Arquitectura del Microcontrolador

Temas

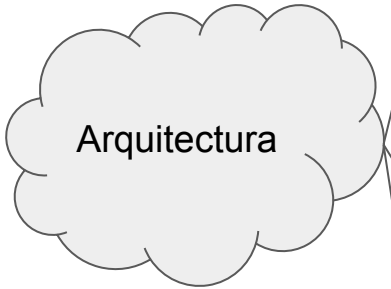
- Introducción a la arquitectura
- Procesador Cortex
- Registros
- Set de Instrucciones
- Buses y Memoria
- Modelo de memoria
- Manejo de Excepciones
- Tabla de vectores
- Clock
- Debug

Introducción - Arquitectura

¿Qué es la arquitectura?

- Veremos la Arquitectura de un µC representativo, con el que haremos las prácticas.
- La “arquitectura” es una descripción de las estructuras que componen un sistema y su interacción.
- Conocer la arquitectura permite:
  - Fase inicial: Tener los conocimientos básicos para trabajar con la herramienta
  - Fase avanzada: Obtener la mejor performance y depurar casos problemáticos
- Se describe desde distintos niveles de abstracción y desde distintas “vistas” que en conjunto describen el sistema completo.

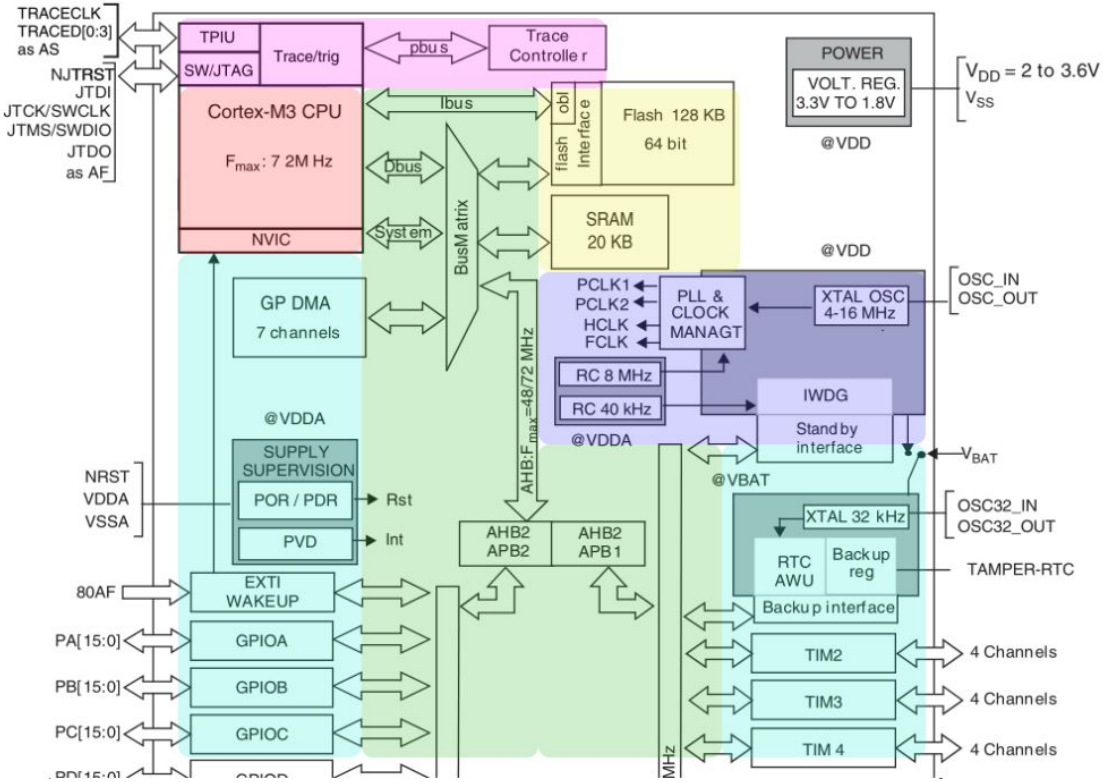
Arquitectura del micro -procesador -controlador



Set de Instrucciones	• Modelo de programación • Excepciones • Modelo de memoria • Arquitectura del set de Instrucciones (ISA)	• Estrategia de acceso a periféricos
Organización	• Registros del procesador • Estructura del procesador • Buses • “Periféricos” del procesador	• Complementos de manejo de Interrupciones • Estructura • Buses • Periféricos • Acceso a memoria • Reloj
Tecnología	• Implementación micro(nano)electrónica	• Implementación de la memoria • Características eléctricas

Caso de estudio: STM32F103C8

- Proc. ARM Cortex M3
- Manejo de Excepciones
- Memoria
- Características de Debug
- Buses
- Temporización
- Alimentación
- Periféricos



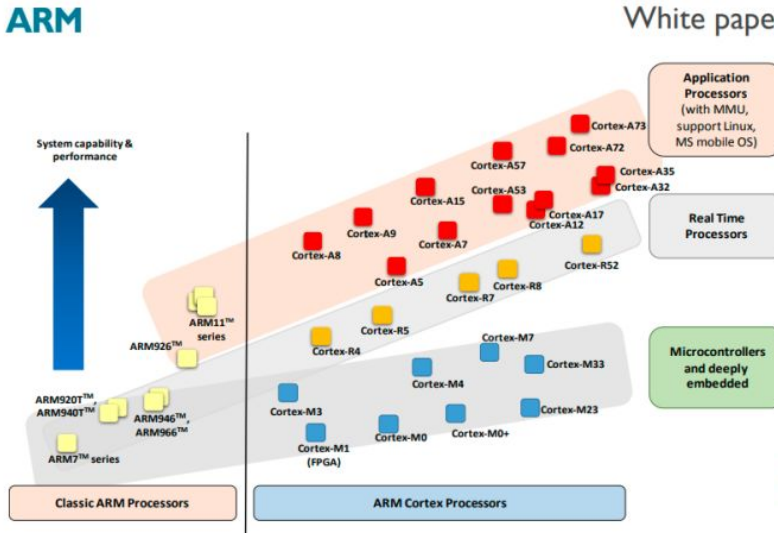
Procesador Cortex M3

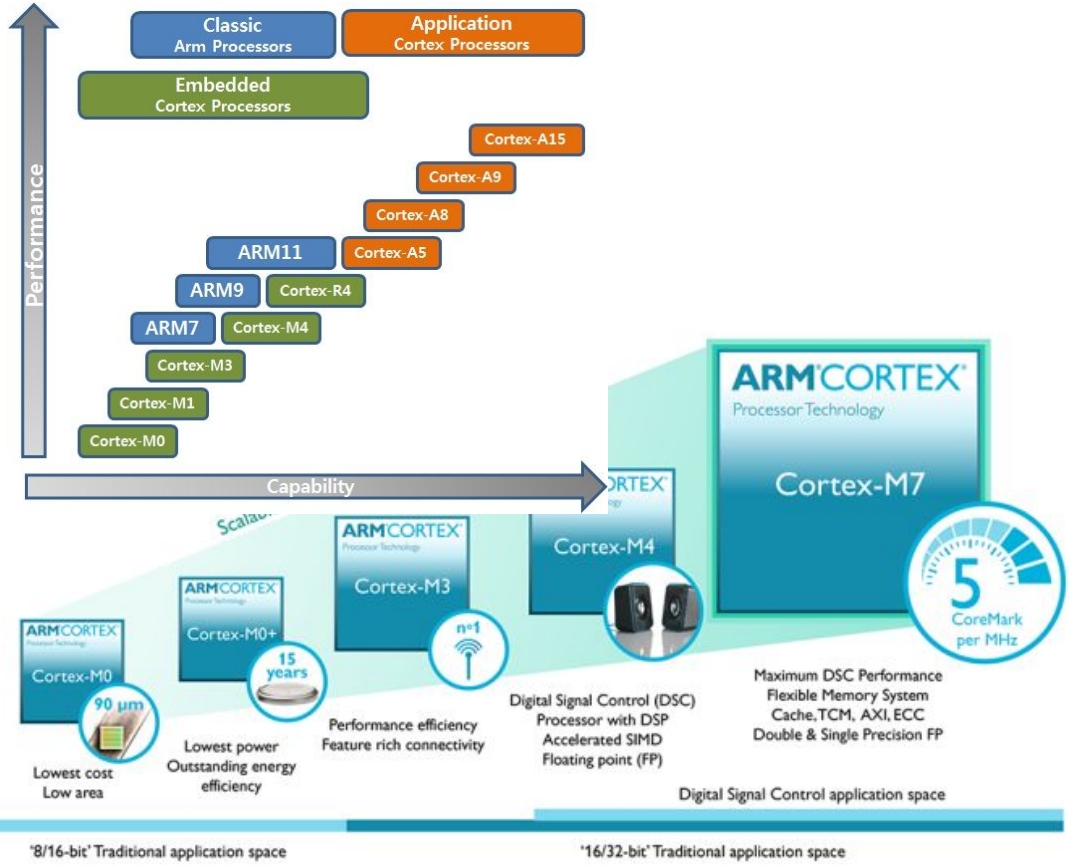
ARM

- ARM diseña micro**procesadores** y los licencia (no fabrica)
  - Por un lado, diseña arquitecturas de set de instrucciones, como la ARMv7-M
  - Por otro lado, diseña procesadores físicos o “cores” que pueden fabricarse
- ST le compra el diseño a ARM, le agrega buses y periféricos y así crea sus µCs
  - El diseño del procesador del **STM32F103** es un Cortex-M3 de ARM
  - Por lo tanto, al describir la arquitectura, hablaremos en parte de los **Cortex-M3** y habrá parte de la documentación que corresponderá a los Cortex
- Otros µCs
  - Basados en Cortex de ARM
    - STM32 (ST)
    - AVR32 (ATMEL)
    - MSP432 (TI)
    - LPC1700 (NXP)
  - Que no están basados en Cortex
    - ESP32 (Espressif)
    - PIC32 (Microchip)
    - DSPIC (Microchip)
    - C2000 (TI)
    - MSP430 (TI)

“Categoría” del Cortex-M3

- Al hablar de Cortex-M3 ya podemos posicionarnos en qué rango de dispositivo estaremos utilizando:

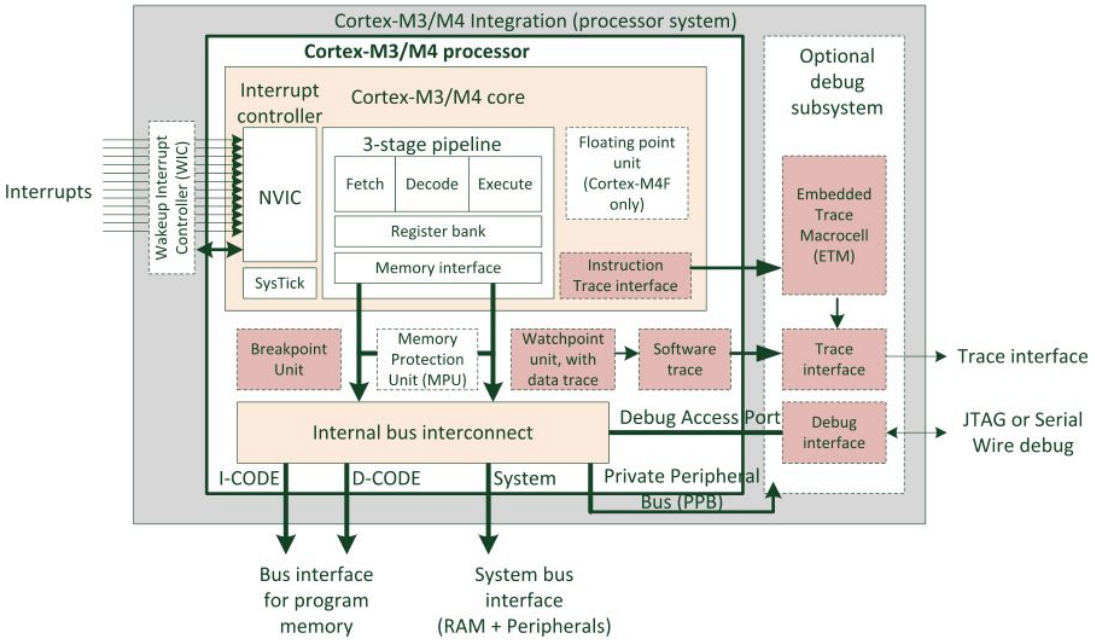




http://www.jkelec.co.kr/img/lecture/arm_arch/arm_arch_3.html

Arquitectura del Cortex-M3

- Procesador de 32 bits
  - Es decir que sus registros internos y sus buses son de 32 bits y por lo tanto puede direccionar una memoria de 4 GB
  - Arquitectura Harvard
  - Pipeline de 3 etapas
- Set de Instrucciones RISC
  - Sigue el paradigma de un set “reducido” de instrucciones
  - En particular la arquitectura de instrucciones es **ARMv7-M** que privilegia lograr un tamaño de código reducido y operación determinística por sobre la performance
  - Utiliza instrucciones Thumb 2 en su mayoría de 16 bits y que pueden ejecutarse casi todas en el equivalente de 1 ciclo de reloj
- Modelo de memoria lineal
  - Todo lo ve organizado dentro del mapa de memoria de 4 GB

