo hardware", es decir, se inician automáticamente al resetea la MCU.

Option Bytes

- Flash write protection: estos bits permiten proteger individualmente contra escritura algunos sectores/páginas flash, evitando escribir en ellos incluso si la memoria flash está desbloqueada. Si un bit determinado se establece en "1", el sector/página correspondiente no está protegido contra escritura; si, en cambio, el bit se establece en "0", entonces el sector/página está protegido contra escritura.

- Para programar los bytes de opciones existe un procedimiento específico a seguir, que es independiente de la programación de toda la memoria flash. Por lo tanto, CubeHAL proporciona rutinas dedicadas para usar.
- En primer lugar, esta región debe desbloquearse llamando a la función:
  - HAL_StatusTypeDef HAL_FLASH_OB_Unlock(void);

Option Bytes

- A continuación, se programa completamente un byte de opción dado utilizando la función:
  - HAL_StatusTypeDef HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
- El valor de un byte de opción se modifica automáticamente borrando primero el bloque de información y luego programando todos los bytes de opción con los valores pasados a la rutina HAL_FLASHEx_OBProgram().
  - La función acepta una instancia de la estructura C FLASH_OBProgramInitTypeDef, cuyos campos representan el contenido del byte de opción dado.
  - Para obtener más información sobre el tipo exacto y el número de campos, consulte el código fuente de CubeHAL.

Option Bytes

- De manera similar, para recuperar el contenido de un byte de opción determinado usamos la función:
  - HAL_FLASHEx_OBGetConfig(FLASH_OBProgramInitTypeDef *pOBInit);
- Una vez modificado un byte de opción, tenemos que forzar a la MCU a recargar su contenido usando la función:
  - HAL_StatusTypeDef HAL_FLASH_OB_Launch(void);
- Tenga en cuenta que cambiar algunos bits de opción en particular MCU STM32 puede provocar un Reset del chip.
- Finalmente, el depurador ST-LINK y la utilidad ST-LINK relacionada brindan la capacidad de modificar fácilmente los Option Bytes.

Option Bytes

- Una vez que haya conectado el depurador ST-LINK a la MCU de destino, vaya al menú Destino->Bytes de opción en la utilidad ST-LINK.
  - Aparece el cuadro de diálogo Bytes de opción, como se muestra en la Figura 1.
  - La herramienta también permite borrar sectores/páginas flash seleccionados.

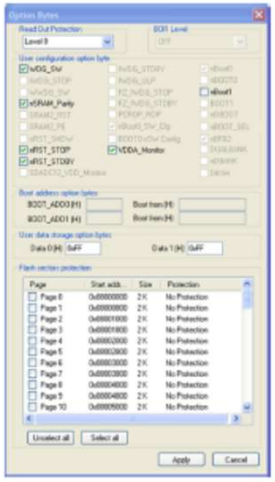


Figure 1: The *Option Bytes* configuration dialog in the ST-LINK Utility

Flash Memory Read Protection

- Precaución: Algunos procedimientos descritos en este párrafo pueden bloquear su microcontrolador impidiéndole flashearlo y borrarlo para siempre. Lea atentamente el contenido de este párrafo y evite realizar operaciones si no le quedan totalmente claras.
- Un byte de opción (llamado RDP) merece un párrafo aparte: el byte de configuración relacionado con la protección de lectura flash. Para evitar el acceso no deseado a la memoria flash a través de la interfaz de depuración, es posible desactivar temporal o permanentemente el acceso de lectura a esta memoria desde el mundo externo (claramente, el acceso desde el núcleo de la CPU y los controladores DMA siempre es posible).

Flash Memory Read Protection

- Existen tres niveles de protección, que corresponden a tres valores diferentes a almacenar en el byte de opción:
  - Nivel 0 (sin protección de lectura): cuando el nivel de protección de lectura se establece en el Nivel 0 escribiendo 0xAA en el byte de opción de protección de lectura (RDP), todas las operaciones de lectura/escritura (si no se establece ninguna protección contra escritura) desde/hacia la memoria flash o la SRAM de respaldo son posibles en todas las configuraciones de arranque (boot) (arranque de usuario flash, depuración o arranque desde RAM).

Flash Memory Read Protection

- Nivel 1 (protección de lectura habilitada): es el nivel de protección de lectura predeterminado después de borrar los bytes de opción (lo cual se realiza automáticamente mediante la rutina HAL_FLASHEx_OBProgram()). El nivel 1 de protección de lectura se activa escribiendo cualquier valor (excepto 0xAA y 0xCC utilizados para configurar el nivel 0 y el nivel 2, respectivamente) en el byte de opción RDP. Cuando se establece el Nivel 1 de protección de lectura, no se puede realizar ningún acceso (lectura, borrado, programación) a la memoria flash o a la SRAM de respaldo mientras el depurador está conectado o mientras se inicia desde la RAM o el cargador de arranque de la memoria del sistema.
- Se genera un error de bus en caso de solicitud de lectura. En cambio, al arrancar desde la memoria flash, se permiten accesos (lectura, borrado, programación) a la memoria flash y respaldo de SRAM desde el código de usuario.