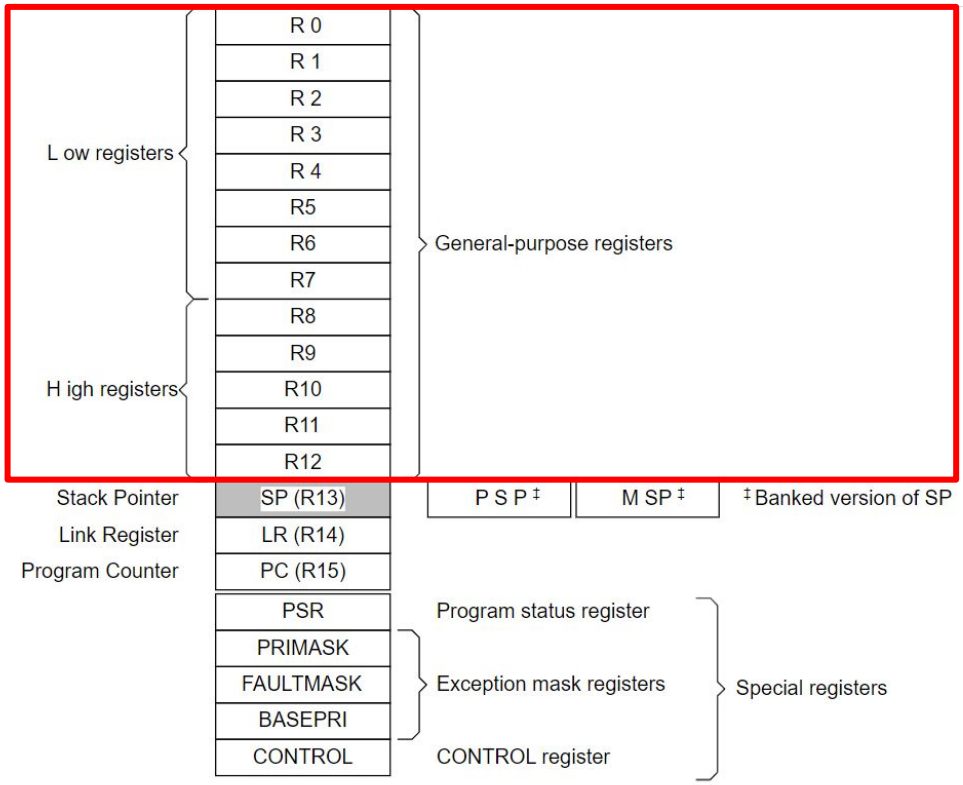


Joseph Yiu - The Definitive Guide to ARM Cortex -M3 and Cortex-M4 Processors

Registros

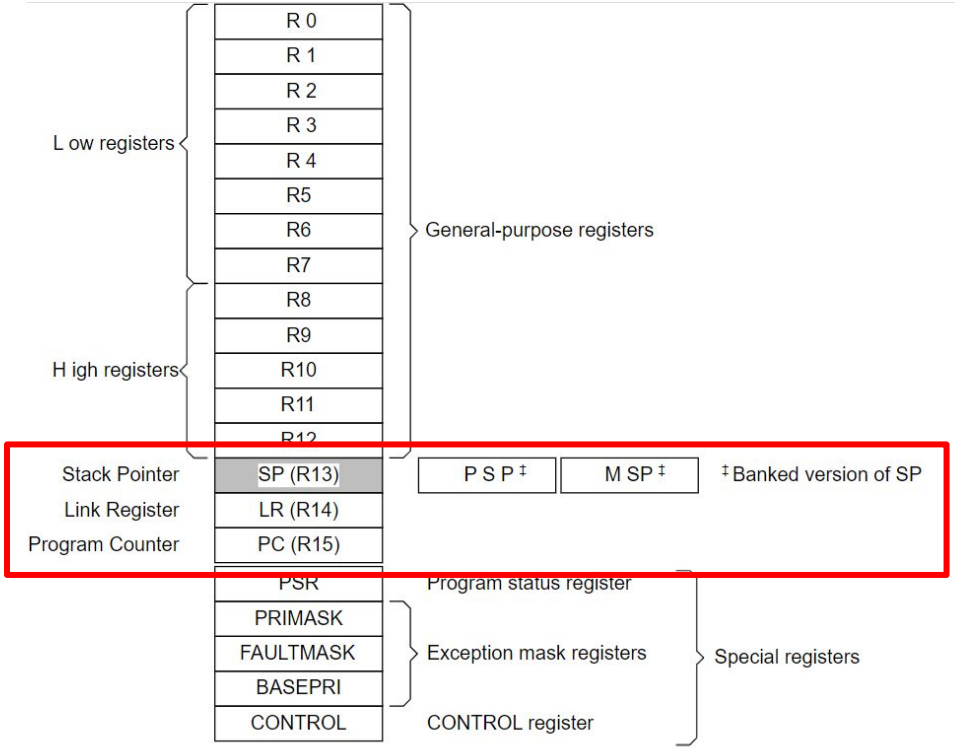
Registros del procesador

- 16 registros + regs. especiales
- Los regs. de propósito general
  - Debido al set RISC se usa arq. Load-Store y todas las instrucciones se desarrollan dentro de estos registros (salvo load y store)
  - Debido al uso de instrucciones de 16 bits, y a las capacidad de ejecución de las instrucciones, hay “poco espacio” sobrante en cada instrucción y algunas sólo pueden direccionar con 3 bits, y por eso la distinción de “low” y “high” registers



Registros de aplicaciones

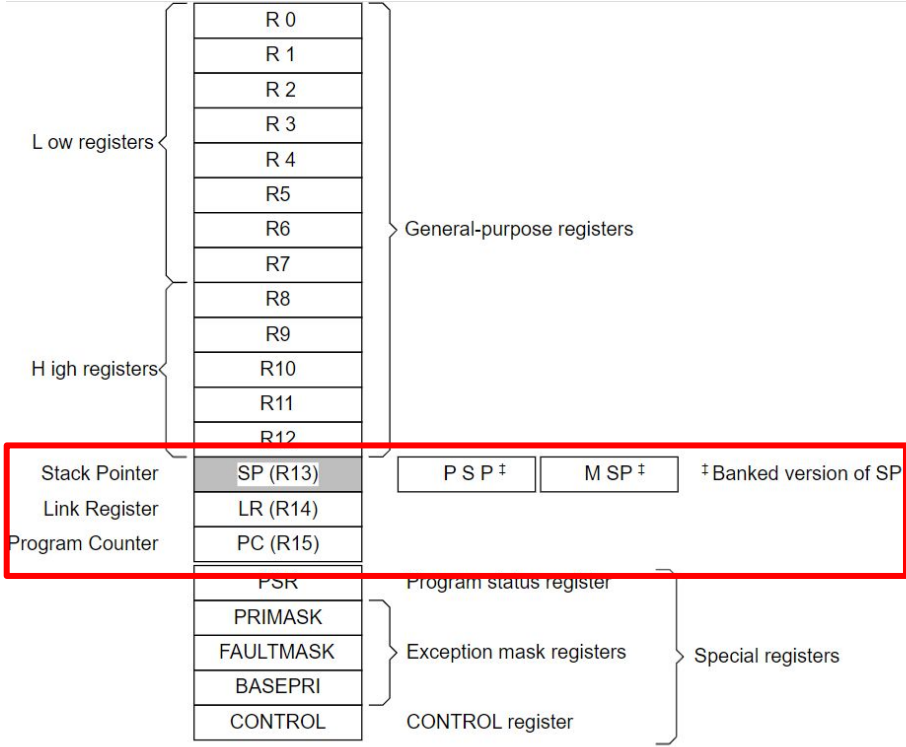
- Stack pointer o “R13”
  - Sirve para guardar los registros al cambiar de contexto
  - Es “full descending”: Al iniciar la ejecución se apunta **a la posición más alta** de la memoria RAM y se decrementa al hacer los push.
  - Tiene 2 “bancos” o registros físicos. Según el estado de ejecución muestra el “Main SP” o el “Process SP”
  - El PSP se usa con OSs
  - Los últimos 2 bits son siempre 0 (push y pop son siempre de 32 bits)



ST PM0056 STM32F10xxx/20xxx/21xxx/L1xxxx
Cortex®-M3 programming manual

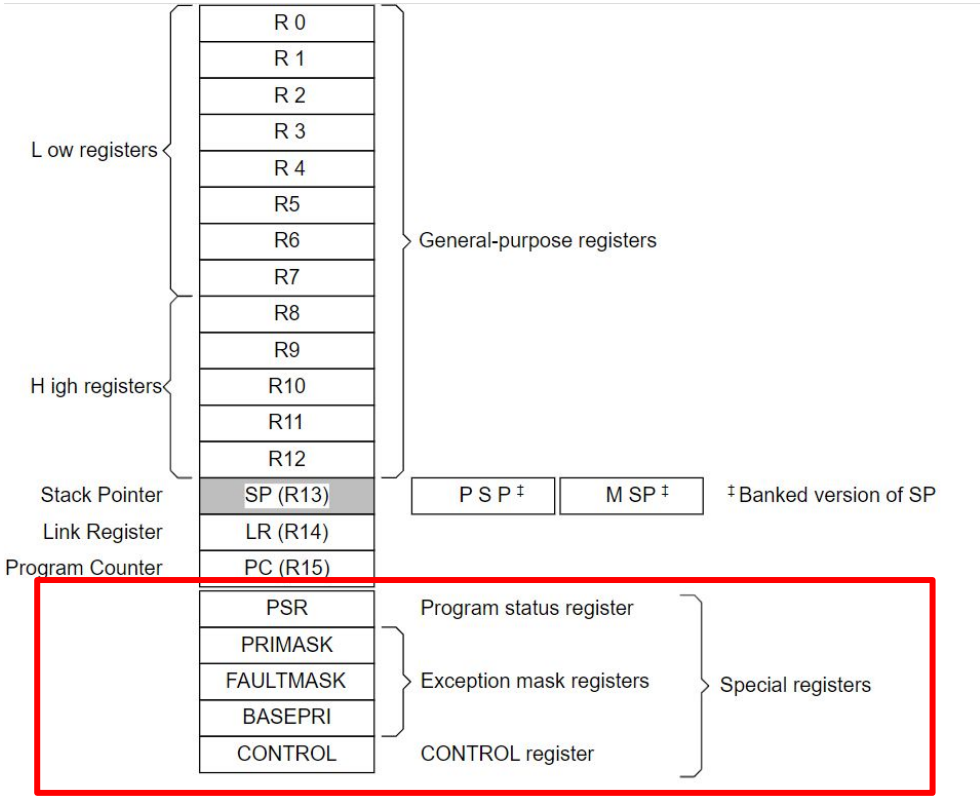
Registros de aplicaciones

- Link register
  - Guarda el salto de retorno al cambiar de contexto con la instrucción “branch with link”
  - Cuando se atiende una excepción almacena un código especial
- Program counter
  - Se inicializa en la dir. 0x00000004 donde está la “tabla de vectores” <- el comienzo de todo! (en la 0 está la dirección del stack)
  - Se puede alinear a 32 o 16 bits. Por lo tanto el LSB sería 0, pero debe escribirse siempre a 1 para indicar “thumb state”



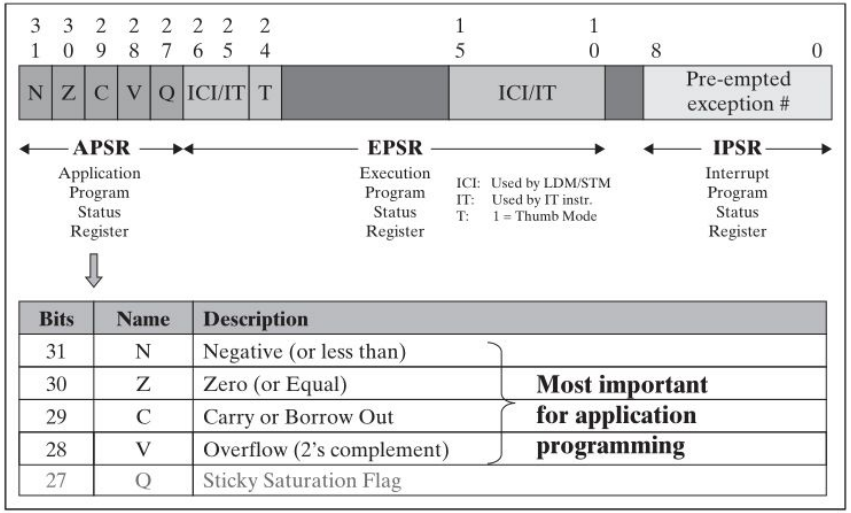
Registros Especiales

- Sirven para conocer y controlar el estado de ejecución del código
- Se accede con instrucciones
  - MRS <reg>, <special_reg>;
    - Read special register into register
  - MSR <special_reg>, <reg>;
    - write to special register
  - Estas instrucciones pueden “bloquearse” en un modo privilegiado y de esa manera se implementa un rudimentario control de acceso



Registros Especiales

- El PSR tiene 3 partes
  - Application PSR:
    - Contiene los resultados de las operaciones y es utilizado constantemente en cualquier aplicación
  - Execution PSR:
    - Le sirve al procesador para asentar el estado durante instrucciones que conllevan varios pasos secuenciales
  - Interrupt PSR
    - Se relaciona con el manejo de excepciones



Bits	Name	Description	
31	N	Negative (or less than)	
30	Z	Zero (or Equal)	**Most important**
29	C	Carry or Borrow Out	**for application**
28	V	Overflow (2's complement)	**programming**
27	Q	Sticky Saturation Flag	

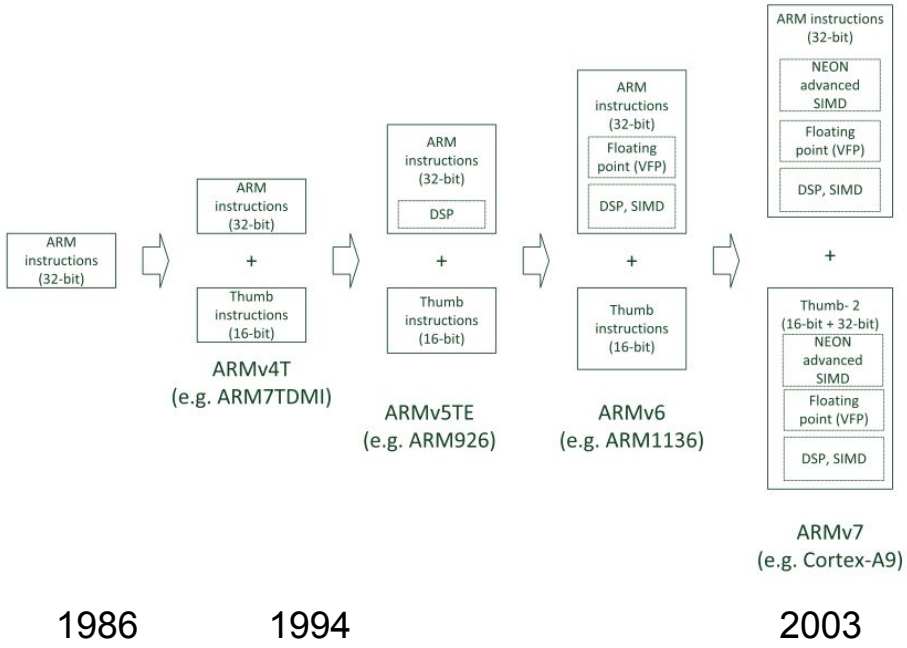
FIGURE 5-12 Bit fields within the Program Status Register.

D.W. Lewis - Fundamentals of Embedded Software with the Arm Cortex M3 (2012)

Set de Instrucciones

Set de Instrucciones thumb-2

- Instruction Set Architecture (ISA) **ARMv7-M**
  - Documentación: ARMv7-M Reference Manual
- Son de tipo Thumb 2, es decir que mezclan 16 y 32 bits.
  - Una arq. de 32 bits permitiría tener instrucciones largas y complejas
  - Pero la filosofía es RISC, por lo tanto se tendrán pocas instr. sencillas
  - Entonces, ARM decidió compactarlas en 16 bits
  - Pero 16 bits puede ser muy restrictivo! (por ejemplo, sólo podrían usarse los registros Low)
  - Se implementó una mezcla: El modo Thumb, y luego Thumb 2.
  - Algunos proc. pueden operar en un modo u otro (llamados ARM o Thumb).
  - La arq. -M que privilegia tamaño y determinismo, sólo opera en modo Thumb y prevalecen las instrucciones de 16 bits. Por compatibilidad (o portabilidad) se mantiene el bit de modo thumb.
- Al ingresar al procesador, todas las instrucciones se expanden a 32 bits



Manejo de datos

- ISA ARMv7-M soporta los siguientes tipos en memoria:
  - Byte 8 bits
  - Halfword “media palabra” 16 bits
  - Word “palabra ”32 bits
- Existen instrucciones para los siguientes tipos de datos en los registros:
  - Punteros de 32 bits
  - Enteros con o sin signo de 32 bits
  - Enteros sin signo de 16 o 8 bits almacenados rellenando con 0s
  - Enteros con signo de 16 o 8 bits almacenados con extensión de signo
  - Enteros de 64 bits con o sin signo almacenados en 2 registros

Direccionamiento

- El direccionamiento de memoria en las instrucciones puede darse a través de varios métodos, entre ellos:
  - El contenido de registros (incluyendo a veces un shift opcional)
  - Valores constantes
  - Valores relativos al PC (un método que el compilador usa comúnmente)
  - Valores relativos al SP
- El formato es Little Endian y aunque puede ser configurable, es el modo más común y así lo encontraremos en la documentación de ST

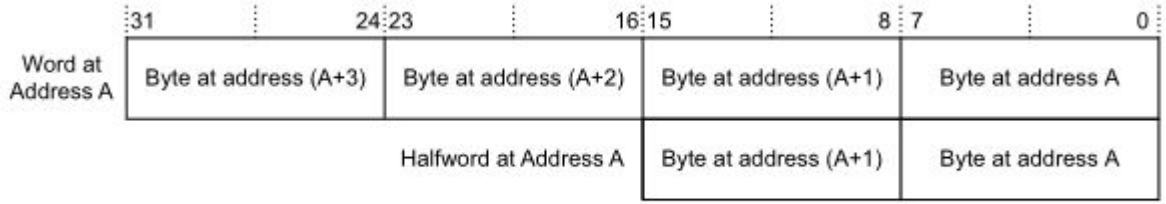


Figure A3-1 Little-endian byte format

Cortex M3 Reference Manual, ARM DDI 0403D

Operaciones

- El CPU puede realizar varias operaciones de bajo nivel en paralelo
- Las instrucciones tienen operandos Rm, Rn y destino Rd
- Existe un “extensor de signo” para expandir los operandos a 32 bits
- También un barrel shifter que permite hacer el desplazamiento que se quiera en 1 ciclo (que además puede ayudar a la ALU a realizar multiplicaciones)
- Un “memory address calculator” ayuda a calcular los direccionamientos y un sistema de incremento permite ir avanzando sin recalcular

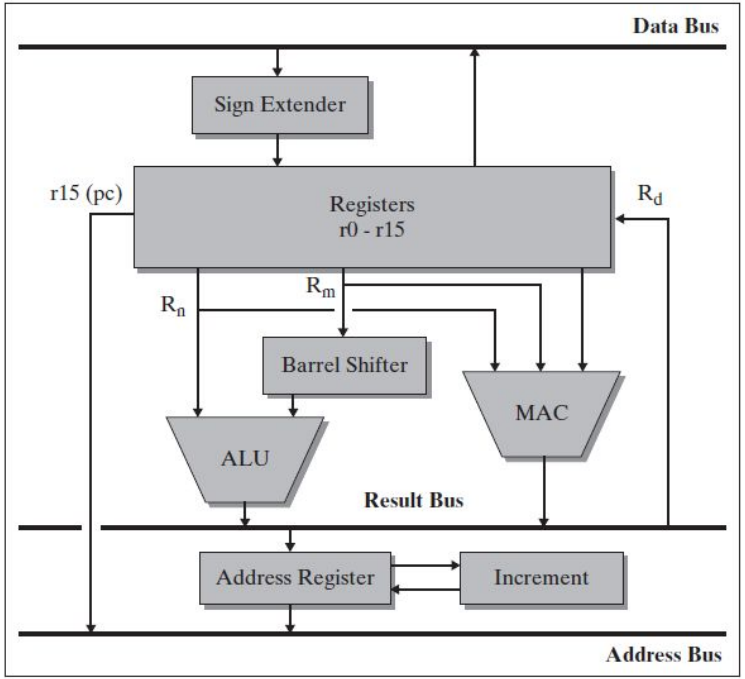


FIGURE 5-11 Internal organization of the ARM Cortex-M3 v7M CPU.

D.W. Lewis - Fundamentals of Embedded Software with the ARM Cortex M3 - Pearson (2012)

Formato de las instrucciones

- Todas las instrucciones incluyen un campo para hacer condicional su ejecución según los flags de resultados anteriores (mirando el Application PSR)
- Recordar que todas las operaciones se hacen sobre registros (R0-R12) del procesador salvo LOAD y STORE
- En algunas instrucciones se puede almacenar un valor constante de hasta 8 bits, que junto con la posibilidad de un shift permite implementar un rango amplio de valores “de consumo instantáneo”
- Algunas instrucciones incluyen un “desplazamiento instantáneo”

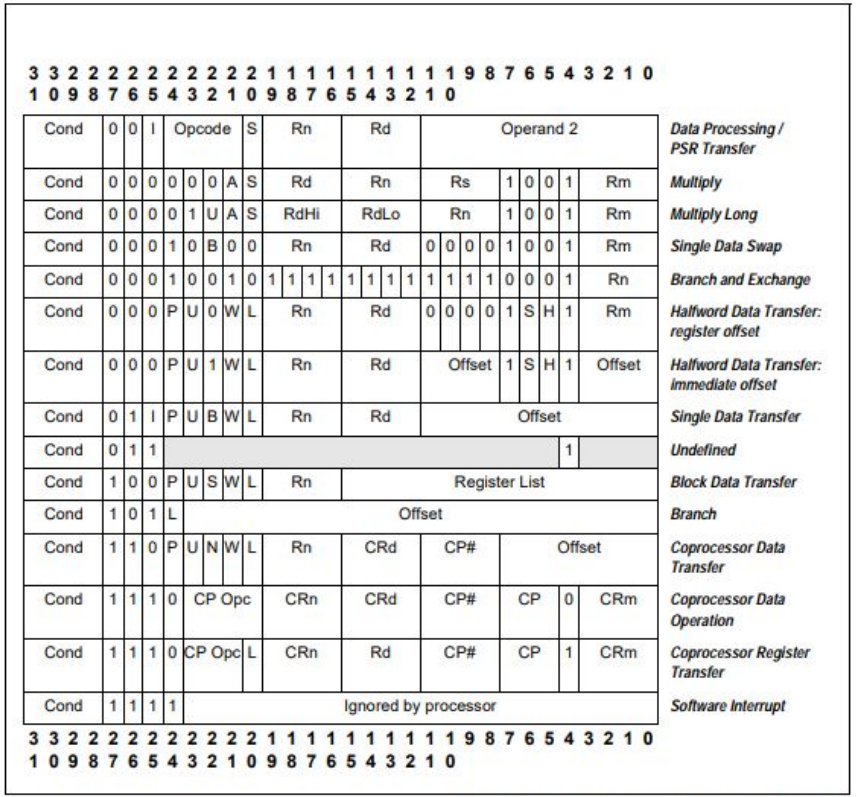


Figure 4-1: ARM instruction set formats

Resumen de instrucciones - Matemática y lógica

Suma y resta

ADC, ADCS	Add with Carry
ADD, ADDS, ADDW	Add
SUB, SUBS, SUBW	Subtract
SBC, SBCS	Subtract with Carry
RSB, RSBS	Reverse Subtract

Multiplicación y división de enteros

MUL, MULS	Multiply, 32-bit result
UMULL, SMULL	Un/Signed Multiply (32 x 32), 64-bit result
MLA	Multiply with Accumulate, 32-bit result
MLS	Multiply and Subtract, 32-bit result
SMLAL	Signed Multiply with Accumulate (32 x 32 + 64),
UMLAL	Unsigned Multiply with Accumulate (32 x 32 + 64),
UDIV, SDIV	Un/Signed Divide
USAT, SSAT	Un/signed Saturate

Manipulación de bits/bytes

ASR, ASRS	Arithmetic Shift Right
CLZ	Count Leading Zeros
SXTB	Sign extend a byte
SXTH	Sign extend a halfword
UXTB	Zero extend a Byte
UXTH	Zero extend a Halfword
RBIT	Reverse Bits
REV	Reverse byte order in a word
REV16	Reverse byte order in each halfword
REVSH	Reverse byte order in bottom halfword and
ROR, RORS	Rotate Right
RRX, RRXS	Rotate Right with Extend
BIC, BICS	Bit Clear

Lógica

AND, ANDS	Logical AND
EOR, EORS	Exclusive OR
ORN, ORNS	Logical OR NOT
ORR, ORRS	Logical OR
LSL, LSLS	Logical Shift Left
LSR, LSRS	Logical Shift Right
CMN	Compare Negative
CMP	Compare
TEQ	Test Equivalence
TST	Test

Bit field

BFC	Bit Field Clear
BFI	Bit Field Insert
SBFX	Signed Bit Field Extract
UBFX	Unsigned Bit Field Extract

Resumen de instrucciones - Manipulación de memoria

Cargar

ADR	Load PC-relative Address
LDR	Load Register with word
LDRB, LDRBT	Load Register with byte
LDRSB, LDRSBT	Load Register with Signed Byte
LDRD	Load Register with two bytes
LDRH, LDRHT	Load Register with Halfword
LDRSH, LDRSHT	Load Register with Signed Halfword
LDRT	Load Register with word
LDM	Load Multiple registers, increment after
LDMDB, LDMEA	Load Multiple registers, decrement before
LDMFD, LDMIA	Load Multiple registers, increment after

Guardar

STR	Store Register word
STRT	Store Register word
STRB, STRBT	Store Register byte
STRH, STRHT	Store Register Halfword
STRD	Store Register two words
STM	Store Multiple registers,
STMDB, STMEA	Store Multiple registers,
STMFD, STMIA	Store Multiple registers,

Mover

MOV, MOVS	Move
MOVT	Move Top
MOVW, MOV	Move 16-bit constant
MRS	Move from Special Register to general
MSR	Move from general register to Special
MVN, MVNS	Move NOT
POP	Pop registers from stack
PUSH	Push registers onto stack

Sincronización

LDREX	Load Register Exclusive
LDREXB	Load Register Exclusive with
LDREXH	Load Register Exclusive with
CLREX	Clear Exclusive

STREX	Store Register Exclusive
STREXB	Store Register Exclusive Byte
STREXH	Store Register Exclusive Halfword

Resumen de instrucciones - Control de flujo y ejecución

- Table branch
  - Se le suministra la dirección de una tabla y un índice. Cada entrada de la tabla contiene un salto relativo al PC. Equivale a una “selectiva múltiple”
- If-then
  - En lugar de hacer saltos, esta instrucción indica si ejecutar o no las próximas instrucciones (hasta 4)
  - EJ:

```
ITETT (cond)
instr1 <- esta si true
instr2 <- esta si false
instr3 <- esta si true
instr3 <- esta si true
```

- CPSID, CPSIE
  - Modifican los registros especiales de enmascaramiento de excepciones

Saltos

B	Branch
BL	Branch with Link
BLX	Branch indirect with Link
BX	Branch indirect
CBNZ	Compare and Branch if Non Zero
CBZ	Compare and Branch if Zero
TBB	Table Branch Byte
TBH	Table Branch Halfword

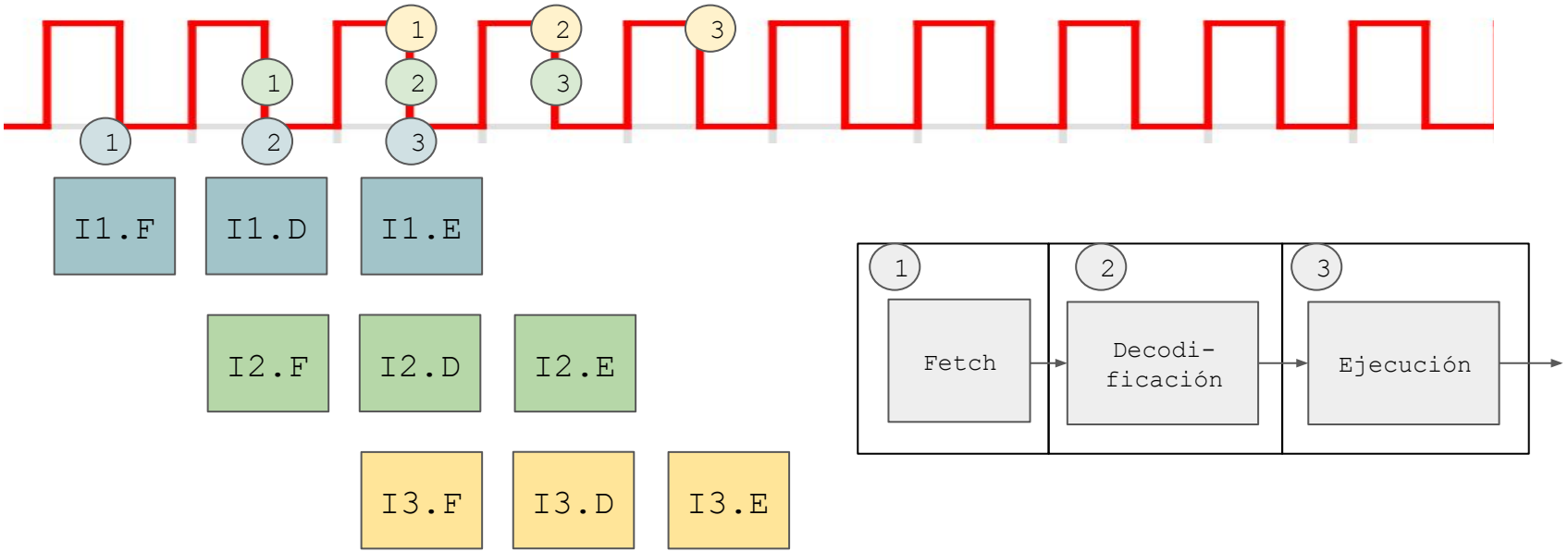
Control de ejecución

BKPT	Breakpoint
CPSID	Change Processor State, Disable
CPSIE	Change Processor State, Enable
DMB	Data Memory Barrier
DSB	Data Synchronization Barrier
ISB	Instruction Synchronization
NOP	No Operation
SEV	Send Event
SVC	Supervisor Call
IT	If-Then condition block
WFE	Wait For Event
WFI	Wait For Interrupt

https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set

Pipeline

Tiempo de ejecución - Pipeline



“Problemas” del pipeline - con soluciones parciales

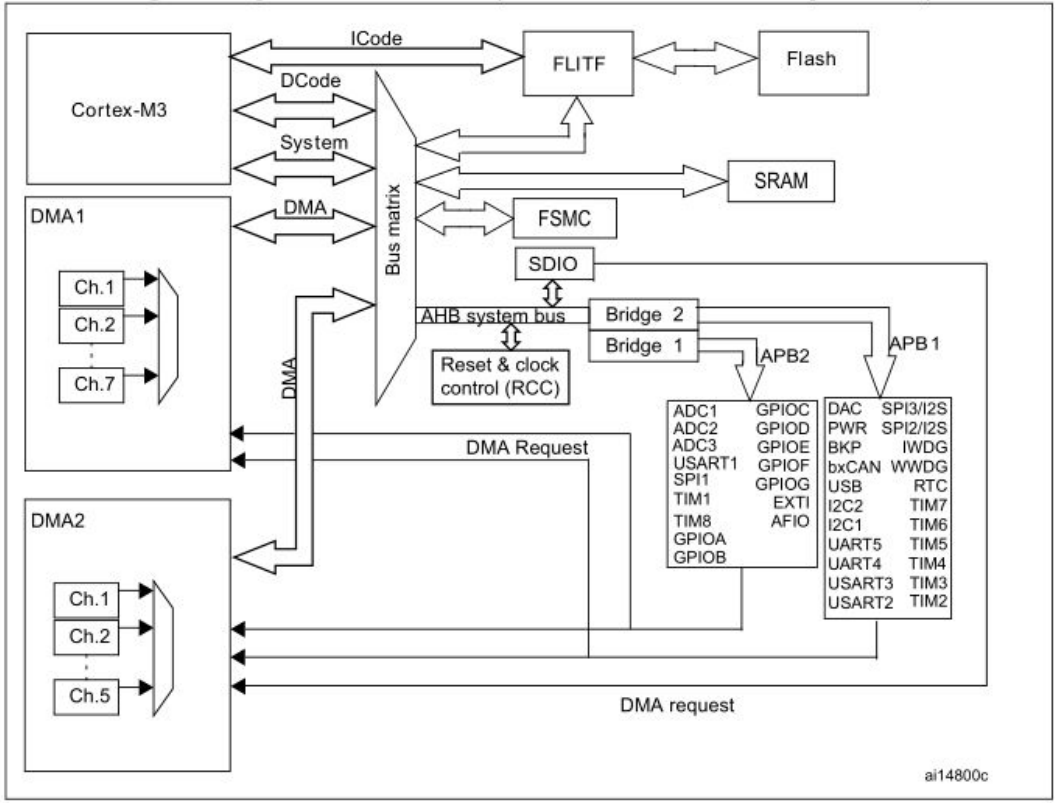
- No siempre pueden completarse las instrucciones del pipeline a tiempo
  - Algunas instrucciones demoran más ciclos
  - Los cambios de contexto, si bien tratan de predecirse, “desperdician” instrucciones
- Hay algunos peligros asociados al pipeline
  - Por ejemplo si la decodificación de una instrucción depende de que se complete la ejecución de la instrucción previa (e.g. MPU, prioridad de excepciones)
  - Para evitarlo existen instrucciones de “barrera”
- Velocidad de acceso a memoria
  - Los pipeline permiten ejecutar accesos a gran velocidad
  - Pero la memoria flash tiene un límite, y **ambos valores están tecnológicamente en puja**
  - Se introducen wait states para esperar a que las instrucciones o los datos estén disponibles
  - *- 0 wait states, if 0 < SYSCLK ≤ 24 MHz*
    - 1 wait state, if 24 MHz < SYSCLK ≤ 48 MHz
    - 2 wait states, if 48 MHz < SYSCLK ≤ 72 MHz
  - Se introducen caches de instrucciones para acceder más rápidamente