Flash Memory Read Protection

- Cuando el Nivel 1 está activo, la programación del byte de opción de protección (RDP) en el Nivel 0 hace que la memoria flash y la SRAM de respaldo se borre en masa. Como resultado, el área del código de usuario se borra antes de que se elimine la protección contra lectura. El borrado masivo sólo borra el área del código de usuario.
- Los otros bytes de opción, incluidas las protecciones contra escritura, permanecen sin cambios desde antes de la operación de borrado masivo. El área OTP no se ve afectada por el borrado masivo y permanece sin cambios. El borrado masivo se realiza sólo cuando el Nivel 1 está activo y se solicita el Nivel 0. Cuando se aumenta el nivel de protección (0->1, 1->2, 0->2) no hay borrado masivo.
- Nivel 2 (!!!depuración/protección de lectura de chip permanentemente deshabilitada!!!): el Nivel 2 de protección de lectura se activa escribiendo 0xCC en el byte de opción RDP. Cuando se establece el nivel 2 de protección de lectura:

Flash Memory Read Protection

- – Todas las protecciones proporcionadas por el Nivel 1 están activas.
- – Ya no se permite arrancar desde la RAM.
- – Es posible iniciar el gestor de arranque de la memoria del sistema y no se puede acceder a todos los comandos excepto Get, GetID y GetVersion. Consulte AN2606.
- – JTAG, SWV (visor de un solo cable), ETM y escaneo de límites están deshabilitados.
- – Los bytes de opciones de usuario ya no se pueden cambiar.
- – Al arrancar desde la memoria Flash, se permiten accesos (lectura, borrado y programación) a la memoria Flash y respaldo SRAM desde el código de usuario.

La protección de lectura de memoria Nivel 2 es una operación irreversible. Cuando se activa el Nivel 2, el nivel de protección no se puede reducir al Nivel 0 o al Nivel 1. Solo para aclarar una vez más, esto significa que ya no podrá actualizar ni depurar su MCU.

Flash Memory Read Protection

- La Tabla 4 resume los efectos de un nivel de protección determinado en la memoria flash, los Option Bytes y la memoria OTP, cuando se accede a estas memorias mediante la interfaz del depurador, uno de los cargadores de arranque preprogramados, el código colocado en SRAM y en la memoria flash. Como puede ver, el Nivel 2 no impide que el código de usuario se escriba en la memoria flash (por ejemplo, un gestor de arranque personalizado -bootloader- aún puede programar la MCU).

Memory Area	Protection Level	Debug features, Boot from RAM or from System memory bootloader			Booting from Flash memory		
		Read	Write	Erase	Read	Write	Erase
Main Flash Memory and Backup SRAM	Level 0	YES			YES		
	Level 1	NO	NO	YES	YES		
	Level 2	**NO**			**YES**		
Option Bytes	Level 0	YES			YES		
	Level 1	YES			YES		
	Level 2	**NO**			**NO**		
OTP memory	Level 0	YES		N/A	YES		N/A
	Level 1	NO		N/A	YES		N/A
	Level 2	NO		N/A	YES		N/A

Table 4: The effects of read protection levels on the individual NVM memories

Optional OTP and True-EEPROM Memories

- Los microcontroladores STM32 más recientes y potentes proporcionan una memoria programable una sola vez (OTP).
  - Se trata de una memoria dedicada con un tamaño que va desde 512 hasta 1024 bytes con una característica única: una vez que un bit de esta memoria pasa de 1 a 0 ya no es posible restaurarlo a 1.
- Esto significa que esta región no se puede borrar.
  - Esta área de memoria es especialmente útil para almacenar parámetros de configuración relevantes conectados con el dispositivo dado, como números de serie, dirección MAC, valores de calibración, etc.Una práctica típica en la industria electrónica es producir dispositivos con diferentes funcionalidades a partir de la misma PCB o incluso de la misma placa completa. Esta área también podría usarse para almacenar los parámetros de configuración empleados por el firmware para adaptar las características de la placa.

Optional OTP and True-EEPROM Memories

- El área OTP se divide en N bloques de datos OTP de 32 bytes y un bloque OTP de bloqueo de N bytes.
  - Los datos OTP y los bloques de bloqueo no se pueden borrar.
  - El bloque de bloqueo contiene N bytes LOCKBi ($0 \leq i \leq N-1$) para bloquear el bloque de datos OTP correspondiente (bloques 0 a N).
  - Cada bloque de datos OTP se puede programar hasta que se programe el valor 0x00 en el byte de bloqueo OTP correspondiente (claramente, un bit individual ya establecido en 0 no se puede restaurar a 1).
  - Los bytes de bloqueo solo deben contener valores 0x00 y 0xFF; de lo contrario, es posible que los bytes OTP no se tengan en cuenta correctamente.

Optional OTP and True-EEPROM Memories

- La Tabla 5 muestra la organización de la memoria OTP en una MCU STM32F401RE y se extrae del manual de referencia relacionado.
  - Como puede ver, esta MCU proporciona 16 bloques de datos OTP, con un total de 512 bytes.
  - Dieciséis bytes de bloqueo permiten bloquear el bloque de datos OTP correspondiente.

Block	[128:96]	[95:64]	[63:32]	[31:0]	Address byte 0
0	OTP0	OTP0	OTP0	OTP0	0x1FFF 7800
	OTP0	OTP0	OTP0	OTP0	0x1FFF 7810
1	OTP1	OTP1	OTP1	OTP1	0x1FFF 7820
	OTP1	OTP1	OTP1	OTP1	0x1FFF 7830
. . .			. . .		. . .
15	OTP15	OTP15	OTP15	OTP15	0x1FFF 79E0
	OTP15	OTP15	OTP15	OTP15	0x1FFF 79F0
Lock block	LOCKB15 ... LOCKB12	LOCKB11 ... LOCKB8	LOCKB7 ... LOCKB4	LOCKB3 ... LOCKB0	0x1FFF 7A00

Table 5: The organization of the OTP memory in an STM32F401RE MCU

Optional OTP and True-EEPROM Memories

- Otra práctica común en electrónica digital es utilizar memorias EEPROM dedicadas y a menudo externas para almacenar parámetros de configuración.
  - Las memorias EEPROM tienen varias ventajas respecto a las memorias flash:
    - Sus bloques se pueden borrar individualmente.
    - Cada bloque se puede borrar hasta e incluso más de 1.000.000 de veces (los ciclos de borrado flash están limitados a 100.000 ciclos).
    - La vida útil nominal suele ser mayor que la de las memorias flash.
    - Suelen ser más económicas que las memorias flash (NOR y NAND).
    - Existen memorias EEPROM capaces de operar hasta 200°C.
- Sin embargo, el principal inconveniente de las memorias EEPROM es que suelen ser mucho más lentas que las memorias flash y ocupan espacio adicional en la PCB.

Optional OTP and True-EEPROM Memories

- Si su diseño tiene como objetivo reducir el costo de la lista de materiales, ST proporciona varias notas de aplicación que describen cómo emular una memoria EEPROM utilizando la memoria flash integrada STM32 (esta nota de aplicación se titula "Emulación EEPROM en microcontroladores STM32Fxx").
- Finalmente, varias MCU de la serie STM32L proporcionan una verdadera EEPROM embebida.
- Para obtener más información, consulte la hoja de datos de su MCU.

Flash Read Latency and the ART™ Accelerator

- En el Capítulo 1 vimos que los núcleos Cortex-M proporcionan un pipeline de instrucciones de n etapas diseñado para impulsar la ejecución del programa.
- Sin embargo, ese pipeline debe llenarse con instrucciones de máquina que normalmente se almacenan dentro de la memoria flash.
- Esta operación es un cuello de botella importante, porque las memorias flash son más lentas en comparación con la velocidad del reloj de la CPU.
- Si tanto la CPU como la memoria flash funcionan a la misma velocidad, la CPU puede alimentar su pipeline interno sin ninguna penalización.
- Por ejemplo, una MCU STM32F401RE que funciona a una velocidad de reloj inferior a 30 MHz puede acceder a la memoria flash sin demoras.
  - Desafortunadamente, en MCU de mayor rendimiento es necesario intercalar dos accesos sucesivos a la memoria flash con uno o más (en algunos casos incluso hasta diez) retrasos, llamados estados de espera.

Flash Read Latency and the ART™ Accelerator

- Los estados de espera corresponden a "esperas de ocupado" de hardware realizadas en uno o más ciclos de CPU y son una forma de sincronizar la CPU con la memoria flash más lenta.
- Los estados de espera reducen drásticamente el rendimiento efectivo de la CPU.
- Esta limitación generalmente se soluciona mediante el uso de memorias caché dedicadas.

Configurar la cantidad exacta de estados de espera necesarios es un paso crítico que depende de la MCU STM32 específica que esté considerando.

- Esta operación generalmente se realiza durante la configuración SYSCLK, porque cuanto mayor es la frecuencia de la CPU, más estados de espera se necesitan.

Flash Read Latency and the ART™ Accelerator

- Configurar la cantidad correcta de estados de espera es fundamental, especialmente cuando aumentamos la velocidad de la CPU: tenemos que configurar la cantidad correcta de estados de espera antes de aumentar la velocidad de la CPU; de lo contrario, se genera un BusFault.
- Sin embargo, CubeMX está diseñado para abstraer estos detalles y genera el código de configuración correcto según la MCU STM32 específica y la velocidad del núcleo deseada (consulte el código dentro de la rutina SystemClock_Config()).

- ST ha desarrollado una tecnología distintiva disponible en sus microcontroladores STM32 más potentes: el acelerador ART™. El acelerador ART™ es un conjunto de tecnologías de caché (consulte la Figura 2), externo al núcleo Cortex-M, que puede eliminar los efectos de los estados de espera.
  - El acelerador ART™ está diseñado para preservar la arquitectura Harvard de los microcontroladores Cortex-M, proporcionando grupos de caché separados para el I-Bus y el D-Bus.

Flash Read Latency and the ART™ Accelerator

- El Acelerador ARTTM está compuesto por:
  - un prefetch buffer de instrucciones;
  - un caché de instrucciones dedicado para reducir los efectos de la bifurcación;
  - una caché de datos for literal pools;
  - una política de planificación del bus AHB que facilita el acceso de la CPU al controlador flash a través del bus D-Bus.
- Analicemos el papel exacto de estas tecnologías.

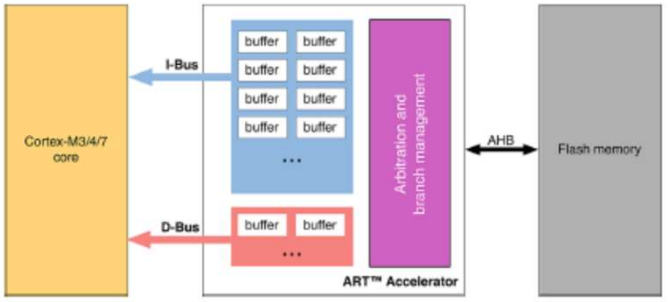


Figure 2: The main blocks forming the ART™ Accelerator

Flash Read Latency and the ART™ Accelerator

- The Instruction Prefetch Buffer.
  - Cuando la CPU accede a la memoria flash, no recupera un byte a la vez, pero generalmente lee desde 64 hasta 256 bits a la vez, dependiendo de la MCU STM32 específica.
  - Estos bits contienen un número variable de instrucciones y por esta razón se llaman líneas de instrucciones: suponiendo que la CPU lee 128 bits (esto es lo que sucede en las MCU STM32F4), esto puede contener cuatro instrucciones de 32 bits de ancho u ocho de 16 bits de ancho. instrucciones (depende si la CPU está funcionando en modo pulgar o no).
  - Entonces, en el caso de código secuencial, se necesitan al menos cuatro ciclos de CPU para ejecutar la línea de instrucción de lectura anterior.