Memoria

Acceso a memoria

- Acceso a memoria de programa (Flash)
  - De instrucciones por ICode
  - De datos y debug por DCode
  - FLIFT es un caché de prefetch de 2x64 bits
  - FLIFT permite escribir y borrar la Flash de a 16 bits (reprogramar)
- Acceso a la memoria de datos (RAM) y periféricos
  - Bus System tipo AHB
  - Un arbitrador da prioridad a los distintos maestros
    - Maestros: Cortex, DMA
    - Esclavos: Memorias, Periféricos
- **Periféricos en buses APB1, APB2**

Figure 1. System architecture (low-, medium-, XL-density devices)



Memoria

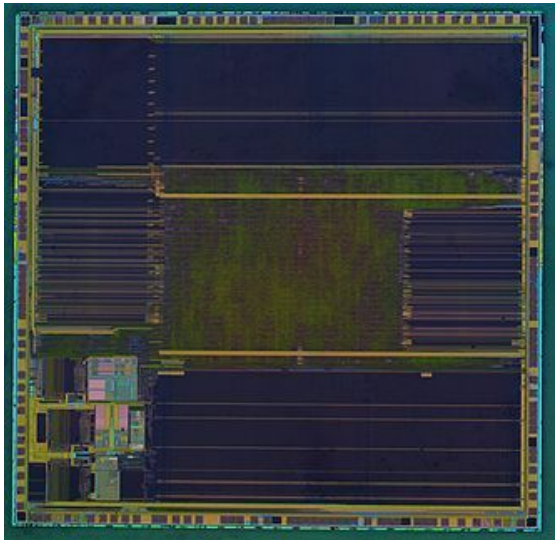
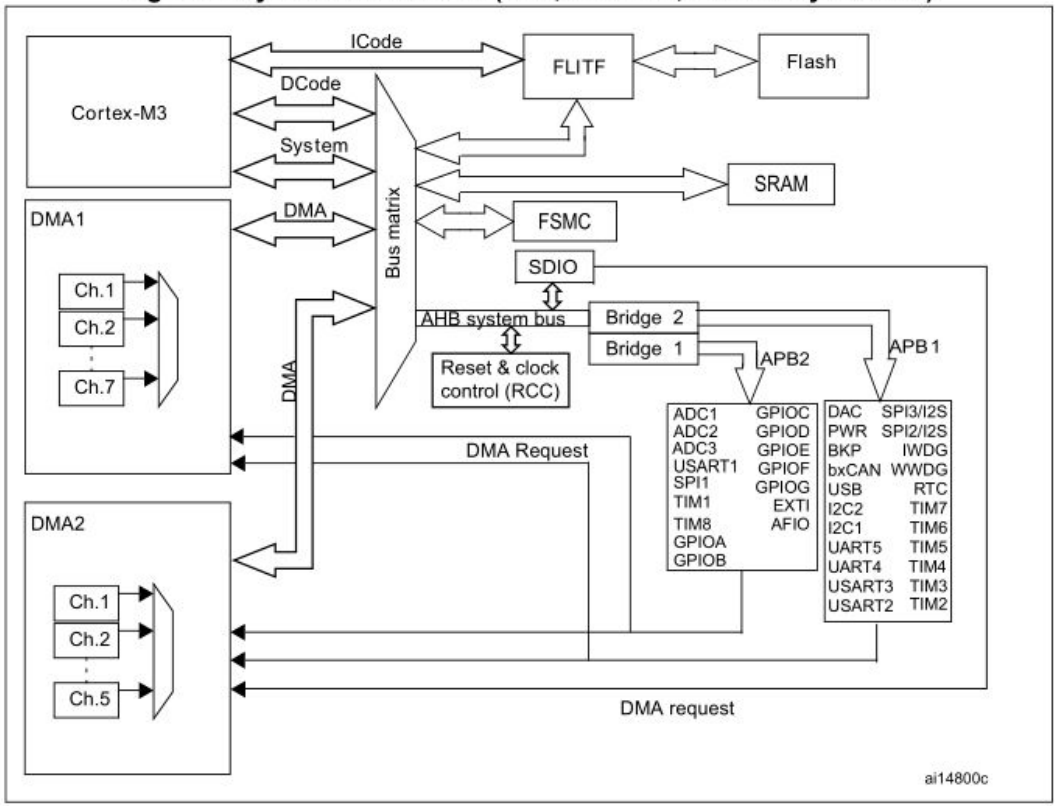


Figure 1. System architecture (low-, medium-, XL-density devices)



RM0008 - Reference manual„ ST

Particularidades del acceso a memoria : bus

- Ciclos ocultos por el bus
  - Si bien el procesador ve algunas operaciones como “de un ciclo”, el controlador del bus realiza estas operaciones en más de un ciclo:
    - Accesos no alineados
    - bit-band (lee-modifica-escribe)
    - etc
  - Esto puede causar retardos que no nos esperamos en algunas operaciones (importantemente: manejo de excepciones)
- Orden de Acceso
  - Normal: Se pueden reordenar los accesos
  - Device: Se preserva el orden respecto de accesos a memorias N y D
  - Strongly Ordered: Se preserva el orden respecto de todos los demás accesos

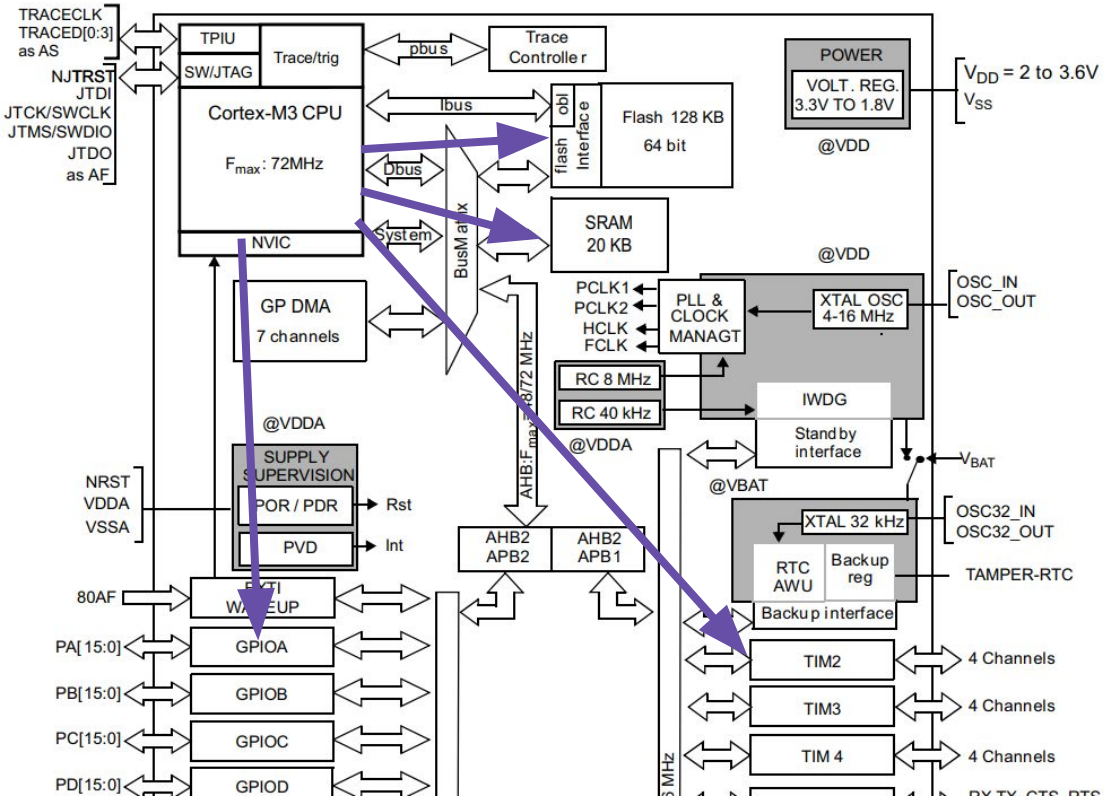
Table 12. Memory access behavior

Address range	Memory region	Memory type	XN	Description
0x00000000- 0x1FFFFFFF	Code	Normal(1)	-	Executable region for program code. You can also put data here.
0x20000000- 0x3FFFFFFF	SRAM	Normal(1)	-	Executable region for data. You can also put code here. This region includes bit band and bit band alias areas, see *Table 13 on page 28*.
0x40000000- 0x5FFFFFFF	Peripheral	Device(1)	XN(1)	This region includes bit band and bit band alias areas, see *Table 14 on page 28*.
0x60000000- 0x9FFFFFFF	External RAM	Normal(1)	-	Executable region for data.
0xA0000000- 0xDFFFFFFF	External device	Device(1)	XN(1)	External Device memory
0xE0000000- 0xE00FFFFF	Private Peripheral Bus	Strongly-ordered(1)	XN(1)	This region includes the NVIC, System timer, and system control block.
0xE0100000- 0xFFFFFFFF	Memory mapped peripherals	Device(1)	XN(1)	This region includes all the STM32 standard peripherals.

RM0008 - Reference manual, ST

Modelo de memoria

Vista del sistema desde el procesador



Modelo de memoria - Cortex M

- Los procesadores Cortex-M ven **todo** como una posición de memoria en un mapa de 4 GB (el máximo direccionable con 32 bits)
- No son direcciones “reales”, pero pueden verse como si lo fueran para usar el dispositivo
- Las direcciones de los segmentos son un “estándar” dentro de la familia Cortex-M para poder portar el código

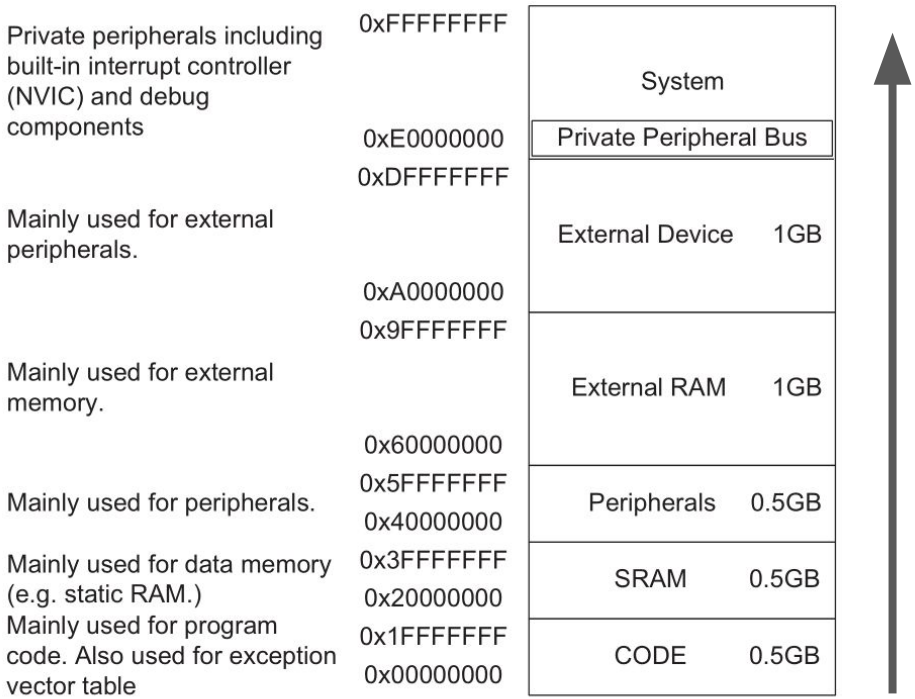


FIGURE 4.18

Memory map

Joseph Yiu - The Definitive Guide to ARM Cortex -M3 and Cortex-M4 Processors

Modelo de memoria - Cortex M

- Como el mapa de memoria es extenso, en toda la documentación será muy común encontrar las direcciones de memoria como

  `dirección = dirección_base + offset`

- La dirección base dependerá en contexto de lo que se esté leyendo, la mayoría de las veces será la dirección donde comienzan los registros de un periférico