Flash Read Latency and the ART™ Accelerator

- La captación previa en el bus I-Bus se puede utilizar para leer la siguiente línea de instrucción secuencial de la memoria flash mientras la CPU solicita la línea de instrucción actual.
- Esta característica es útil si se necesita al menos un estado de espera para acceder a la memoria flash.
- El búfer de captación previa de instrucciones se puede habilitar configurando la macro PREFETCH_ENABLE en 1 dentro del archivo stm32xxxx_hal_conf.h.

Flash Read Latency and the ART™ Accelerator

- The Instruction Cache Memory
  - El contenido del búfer de captación previa puede invalidarse debido a la bifurcación. Para limitar el tiempo perdido debido a los saltos, es posible retener un número determinado de líneas de instrucciones en una memoria caché de instrucciones.
  - Cada vez que se produce un error (los datos solicitados no están presentes en la línea de instrucción utilizada actualmente, en la línea de instrucción precargada o en la memoria caché de instrucciones), la línea leída se copia en la memoria caché de instrucciones.
  - Si la CPU solicita datos contenidos en la memoria caché de instrucciones, se los proporciona sin insertar ningún retraso.

Flash Read Latency and the ART™ Accelerator

- Una vez que se han llenado todas las líneas de la memoria caché de instrucciones "vacías", se utiliza una política de uso menos reciente (LRU) para determinar la línea que se reemplazará en la memoria caché de instrucciones.
- Esta característica es particularmente útil en el caso de código que contiene bucles.
- Esta característica se puede habilitar configurando la macro INSTRUCTION_CACHE_ENABLE en 1 dentro del archivo stm32xxxx_hal_conf.h, para aquellas MCU que proporcionan el acelerador ART™.

Flash Read Latency and the ART™ Accelerator

- Data Cache Memory
  - Las instrucciones de assembly a menudo mueven datos entre ubicaciones de memoria y registros de CPU.
  - En ocasiones, estos datos se almacenan dentro de la memoria flash (son valores constantes): en este caso, hablamos de literal pools.
  - Los grupos literal se obtienen de la memoria flash a través del bus D-Bus durante la etapa de ejecución de pipeline de la CPU. En consecuencia, el pipeline de la CPU se detiene hasta que se proporciona el literal pool solicitado.
  - Para limitar el tiempo perdido debido a los literal pools, los accesos a través del bus de datos AHB D-Bus tienen prioridad sobre los accesos a través del bus de instrucciones AHB I-Bus (esta es de hecho una política de arbitraje sobre el bus D-Bus).

Flash Read Latency and the ART™ Accelerator

- Además, existe una memoria caché de datos dedicada entre el bus D-Bus y la memoria flash.
- Este caché es más pequeño que el caché de instrucciones, pero ayuda a aumentar el rendimiento general de la CPU.
- Esta característica se puede habilitar configurando la macro DATA_CACHE_ENABLE en 1 dentro del archivo stm32xxxx_-hal_conf.h, para aquellas MCU que proporcionan el acelerador ART™.

The Role of the TCM Memories in STM32F7 MCUs

- La organización de la memoria de las MCU STM32F7 más recientes y potentes merece una mención aparte.
- De hecho, esta familia de microcontroladores se enfrenta a una organización de memoria y bus más compleja y flexible, ofreciendo dos interfaces distintas para acceder a memorias flash y SRAM: la Interfaz Extensible Avanzada (AXI), que es una especificación de bus ARM que interconecta el núcleo de la CPU a otros periféricos; la interfaz de memoria estrechamente acoplada (TCM) que interconecta el núcleo de la CPU con memorias volátiles y no volátiles acopladas directamente con él.
- Ambas interfaces, AXI y TCM, se enfrentan a una arquitectura Harvard, proporcionando líneas separadas para instrucciones (I-Bus) y datos (D-Bus).

The Role of the TCM Memories in STM32F7 MCUs

- Al observar la Figura 3, puede ver que el núcleo Cortex-M7 tiene tres rutas distintas para acceder al controlador flash (y por lo tanto a la memoria flash).
- Antes de describir estos tres caminos, es importante señalar una cosa fundamental: el núcleo Cortex-M7 ya proporciona un caché L1 integrado.
- Este caché tiene dos grupos de caché dedicados, cada uno de 64 KB de ancho, uno dedicado al I-Bus y otro al D-Bus: esto difiere de otras familias STM32, donde los cachés de datos e instrucciones se implementan exclusivamente dentro del Acelerador ART™.