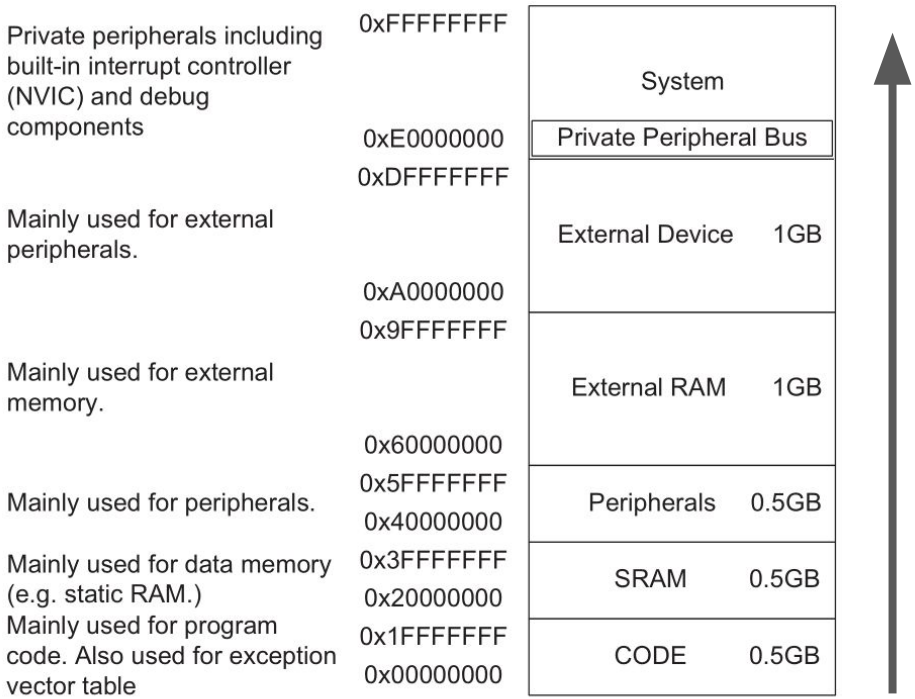
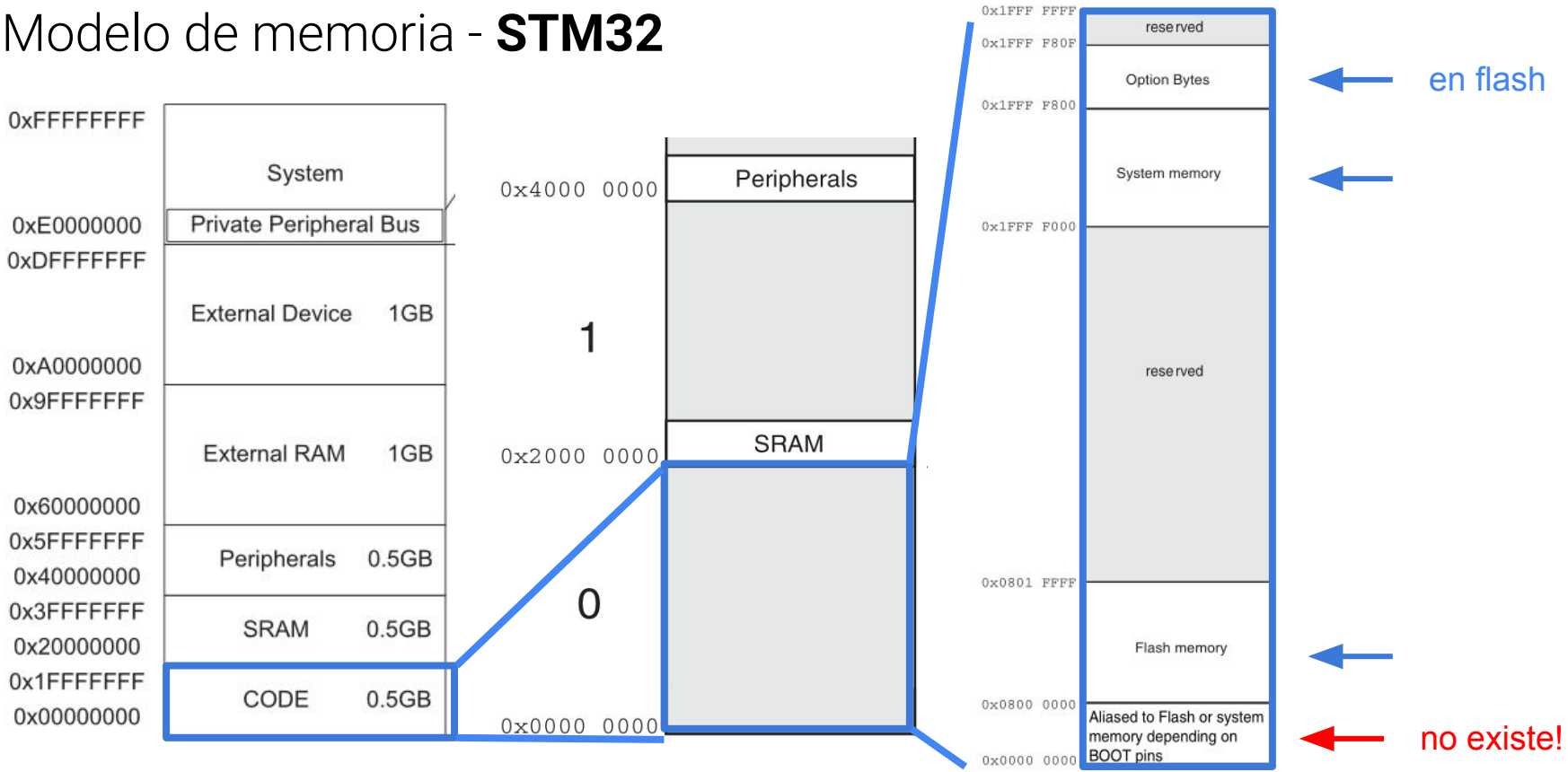


FIGURE 4.18

Memory map

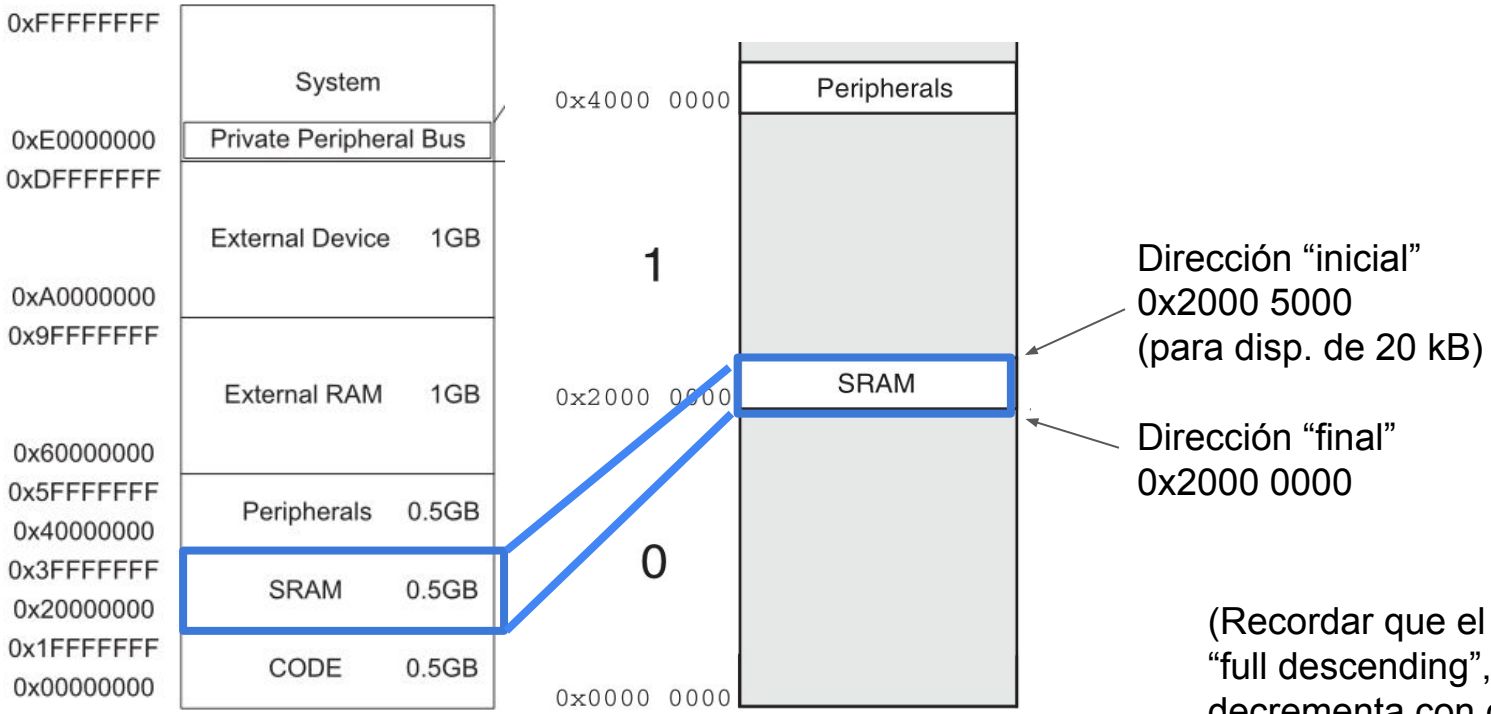
Joseph Yiu - The Definitive Guide to ARM Cortex -M3 and Cortex-M4 Processors

Modelo de memoria - **STM32**



Datasheet - STM32F103x8 STM32F103xB
Doc ID 13587 Rev 12, ST

Modelo de memoria - STM32 - SRAM



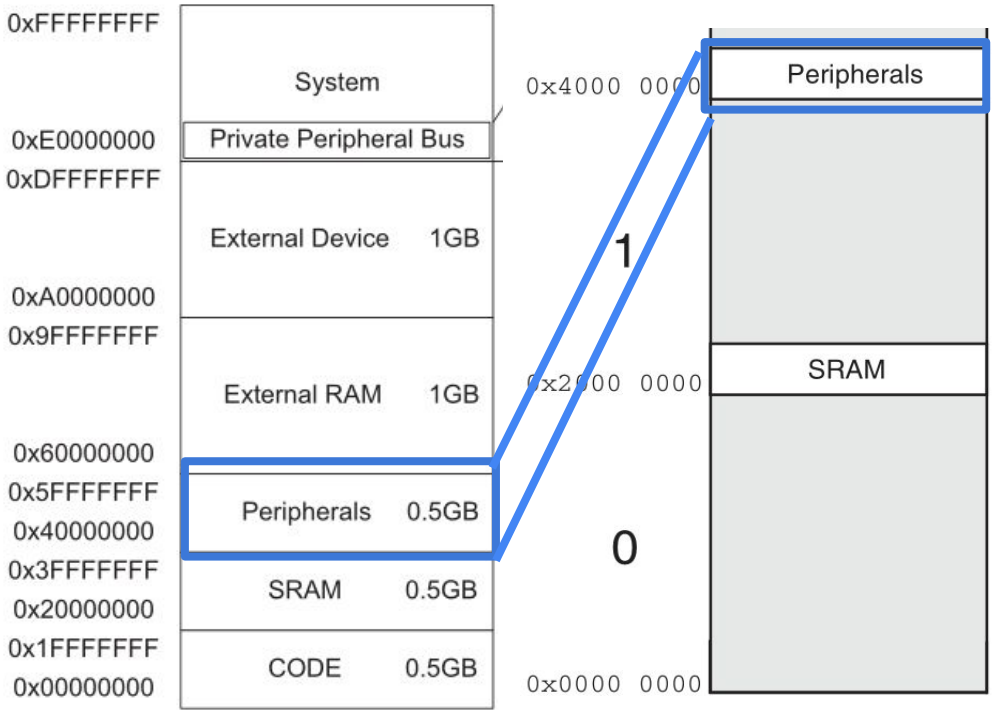
Dirección “inicial”
0x2000 5000
(para disp. de 20 kB)

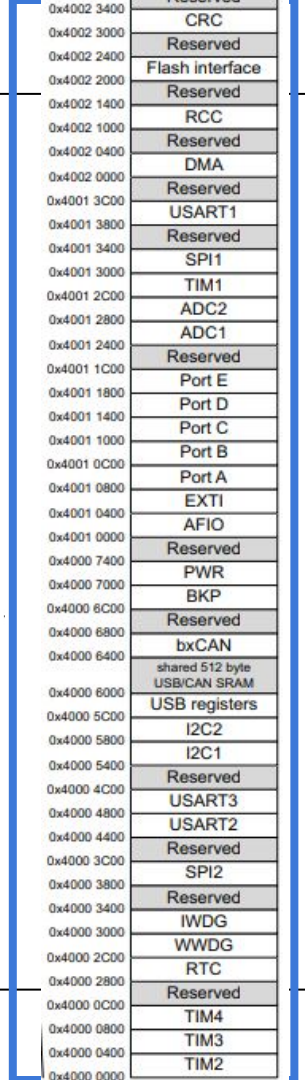
Dirección “final”
0x2000 0000

(Recordar que el stack pointer es “full descending”, es decir que se decrementa con cada push)

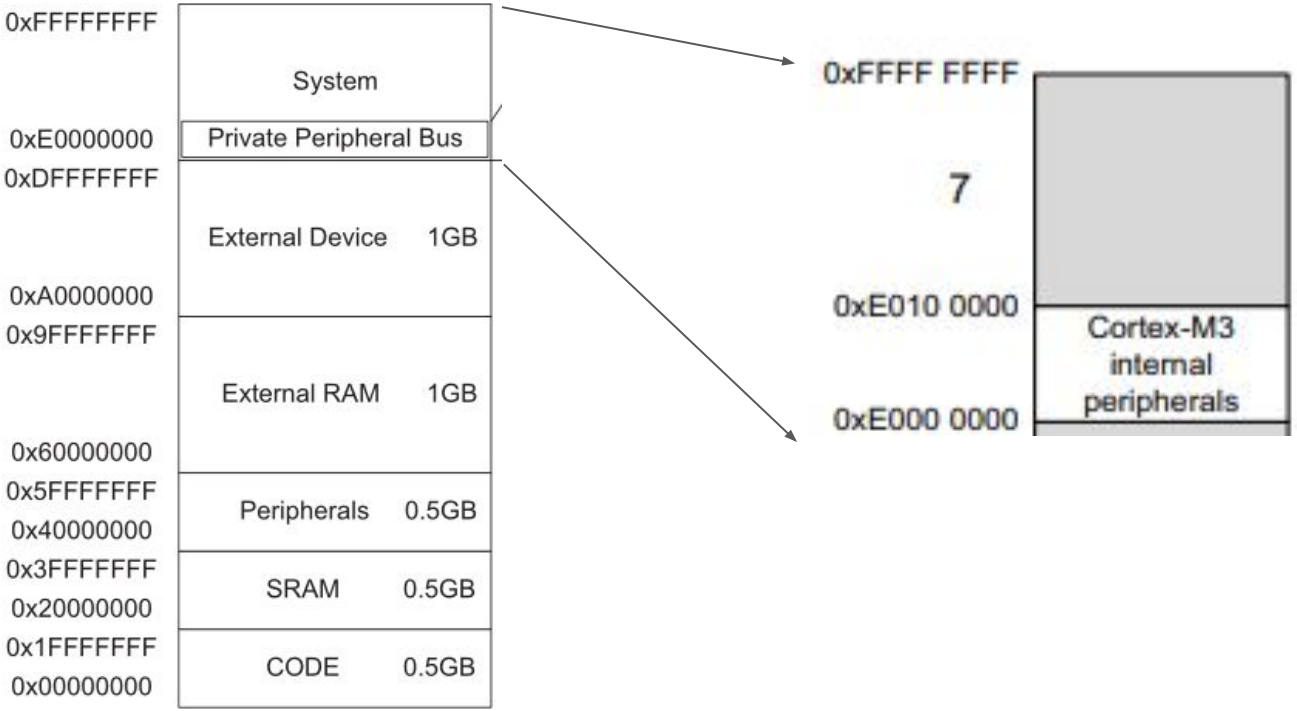
Datasheet - STM32F103x8 STM32F103xB
Doc ID 13587 Rev 12, ST

Modelo de memoria - STM32 - Perif.





Modelo de memoria - STM32 - Perif.