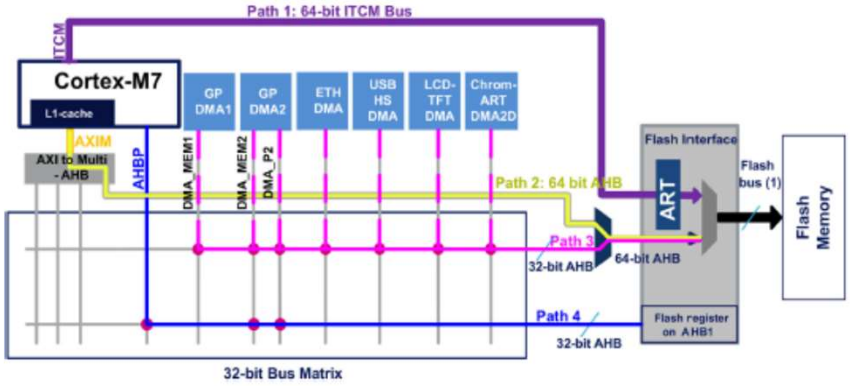


Figure 3: How the flash memory is accessed in an STM32F7 MCU

The Role of the TCM Memories in STM32F7 MCUs

- En todas las MCU STM32F7, se puede acceder a la memoria flash a través de tres interfaces principales para accesos de lectura y/o escritura:
  - Una interfaz ITCM de 64 bits: conecta la memoria flash incorporada al Cortex-M7 a través del bus ITCM (Ruta 1 en la Figura 3) y se utiliza para la ejecución del programa y el acceso de lectura de datos para valores literales.
    - A través de este bus no está permitido el acceso de escritura a la memoria flash.
    - La CPU puede acceder a la memoria flash a través de ITCM a partir de la dirección 0x0020 0000.
    - Al ser la memoria flash incorporada más lenta que el núcleo de la CPU, el acelerador ARTTM permite la ejecución de espera 0 desde la memoria flash a una frecuencia de CPU de hasta 216 MHz.

The Role of the TCM Memories in STM32F7 MCUs

- El acelerador STM32F7 ARTTM está disponible solo para acceso a memoria flash en la interfaz ITCM.
- Implementa una instrucción unificada y caché de rama de 256 bits x 64 líneas en los dispositivos STM32F74xxx y STM32F75xxx y 128/256 bits x 64 líneas en los dispositivos STM32F76xxx y STM32F77xxx siguiendo el modo de banco seleccionado.
- El Acelerador ART™ está disponible tanto para el acceso a instrucciones como a datos, lo que aumenta la velocidad de ejecución de código secuencial y bucles.
- El acelerador ART™ implementa también un búfer de prefetch de instrucciones.

The Role of the TCM Memories in STM32F7 MCUs

- Una interfaz AHB de 64 bits: conecta la memoria flash integrada al Cortex-M7 a través del puente AXI/AHB (Ruta 2 en la Figura 3).
  - Se utiliza para la ejecución de código, accesos de lectura y escritura.
  - La CPU puede acceder a la memoria flash a través del puente AXI/AHB a partir de la dirección 0x0800 0000 y es almacenable en caché (es decir, puede usar el caché L1) alcanzando el mismo rendimiento de espera 0 del acelerador ARTTM.
  - La memoria caché L1 en los núcleos Cortex-M7 puede oscilar entre 4 KB y 16 KB.
  - Las MCU STM32F74xxx y STM32F75xxx proporcionan dos grupos de caché, uno para las instrucciones (I-Bus) y otro para los grupos literales (D-Bus), cada uno de 4 KB de ancho.

The Role of the TCM Memories in STM32F7 MCUs

- En cambio, las MCU STM32F76xxx y STM32F77xxx proporcionan dos grupos de caché, cada uno de 16 KB de ancho.
- Los cachés L1 de todos los núcleos Cortex-M7 se dividen en líneas de 32 bytes.
- Cada línea está etiquetada con una dirección. La caché de datos es asociativa de 4 vías (cuatro líneas por conjunto) y la caché de instrucciones es asociativa de 2 vías.
- Se trata de un compromiso de hardware para evitar tener que etiquetar cada línea con una dirección.

- Una interfaz AHB de 32 bits: se utiliza para transferencias DMA desde la memoria flash (Ruta 3 en la Figura 3).
  - El acceso a la memoria flash del DMA se realiza a partir de la dirección 0x0800 0000.

The Role of the TCM Memories in STM32F7 MCUs

- Existe una cuarta ruta (consulte la Figura 3) a través de la interfaz del periférico de bus avanzado (AHBP) y está reservada para el acceso a registros periféricos flash dentro de la región asignada de periféricos 0x4000 0000.

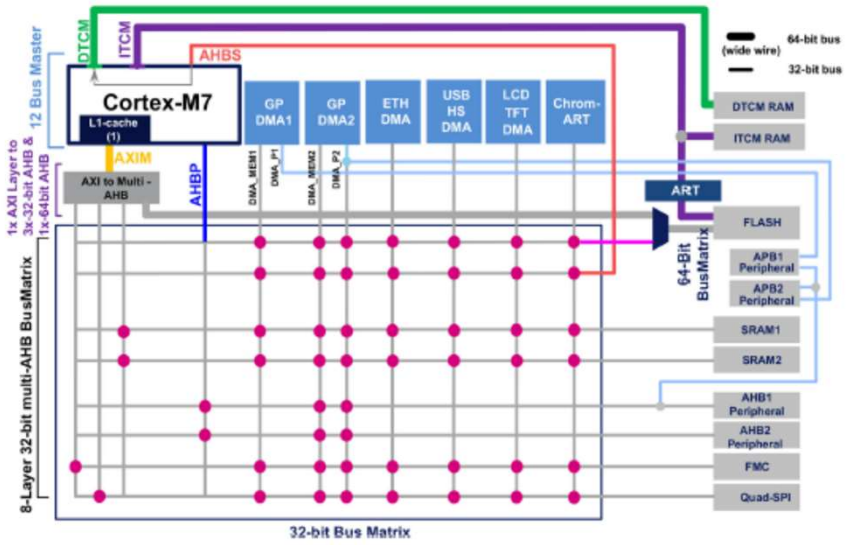


Figure 4: The bus matrix in an STM32F7 MCU

The Role of the TCM Memories in STM32F7 MCUs

- ¿Cuál es la ventaja de esta arquitectura aparentemente compleja?
  - Si ambas interfaces flash, es decir, AXI/AHB e ITCM, proporcionan ejecución de espera 0 (una gracias al caché L1 interno y otra gracias al acelerador ARTTM), ¿por qué deberíamos lidiar con esta complejidad durante el diseño del firmware?
- La respuesta proviene de la arquitectura de matriz de bus de una MCU STM32F7, que se muestra en la Figura 4.
  - Como puede ver, el bus AXI/AHB está conectado al caché L1 interno gracias a la interfaz AXIM.
  - Esto significa que los accesos a algunos periféricos del bus se pueden almacenar en caché. Y este es el caso de los controladores FMC y QuadSPI.