Manejo de Excepciones

Excepciones

- En sistemas embebidos es muy importante el concepto de “evento” y nuestros programas muy comúnmente responderán a eventos
  - Un ejemplo común son señales enviadas por periféricos, e.g. un ADC que indica que hay una muestra, un pulsador conectado al GPIO que fue presionado por el usuario
- Estos eventos pueden asociarse a “interrupciones”
  - Ya se conoce el concepto: las interrupciones permiten interrumpir rápidamente el flujo de ejecución del procesador y cambiar de contexto para atender una rutina
- También puede haber otro tipo de eventos, por ejemplo errores al decodificar una instrucción, que necesiten interrumpir el flujo de ejecución
- Por lo tanto, en los procesadores Cortex se tiene el concepto de manejo de **excepciones. Las interrupciones son un caso particular de excepciones.**

Excepciones - NVIC

- Las excepciones tienen prioridad configurable
- Se pueden anidar (la N es de Nested)
  - Implementa un procedimiento para atenderlas que, para el procesador, es muy parecido a un simple cambio de contexto.
  - Eso permite atender una excepción mientras se ejecuta otra, como cualquier subrutina
  - Implementa la prioridad en la que se ejecutarán
- Se ejecutan desde una tabla (la V de Vector)
  - Para ejecutar la rutina de atención de la excepción, debe saberse dónde está en memoria
  - Hay una tabla configurable en memoria con todas las posibles fuentes de excepción y la dirección de la subrutina que la maneja
  - El NVIC instantáneamente asocia la excepción a su lugar de la tabla y recupera dirección de la subrutina
- El NVIC recuerda, para cada excepción, si está pendiente o no aunque la fuente de excepción haya “bajado la mano” (pulsed vs level)

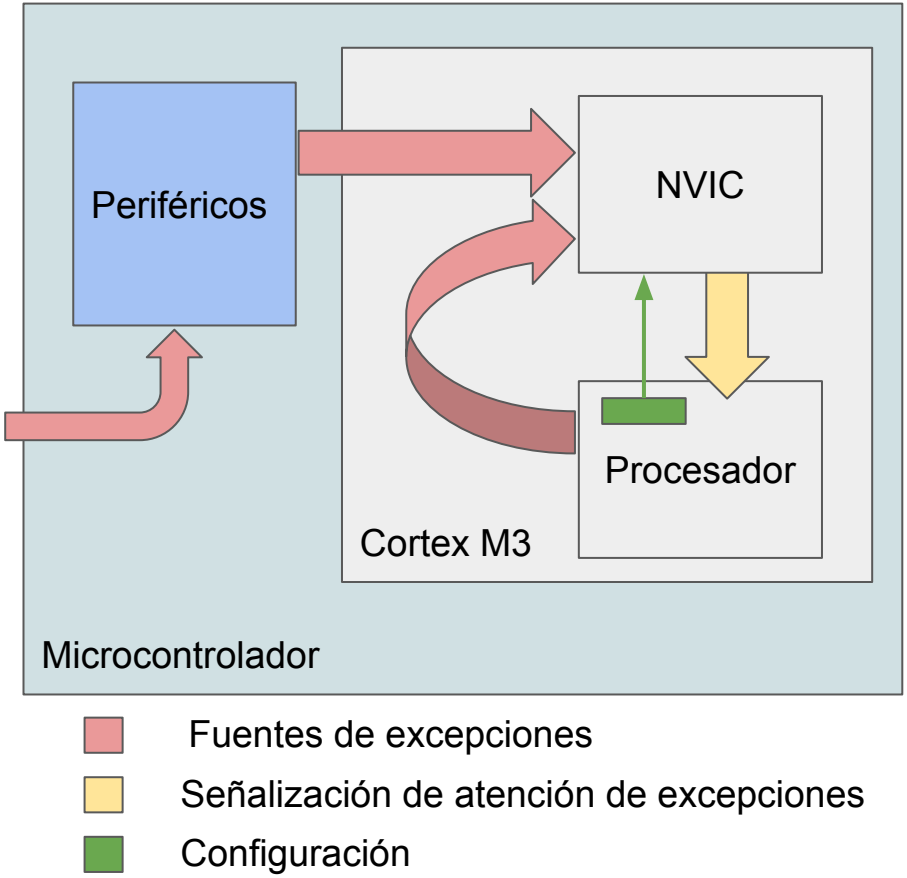


Tabla de excepciones - No enmascarables

Habilitadas por defecto
El resto deshabilitadas

Priority	Type of priority	Acronym	Description	Address
-	-	-	Reserved	0x0000_0000
-3	fixed	Reset	Reset	0x0000_0004
-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-1	fixed	HardFault	All class of fault	0x0000_000C
0	settable	MemManage	Memory management	0x0000_0010
1	settable	BusFault	Prefetch fault, memory access fault	0x0000_0014
2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	Reserved	0x0000_001C - 0x0000_002B
3	settable	SVCall	System service call via SWI instruction	0x0000_002C
4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	Reserved	0x0000_0034
5	settable	PendSV	Pendable request for system service	0x0000_0038
6	settable	SysTick	System tick timer	0x0000_003C

Se dispara cuando se resetea el uC para comenzar la ejecución del firmware

Interrupción no enmascarable del sistema de clock (avisa si falla el HSE)

Cualquier falla del sistema

Violaciones de acceso a la memoria o MPU

Errores del bus AHB

Errores de programa (e.g. instrucción ilegal)

Tabla de excepciones - Periféricos

7	settable	WWDG	Window watchdog interrupt	0x0000_0040
8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
9	settable	TAMPER	Tamper interrupt	0x0000_0048
10	settable	RTC	RTC global interrupt	0x0000_004C
11	settable	FLASH	Flash global interrupt	0x0000_0050
12	settable	RCC	RCC global interrupt	0x0000_0054
13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074

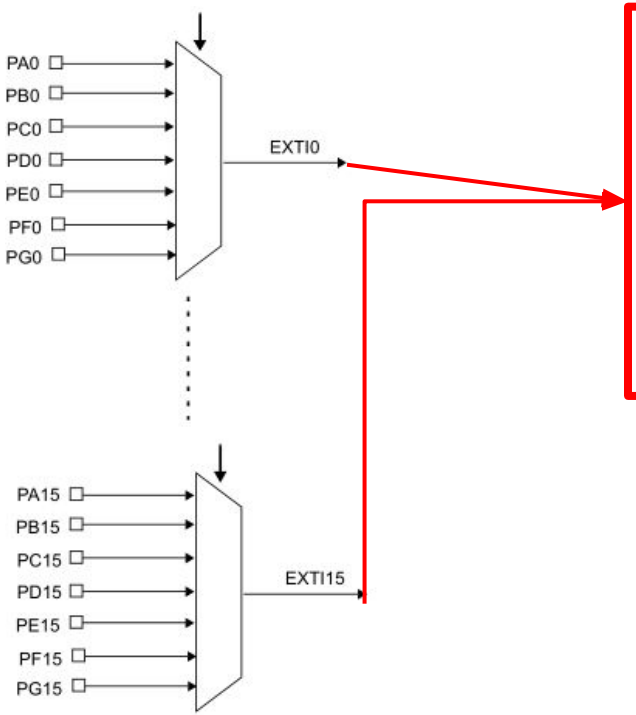
A partir de la 0x40 empiezan las interrupciones de **periféricos**

RM0008 - Reference manual, ST - Tabla 63

Controlador EXTI

- El uC tiene muchas entradas de GPIO y asignar cada posible fuente de interrupción de esos pines en el NVIC no es posible o no conviene para no implementar un NVIC “monstruoso”
- ST decidió concentrar esas interrupciones en un periférico adicional: el External Interrupt/Event Controller **EXTI**
- Concentra interrupciones al GPIO, permite implementar interrupciones por software, y también tiene algunas líneas de otros periféricos
  - Es interesante que detecta pulsos de menor duración que el clock del bus
- La configuración del EXTI varía entre dispositivos, por lo que veremos concretamente las características para el STM32F103c8

Configuración del EXTI



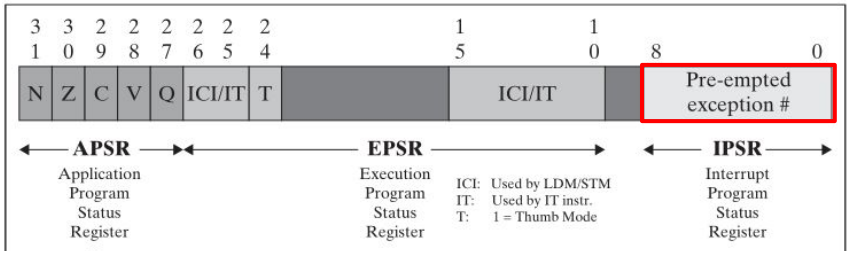
settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4
settable	USBWakeup	USB wakeup from suspend through EXTI line interrupt	0x0000_00E8
settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044

18 líneas EXTI en total

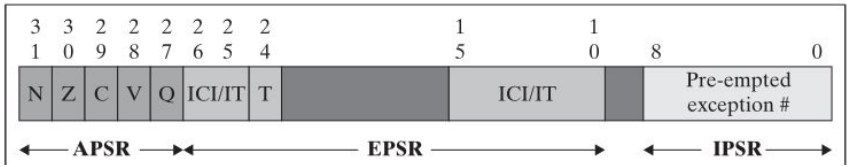
Manejo de excepciones - Registro especial IPSR

- Vimos que el Program Status Register tiene 3 partes, una de ellas es el IPSR
- Indica el tipo de excepción (es decir, el número) que se está ejecutando en el momento

Reg. especial PSR



Manejo de excepciones - Registro especial IPSR



Bits	Description
Bits 31:9	Reserved
Bits 8:0	**ISR_NUMBER**: This is the number of the current exception: 0: Thread mode 1: Reserved 2: NMI 3: Hard fault 4: Memory management fault 5: Bus fault 6: Usage fault 7: Reserved .... 10: Reserved 11: SVCall 12: Reserved for Debug 13: Reserved 14: PendSV 15: SysTick 16: IRQ0(1) .... 83: IRQ67(1) see *Exception types on page 32* for more information.

```
HAL_TIM_Base_Start_IT(&htim2);
```

lr	134218559
pc	0x800045e <TIM2_IRQHandl...
xpsr	16777260
primask	0
basepri	0
faultmask	0
control	0
msp	0x20004fd0
psp	0x0

```
Name : xpsr
    Hex:0x100002c
    Decimal:16777260
    Octal:0100000054
    Binary:10000000000000000000101100
    Default:16777260
```

Ejemplo: Al dispararse la interrupción del TIM2 aparece el valor 0x2C en el IPSR
Comparando las tablas vemos que corresponde al # del TIM2

-	-	-	-
-	-3	fixed	Reset
-	-2	fixed	NMI
-	-1	fixed	HardFault
-	0	settable	MemManage
-	1	settable	BusFault
-	2	settable	UsageFault
-	-	-	-
-	3	settable	SVCall
-	4	settable	Debug Monitor
-	-	-	-
-	5	settable	PendSV
-	6	settable	SysTick
0	7	settable	WWDG
1	8	settable	PVD
2	9	settable	TAMPER
3	10	settable	RTC
25	32	settable	TIM1_UP
26	33	settable	TIM1_TRG_COM
27	34	settable	TIM1_CC
28	35	settable	TIM2
29	36	settable	TIM3
30	37	settable	TIM4
31	38	settable	I2C1_EV

pm0056 - stm32f10xxx Cortex M3 programming manual, ST

D.W. Lewis - Fundamentals of Embedded Software with the Arm Cortex M3 (2012)

Manejo de excepciones - Nivel de prioridad

- El NVIC tiene los registros IPR donde, **para cada tipo de excepción**, se escribe el nivel de prioridad
- En los STM32, cada IPR tiene **sólo 4 bits útiles**
- Es decir que **hay 16 niveles de prioridad** de excepciones, asignables a las interrupciones de la tabla
- Como se utilizan los bits más significativos, los valores van del 0 al 255 en saltos de a 16
  - (Aunque en la librería de software lo vemos con valores del 1 al 16)
- El **número más bajo** es la que tiene **mayor prioridad**
- Este nivel determina si una interrupción puede **apropiarse de la ejecución (“preempt”)** frente a otra de menor nivel

B3.4.8 Interrupt Priority Registers, NVIC_IPR0 - NVC_IPR123

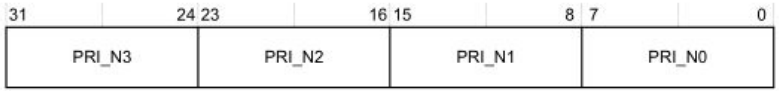


Figure B3-33 NVIC_IPR*n* Register bit assignments

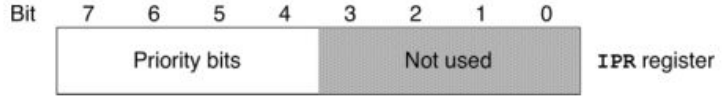


Figure 15: The content of IPR register on an STM32 MCU based on Cortex-M3/4/7 core

RM0008 - Reference manual, ST

Registros NVIC

Table 6-1 NVIC registers

Address	Name	Type	Reset	Description
0xE000E004	ICTR	RO	-	Interrupt Controller Type Register, ICTR
0xE000E100 - 0xE000E11C	NVIC_ISER0 - NVIC_ISER7	RW	0x00000000	Interrupt Set-Enable Registers
0xE000E180 - 0xE000E19C	NVIC_ICER0 - NVIC_ICER7	RW	0x00000000	Interrupt Clear-Enable Registers
0xE000E200 - 0xE000E21C	NVIC_ISPR0 - NVIC_ISPR7	RW	0x00000000	Interrupt Set-Pending Registers
0xE000E280 - 0xE000E29C	NVIC_ICPR0 - NVIC_ICPR7	RW	0x00000000	Interrupt Clear-Pending Registers
0xE000E300 - 0xE000E31C	NVIC_IABR0 - NVIC_IABR7	RO	0x00000000	Interrupt Active Bit Register
0xE000E400 - 0xE000E4EC	NVIC_IPR0 - NVIC_IPR59	RW	0x00000000	Interrupt Priority Register

https://developer.arm.com/documentation/100165/0201/Nested-Vectored-Interrupt-Controller/NVIC-programmers-model/Table-of-NVIC-registers

Manejo de excepciones - Sub-prioridad o prioridad de grupo

- Es opcional configurar un nivel de sub-prioridad con los bits PRIGROUP del registro AIRCR
- Pueden seleccionarse un subgrupo de los bits del IPR que determinarán una sub-prioridad
  - Entonces los bits más significativos seguirán determinando el nivel de prioridad, es decir cuál puede apropiarse de la ejecución
  - Pero los menos significativos indicarán el orden de prioridad dentro de ese nivel, que simplemente indica cuál se ejecuta a continuación, pero no permite la apropiación.

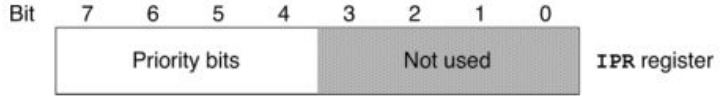
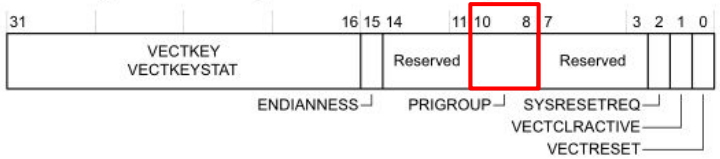


Figure 15: The content of IPR register on an STM32 MCU based on Cortex-M3/4/7 core

B3.2.6 Application Interrupt and Reset Control Register, AIRCR



[10:8]	RW	PRIGROUP	Priority grouping, indicates the binary point position. For information about the use of this field see *Priority grouping* on page B1-636. This field resets to 0b000.

Manejo de excepciones - Ejemplos Sub-prioridad

IPRx

1	1	0	1				

PRIGROUP

0	0

IPRz

1	1	0	0				

En este caso no hay ningún rango asignado a la subprioridad. La IRQ tipo z se aproriará de la ejecución frente a la de tipo x por tener mayor nivel (menor número)

IPRx

1	1	0	1				

PRIGROUP

1	0

IPRz

1	1	0	0				

En este caso los dos últimos bits corresponden a la subprioridad. Ambas IRQ tienen el mismo nivel. Por lo tanto, la de tipo z no se apropiará de la ejecución, pero si están ambas esperando, se ejecutará primero.

Manejo de excepciones - Ejemplos

- El nivel de prioridad determina cual interrupción o excepción puede apropiarse de la ejecución

- Mientras que la subprioridad o “prioridad de grupo” simplemente define cual es la próxima que se ejecutará, dado el mismo nivel

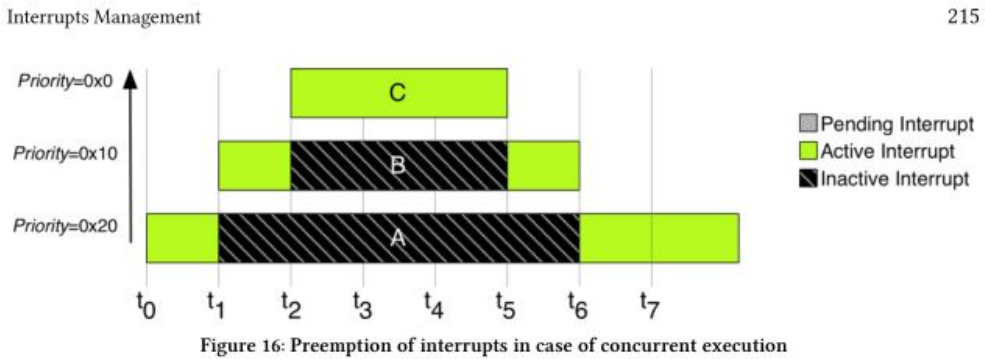
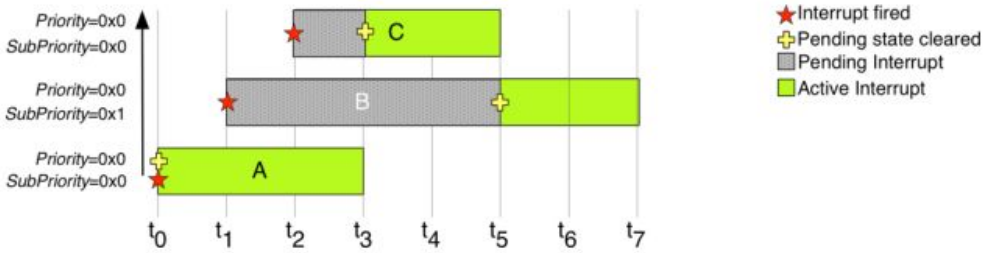


Figure 16: Preemption of interrupts in case of concurrent execution



Carmine Noviello - Mastering STM32 (2016)

Manejo de Excepciones - Registros Especiales Enmascaramiento

- Dentro de los registros especiales, encontramos los que sirven para enmascarar interrupciones
- PRIMASK (sólo LSB)
  - es un único bit que sirve para enmascarar todas las excepciones, excepto la de tipo HardFault
- FAULTMASK (sólo LSB)
  - Para sorpresa de nadie, enmascara las excepciones incluyendo la de HardFault
- BASEPRI (sólo [7:4])
  - Permite determinar un nivel a partir del cual se enmascaran las excepciones
  - Un valor de 0 significa que BASEPRI está deshabilitado
  - Un valor N != 0 significa que desde el nivel N (inclusive) hasta el último, las interrupciones quedarán enmascaradas, y sólo se permitirán las da mayor prioridad que N

Todas las excepciones de falla, a saber

- HardFault
- MemManage
- Bus Fault
- Usage Fault

derivan en una HardFault si no existe un vector asignado o si se está ejecutando una ISR de mayor prioridad (i.e. me ejecuto “si o si”)

Stacking

- Las excepciones se comportan en el Cortex como subrutinas comunes de C
  - Esto permite tener un anidamiento de muchas excepciones
- Pero cuando ocurre un cambio de contexto, siempre se debe guardar el estado de ejecución para poder volver luego y continuar lo que se dejó “a medias”
- Existe un estándar por el cual la función que se está ejecutando debe dejar algunos registros guardados en el stack para restituirlos al continuar, y la función que se llamó se responsabiliza de guardar otros
- Esto se cumple automáticamente al compilar en C, pero es importante porque insume tiempo, y como las excepciones son como cualquier función, también insumirá tiempo para ellas

Stacking

- Además de introducir latencia en la atención de la interrupción, es relevante considerar este procedimiento porque si se limitan las funciones a 4 parámetros (o información que quepa en R0-3), los mismos se pasan a través de los registros R0-3 en lugar de a través de la memoria

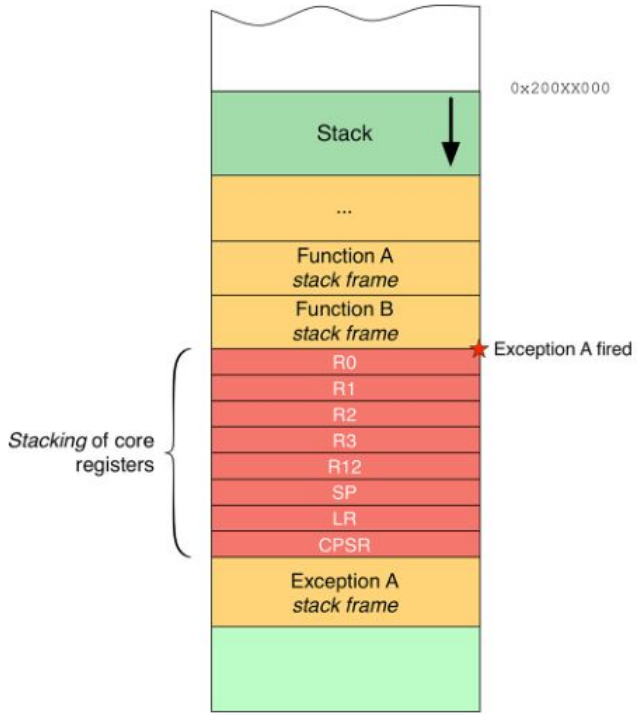
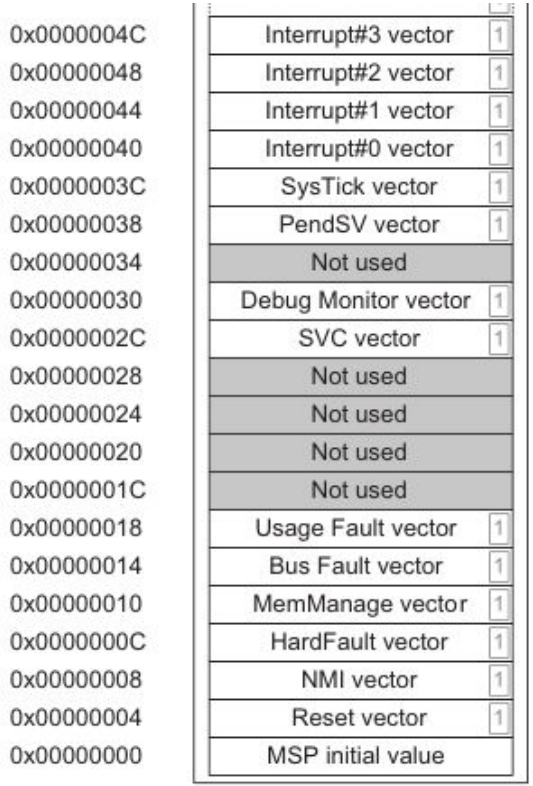


Figure 2: How core registers are stacked by the CPU on exception entrance

Tabla de vectores y Arranque

Localización de la tabla de vectores

- La tabla de vectores es una tabla real en memoria
- El NVIC permite relocalizar la tabla desde su posición predeterminada (0x00000000) a otras posiciones
- Esto se hace con el Vector Table Offset Register (VTOR).
- Luego de un reset, el VTOR vale 0, la posición predeterminada.
- Se puede mover la tabla a la SRAM, por ejemplo, y luego modificar el VTOR para que apunte a la nueva dirección



Joseph Yiu - The Definitive Guide to ARM Cortex -M3 and Cortex-M4 Processors

Booteo - Posición de la tabla de vectores

- Cuando se prende el dispositivo, automáticamente carga el stack pointer (R13) de la posición 0x00 de la tabla, y el PC (R15) de la posición 0x04 (recordar excepción de RESET).
- Por lo tanto debe saber dónde está la tabla! siempre la busca en la pos. 0
- ST implementó una estrategia para poder “cambiar” la posición de la tabla durante el encendido, según los pines de boot.
- En realidad, lo que hace es generar un alias y el procesador “ve” la dir 0x00000000 en el mismo lugar que la 0x80000000 (para el caso de la Flash)

Table 9. Boot modes

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as boot space
0	1	System memory	System memory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space

RM0008 - Reference manual, ST - Tabla 63

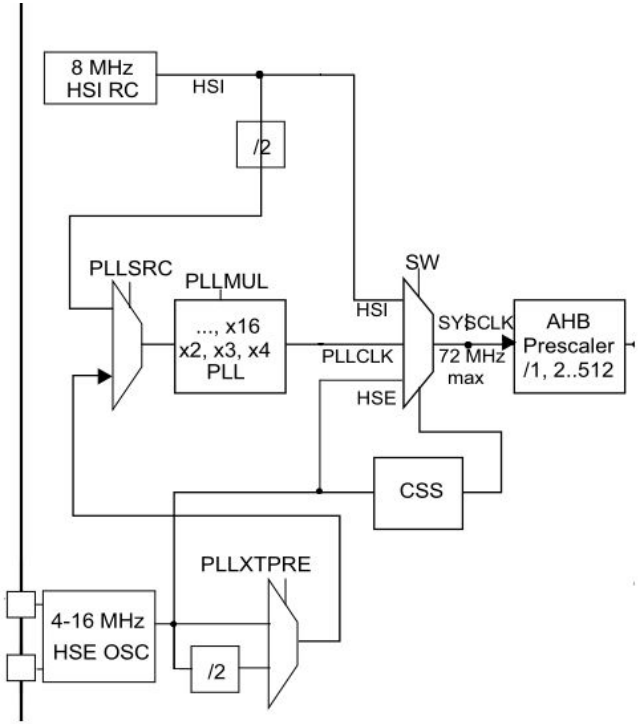
Registros del System Control Block

- El NVIC y el MPU son periféricos del propio procesador Cortex M3
- El procesador también tiene un System Control Block (SBC) que vimos en el mapa de memoria en el segmento 0xE0000000
- Allí se agrupan registros de configuración que comentamos:
  - AIRCR: Application interrupt and reset control register
  - VTOR: Vector table offset register
  - ICSR: Interrupt control and state register
  - entre otros…
- La documentación específica está en el Programming manual (PM0056) escrita por ST

Sistema de Clock

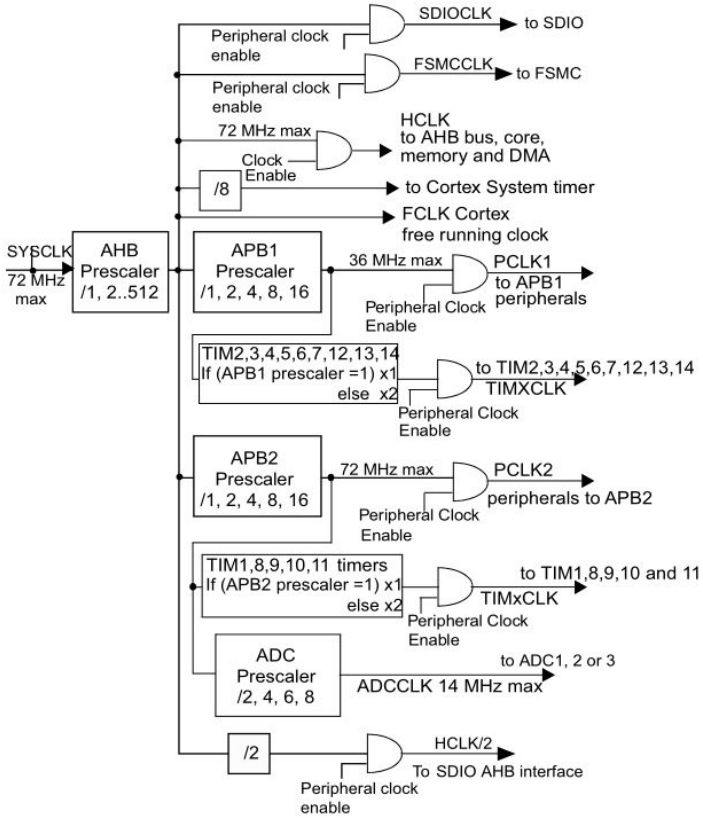
Sistema de Clock - Principal

- El sistema de clock puede configurar:
  - Fuente de clock principal
    - De origen interno (HSI): Tiene un oscilador interno de 8 MHz de precisión “media”
    - De origen externo (HSE): Puede conectarse un oscilador o un resonador de cristal o cerámico para obtener un clock de mayor presición
  - Fuente de clock “de tiempo real” o “lento” (No confundir “tiempo real”; en este contexto significa poder contar el tiempo en segundos exactos para medir minutos, horas, etc)
    - De origen interno: Un oscilador RC de baja precisión, aunque se puede calibrar
    - De origen externo: Puede conectarse un resonador de 32.768 kHz o al HSE/128
- A partir de la fuente de clock, se hace llegar a los periféricos
  - Primero, existe un PLL que permite multiplicar la frecuencia
  - Luego, existe un interconexionado con divisores o prescalers



Sistema de Clock - Periféricos

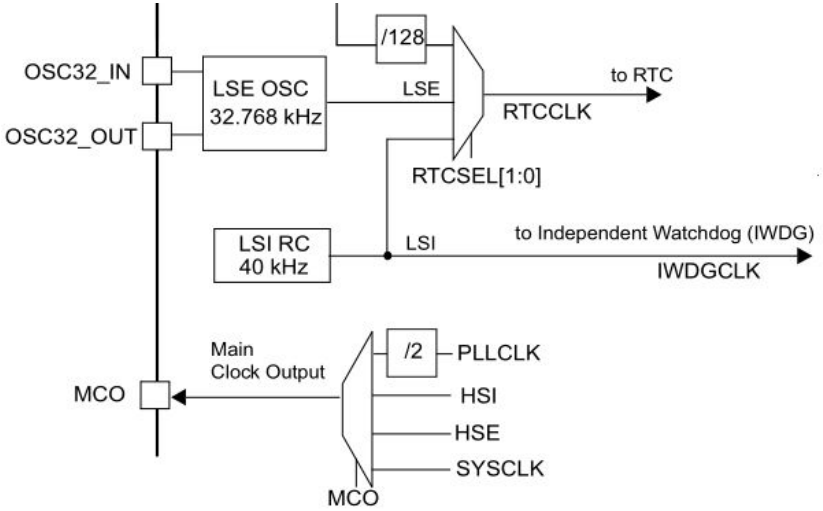
- Del SYSCLK se obtienen los clocks para el resto de los periféricos (excepto algunos que no se muestran en esta imágen)
- Es importante notar que todos los periféricos tienen un “clock enable”
  - Debemos habilitar el clock respectivo para usarlo - Incluso el GPIO!
  - Puede ahorrarse energía deshabilitando lo que no se usa