The Role of the TCM Memories in STM32F7 MCUs

- Gracias a esta arquitectura, es posible utilizar memorias NVM externas para almacenar datos o código de programa, aprovechando el caché L1 de 64K, mientras se tiene acceso paralelo (sin el arbitraje del bus) a la memoria flash interna a través de la interfaz ITCM y el Acelerador ART™.
- Esto supone un gran aumento de rendimiento para dispositivos que utilizan mucha memoria para almacenar imágenes, vídeos y contenidos multimedia en general, pero también grandes tablas de datos constantes, como FFT IV.

La capa CMSIS para MCU basadas en Cortex-M7 define un conjunto dedicado de rutinas para manipular la memoria caché L1 de Cortex-M7 (consulte la Tabla 6).

The Role of the TCM Memories in STM32F7 MCUs

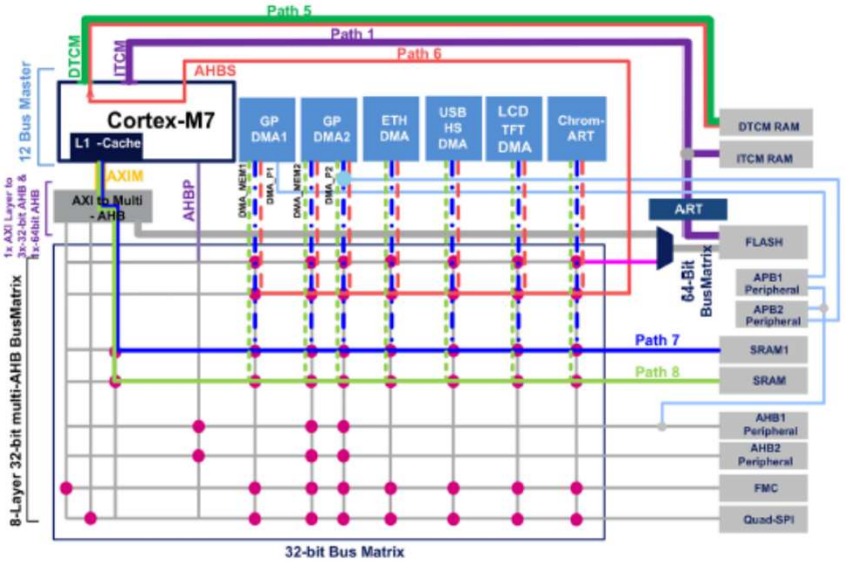


Figure 1: The four SRAM memories available in STM32F7 microcontrollers

- Si observamos la Figura 5, hay otra cosa importante a tener en cuenta.
  - Como puede ver, los microcontroladores STM32F7 ofrecen cuatro memorias SRAM distintas, accesibles a través de tres rutas separadas:

The Role of the TCM Memories in STM32F7 MCUs

- La RAM de instrucciones (ITCM-RAM), asignada en la dirección 0x0000 0000 y accesible sólo por el núcleo, es decir, a través de la Ruta 1 en la Figura 5.
  - Es accesible por bytes, medias palabras (16 bits), palabras (32 bits) o palabras dobles (64 bits). Se puede acceder a la RAM ITCM a una velocidad máxima de reloj de la CPU sin latencia.
  - La ITCM-RAM está protegida contra una contención de bus ya que solo la CPU puede acceder a esta región de RAM.
  - La ITCM-RAM desempeña el mismo papel que la memoria CCM en otras MCU STM32.

The Role of the TCM Memories in STM32F7 MCUs

- La RAM de datos (DTCM-RAM), asignada en la interfaz TCM en la dirección 0x2000 0000 y accesible por todos los maestros AHB desde la matriz del bus AHB: por la CPU a través del bus DTCM (Ruta 5 en la Figura 5) y por los DMA a través del "puente" AHBS específico en el núcleo Cortex-M7 (Ruta 6 en la Figura 5).
  - Es accesible por bytes, medias palabras (16 bits), palabras (32 bits) o palabras dobles (64 bits). Se puede acceder a la DTCM-RAM a una velocidad máxima de reloj de la CPU sin latencia.
  - Los accesos concurrentes a la DTCM-RAM por parte de los maestros (núcleo y DMA) y sus prioridades pueden ser manejados por el registro de control esclavo del núcleo Cortex-M7 (registro CM7_AHBSCR). Se puede dar una mayor prioridad a la CPU para acceder a la DTCM-RAM frente a los otros maestros (DMA). Para obtener más detalles sobre este registro, consulte el “Manual de referencia técnica del procesador ARM Cortex-M7”.

The Role of the TCM Memories in STM32F7 MCUs

- La SRAM1, accesible para todos los maestros AHB desde el bus Matrix de AHB, es decir, todos los DMA de propósito general así como los DMA dedicados.
  - Se puede acceder a la SRAM1 mediante bytes, medias palabras (16 bits) o palabras (32 bits).
  - Consulte la Figura 5 (Ruta 7) para conocer posibles accesos a SRAM1.
  - Se puede utilizar para la carga/almacenamiento de datos, así como para la ejecución de código (incluso si no ofrece ningún aumento de rendimiento específico).