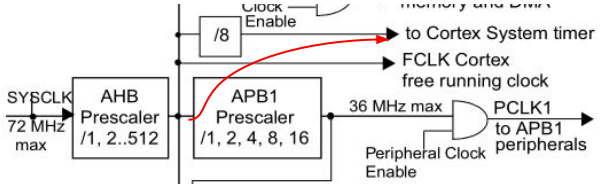


RM0008 - Reference manual, ST

Sistema de Clock - Características

- Clock Security System (CSS)
  - Permite recuperar el microcontrolador ante la pérdida del HSE
  - Se cambia la fuente de clock al HSI y se ejecuta la NMI indefinidamente hasta que se limpia el pending bit del CSS interrupt
- Watchdog independiente
  - La fuente interna LSI que mencionamos para el RTC sirve para alimentar un watchdog que de esta manera será independiente del resto del sistema
- Externalización del clock
  - Puede sacarse la señal de clock de diversas fuentes por un pin
- System timer
  - Desde el clock y a través de un divisor se extrae una señal de clock SYSTICK, importante para el funcionamiento de sistemas operativos, y con su propia excepción asignada

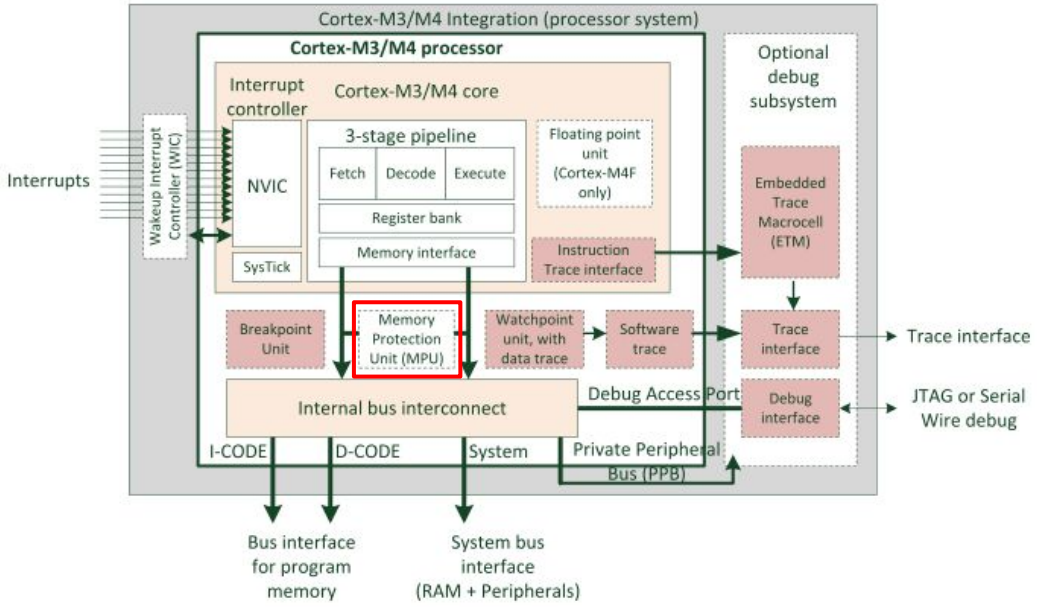




RM0008 - Reference manual, ST

Memory Protection Unit

- Es una característica opcional para los Cortex-M3
- El STM32F103 no lo tiene
- Permite determinar áreas de la memoria donde habrá un control de acceso (notar que se inserta entre el núcleo del procesador y los buses)
- Esto es muy útil para sistemas complejos ya que permite evitar:
  - Que una subrutina sin privilegios corrompa áreas importantes de la memoria o del Stack
  - Que se acceda a ciertos periféricos sin tener permiso



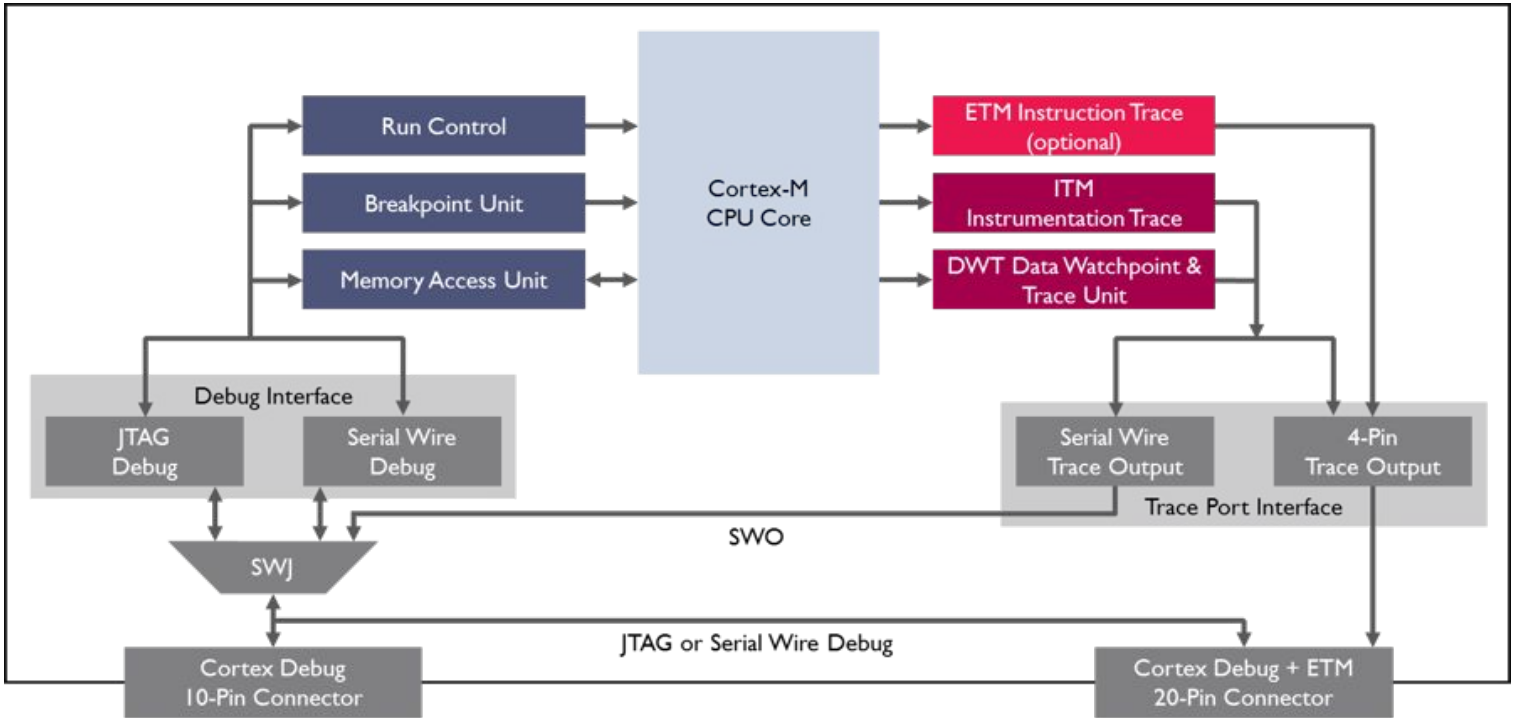
Joseph Yiu - The Definitive Guide to ARM Cortex -M3 and Cortex-M4 Processors

Depuración

Debug

- Los microcontroladores más complejos requieren un firmware más complejo pero también incorporan extensiones de debug más poderosos
- Estas extensiones forman parte del procesador (el microcontrolador a su vez puede incluir otras extensiones que asisten al debug de periféricos)
- Por un lado, las extensiones de **debug** permiten detener al procesador y analizar su estado y el estado de la memoria.
- Por otro lado extensiones de **trace** permiten recuperar información sobre la ejecución de manera “no invasiva”, es decir, durante la ejecución

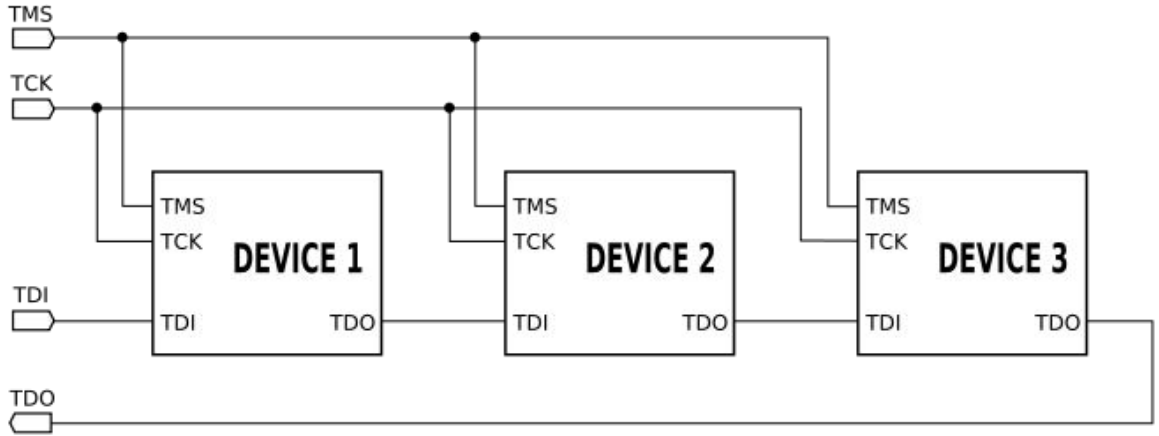
Debug - Características del Cortex-M



https://www2.keil.com/coresight

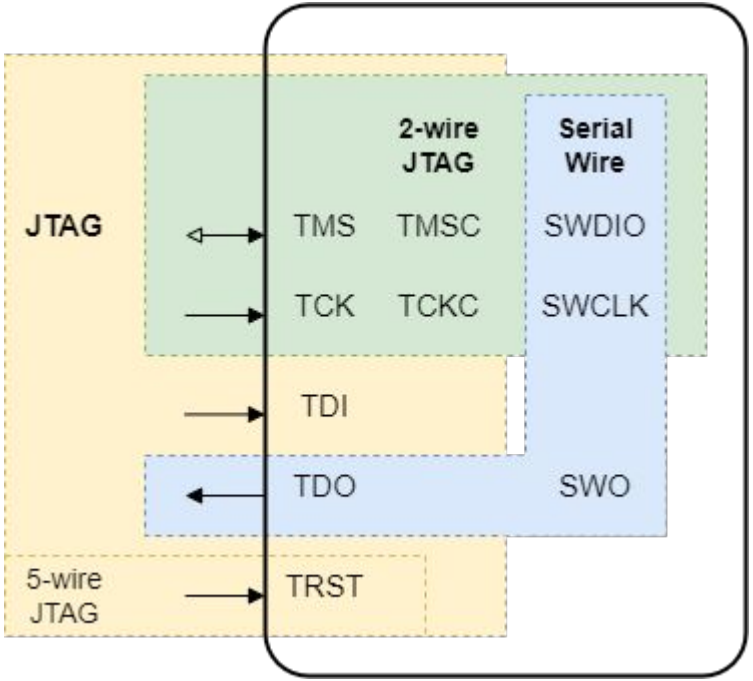
Debug - Interfaz

- La interfaz de debug es
  - una interfaz JTAG estándar que soporta opciones de 5, 4 o 2 pines
  - compartida con una interfaz Serial Wire (de ARM) de 2 pines
- JTAG es un estándar para sistemas embebidos. Casi cualquier SE “moderno” tendrá un puerto JTAG, aunque puede variar el conector
- Una característica interesante es que contempla la conexión de dispositivos en daisy-chain
- Dentro de un mismo dispositivo existen TAPs, sub-unidades independientes que pueden emitir información a través de JTAG, una de ellas siendo el Boundary scan



Debug - Interfaz física STM32F103c8t

- En JTAG
  - TMS es una línea para indicar “modo debug”
  - La comunicación es similar a SPI a través de
    - TCK
    - TDI
    - TDO
  - Opcionalmente en el protocolo de 5 líneas una línea TRST permite resetear la lógica del controlador JTAG en el dispositivo
- En Serial Wire
  - Las características de debug se logran a través de las líneas:
    - SWDIO es una línea bidireccional de comunicación
    - SWCLK es el clock
  - Y si se desea usar el Trace, puede hacerse a través de la línea SWO



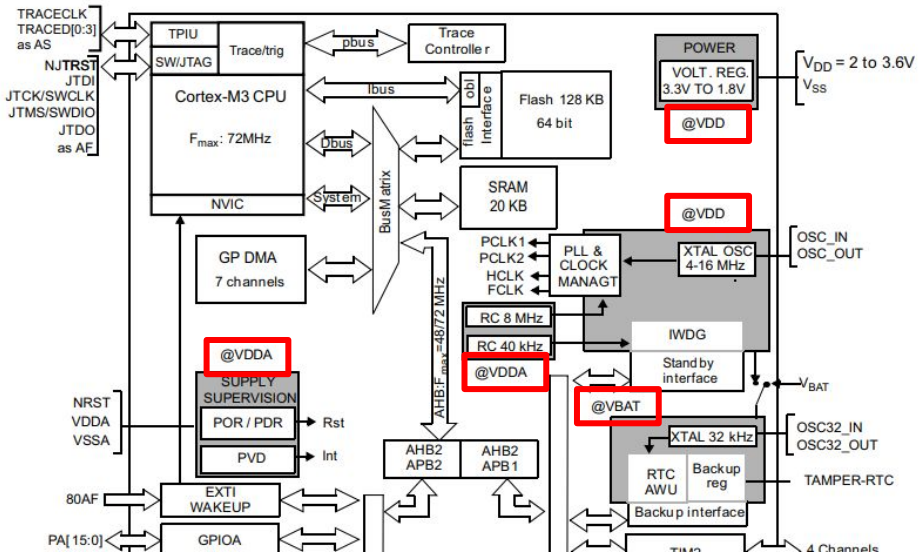
Debug - Interfaz física

- Configuración de pines en STM32F103C8

Table 219. SWJ debug port pins

SWJ-DP pin name	JTAG debug port		SW debug port		Pin assignment
	Type	Description	Type	Debug assignment	
JTMS/SWDIO	I	JTAG Test Mode Selection	IO	Serial Wire Data Input/Output	PA13
JTCK/SWCLK	I	JTAG Test Clock	I	Serial Wire Clock	PA14
JTDI	I	JTAG Test Data Input	-	-	PA15
JTDO/TRACESWO	O	JTAG Test Data Output	-	TRACESWO if async trace is enabled	PB3
NJTRST	I	JTAG Test nReset	-	-	PB4

Alimentación



5.1.6 Power supply scheme

Figure 14. Power supply scheme