The Role of the TCM Memories in STM32F7 MCUs

- La SRAM2, accesible para todos los maestros AHB desde la matriz de bus AHB. Todos los DMA de propósito general, así como los DMA dedicados, pueden acceder a esta región de memoria.
  - Se puede acceder a la SRAM2 mediante bytes, medias palabras (16 bits) o palabras (32 bits).
  - Consulte la Figura 5 (Ruta 8) para conocer posibles accesos a SRAM2. Se puede utilizar para la carga/almacenamiento de datos, así como para la ejecución de código (incluso si no ofrece ningún aumento de rendimiento específico).

The Role of the TCM Memories in STM32F7 MCUs

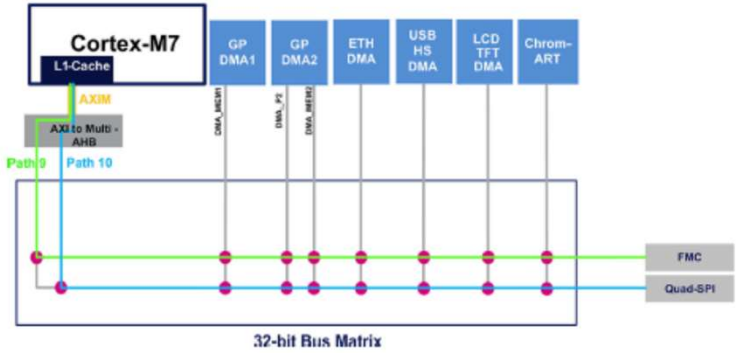


Figure 6: FMC and QuadSPI external memory controllers

- Además de las memorias flash internas y SRAM, los grupos de memoria STM32F7 se pueden ampliar utilizando el controlador de memoria flexible (FMC) y el controlador Quad-SPI. La Figura 6 muestra las rutas que conectan la CPU con estas memorias externas a través del bus AXI.
  - Como se muestra en la Figura 6, las memorias externas pueden beneficiarse del caché L1 de Cortex-M7, alcanzando el máximo rendimiento tanto durante la carga/almacenamiento de datos como durante la ejecución del código.

The Role of the TCM Memories in STM32F7 MCUs

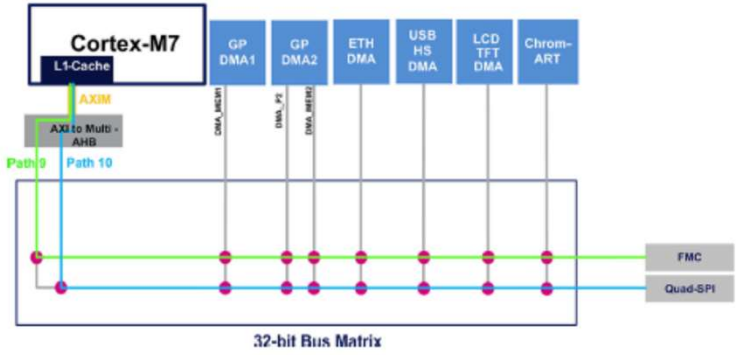


Figure 6: FMC and QuadSPI external memory controllers

- La memoria caché L1 Cortex-M7 ofrece una gran mejora de rendimiento a los microcontroladores STM32F7 en comparación con el STM32F4 con los mismos controladores de memoria externos.

- La Tabla 7 resume los tipos de memoria, tanto interna como externa a la MCU, disponibles en las MCU STM32F74xxx/STM32F75xxx.
  - La tabla muestra el tamaño de estas memorias, cómo están asignadas y la interfaz de bus utilizada para acceder a ellas.

The Role of the TCM Memories in STM32F7 MCUs

- Por ejemplo, puede ver que el rango de direcciones 0x0020 0000 - 0x002F FFFF permite acceder a la memoria flash interna a través de la interfaz ITCM, que se puede almacenar en caché gracias al acelerador ART.
- La Tabla 8 resume las mismas memorias para las MCU STM32F76xxx/STM32F77xxx (las características de FMC y QSPI son las mismas y, por lo tanto, no se enumeran en la Tabla 8).

Para obtener más información sobre estos temas, se recomienda encarecidamente consultar el AN4667 de ST.

The Role of the TCM Memories in STM32F7 MCUs

Memory Type	Memory region	Address range	Size	Cacheable	Access interfaces
Flash	FLASH-ITCM	0x0020 0000-0x002F FFFF	1 MB	YES (ART™)	ITCM (64-bit)
	FLASH-AXIM	0x0800 0000-0x080F FFFF		YES (L1-cache)	AHB (64-bit/32-bit)
RAM	DTCM-RAM	0x2000 0000-0x2000 FFFF	64 KB	YES (ART™)	DTCM (64-bit)
	ITCM-RAM	0x0000 0000-0x0000 3FFF	16 KB	YES (ART™)	ITCM (64-bit)
	SRAM1	0x2001 0000-0x2004 BFFF	240 KB	YES (L1-cache)	AHB (32-bit)
	SRAM2	0x2004 C000-0x2004 FFFF	16 KB	YES (L1-cache)	
FMC	NOR FLASH/RAM	0x6000 0000-0x6FFF FFFF	Up to 256MB	YES (L1-cache)	AHB (32-bit)
	NAND FLASH	0x8000 0000-0x8FFF FFFF		YES (L1-cache)	
	SDRAM1	0xD000 0000-0xDFFF FFFF		NO	
	SDRAM2	0xC000 0000-0xCFFF FFFF		NO	
Quad-SPI	QSPI FLASH	0x6000 0000-0x6FFF FFFF	Up to 256MB	YES (L1-cache)	AHB (4-bit/32-bit)

Table 7: Memory mapping and sizes in STM32F74xxx/STM32F75xxx MCUs

Memory Type	Memory region	Address range	Size	Cacheable	Access interfaces
Flash	FLASH-ITCM	0x0020 0000-0x003F FFFF	2 MB	YES (ART™)	ITCM (64-bit)
	FLASH-AXIM	0x0800 0000-0x081F FFFF		YES (L1-cache)	AHB (64-bit/32-bit)
RAM	DTCM-RAM	0x2000 0000-0x2001 FFFF	128 KB	YES (ART™)	DTCM (64-bit)
	ITCM-RAM	0x0000 0000-0x0000 3FFF	16 KB	YES (ART™)	ITCM (64-bit)
	SRAM1	0x2002 0000-0x2007 BFFF	368 KB	YES (L1-cache)	AHB (32-bit)
	SRAM2	0x2007 C000-0x2007 FFFF	16 KB	YES (L1-cache)	

Table 8: Memory mapping and sizes in STM32F76xxx/STM32F77xxx MCUs

How to Access Flash Memory Through the TCM Interface

- Una pregunta común para todos los principiantes de la plataforma STM32F7 es cómo aprovechar la interfaz TCM.
  - Este es claramente un trabajo del linker script, que tiene que reasignar las direcciones de las regiones .text, .bss y .data usando como direcciones base las reportadas en las Tablas 7 y 8.
- Sin embargo, esta operación no se puede realizar fácilmente cambiando la dirección inicial de la región FLASH dentro del linker script.
  - Esto porque, como se dijo anteriormente, no se permite el acceso en modo escritura a través de la interfaz ITCM.
  - Esto significa que OpenOCD, o cualquier depurador equivalente, no podrá cargar el código del programa usando el rango de direcciones 0x0020 0000 - 0x002F FFFF.

How to Access Flash Memory Through the TCM Interface

- Para solucionar esta limitación, debemos separar el rango de direcciones VMA del LMA, de la misma manera que lo hemos hecho para la región .data.
- Por ejemplo, el siguiente fragmento de secuencia de comandos del vinculador muestra cómo realizar esta operación.

```
/* Specify the memory areas */
MEMORY {
    ITCM_FLASH (rx): ORIGIN = 0x00200000, LENGTH = 1024K
    AXI_FLASH  (rx): ORIGIN = 0x08000000, LENGTH = 1024K
    RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 320K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >ITCM_FLASH AT>AXI_FLASH
```

```

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(4);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */

    KEEP (*(.init))
    KEEP (*(.fini))

    . = ALIGN(4);
    _etext = .;        /* define a global symbols at end of code */
  } >ITCM_FLASH AT>AXI_FLASH

  /* Constant data goes into FLASH */
  .rodata :
  {
    . = ALIGN(4);
    *(.rodata)         /* .rodata sections (constants, strings, etc.) */
    *(.rodata*)        /* .rodata* sections (constants, strings, etc.) */
    . = ALIGN(4);
  } >ITCM_FLASH AT>AXI_FLASH
```

How to Access Flash Memory Through the TCM Interface

- Como puede ver (consulte las líneas 17, 31 y 40), el rango de direcciones VMA (es decir, el rango de direcciones utilizado por la CPU para recuperar el código del programa) se asigna a la interfaz ITCM FLASH, mientras que el rango de direcciones LMA (es decir, el rango de direcciones utilizado para almacenar el programa en la memoria flash) se asigna a la interfaz AXI, lo que permite acceder a la memoria flash en modo de escritura.

Using CubeMX to Configure Flash Memory Interface

- CubeMX simplifica la configuración del bus utilizado para acceder a la memoria flash (TCM/AXI), del acelerador ARTTM y del caché L1 Cortex-M7.
  - Al ir a la sección Configuración y luego hacer clic en el botón Cortex-M7, es posible configurar estos parámetros, como se muestra en la Figura 7.

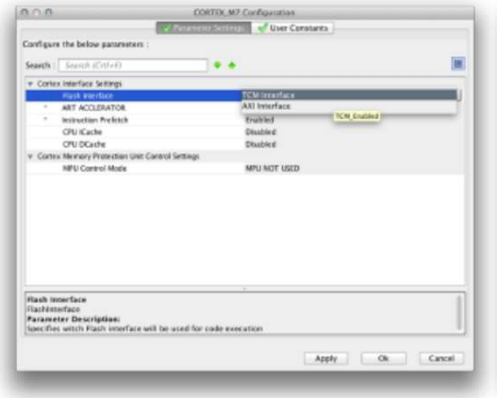


Figure 7: The Cortex-M7 configuration view in CubeMX

- NOTA: Tenga en cuenta que al momento de escribir este capítulo (agosto de 2016) el linker script generado es incorrecto porque no especifica direcciones LMA y VMA distintas, como se muestra en el párrafo anterior.

Referencias

- Mastering STM32 - A step-by-step guide to the most complete ARM Cortex-M platform, using a free and powerful development environment based on Eclipse and GCC - Carmine Noviello (Author)
- Programming with STM32: Getting Started with the Nucleo Board and C/C++ 1st Edición - Donal Norris (Author)
- Nucleo Boards Programming with the STM32CubeIDE, Hands-on in more than 50 projects - Dogan Ibrahim (Author)
- STM32 Arm Programming for Embedded Systems, Using C Language with STM32 Nucleo - Muhammad Ali Mazidi (Author), Shujen Chen (Author), Eshragh Ghaemi (Author)

Manos a la obra con el . . .

. . . Proyecto Intermedio

. . . un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**


Las y los estudiantes preguntarán:
¿en qué lío nos metimos?

¡Muchas gracias!

¿Preguntas?

. . .

Consultas a: jcruz@fi.uba.ar