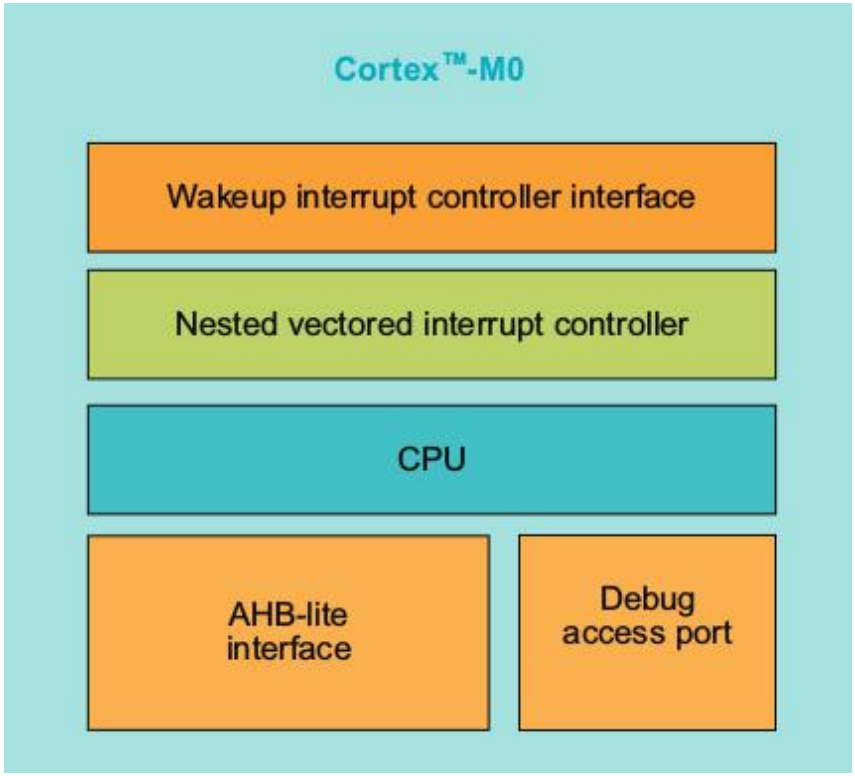
Arquitectura Cortex – M

Juan Alarcón.
jalarcon@electron.frba.utn.edu.ar

Agenda.

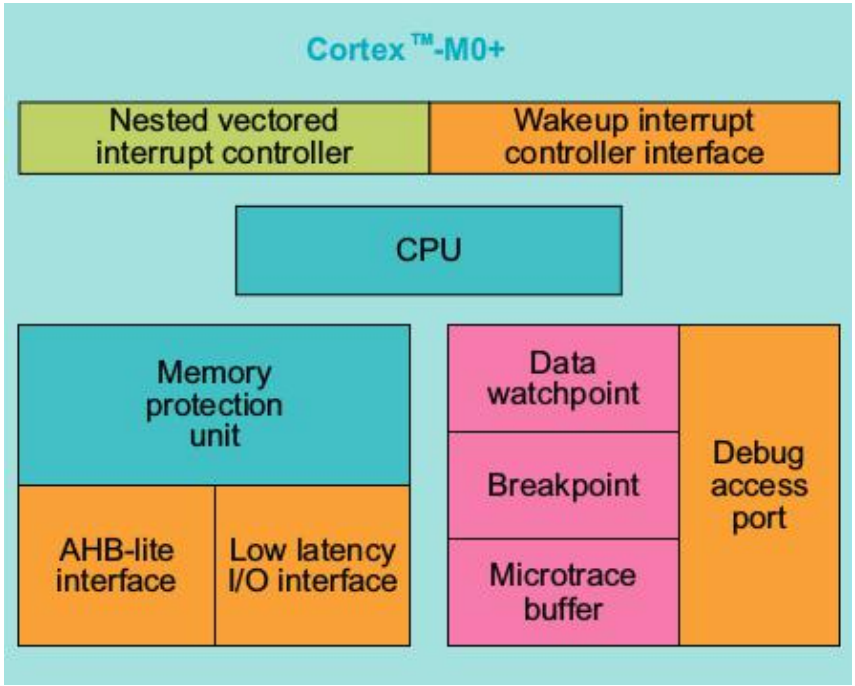
- Cortex M0/M0+
- Cortex M4. DSP
- Bajo Consumo.
- Debug. JTAG. Serial Wire. Cortex M.
- Cortex Microcontroller Software Interface Standard (CMSIS).
- Cortex M. Modos de operación.
- Service Call (SVC)
- Pend_SVC

Cortex-M0.



- Es el procesador más pequeño de la familia.
- Arquitectura Von Neumann. Menor rendimiento que los M3.
- Mismo controlador de Interrupciones que el M3 (NVIC) con 32 fuentes de interrupción máximo.
- Menores capacidades de debug que M3.

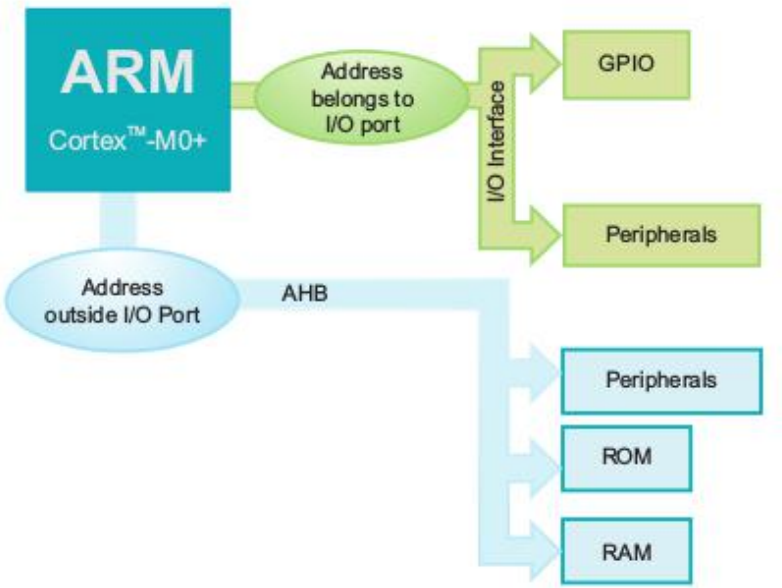
Cortex-M0+



- Es totalmente compatible con el M0.
- Cambios en el pipeline y en la interfaz de I/O.
- Diseñado para utilizar instrucciones de 16bits (casi todas en Cortex M0).
- Mayores capacidades de Debug.

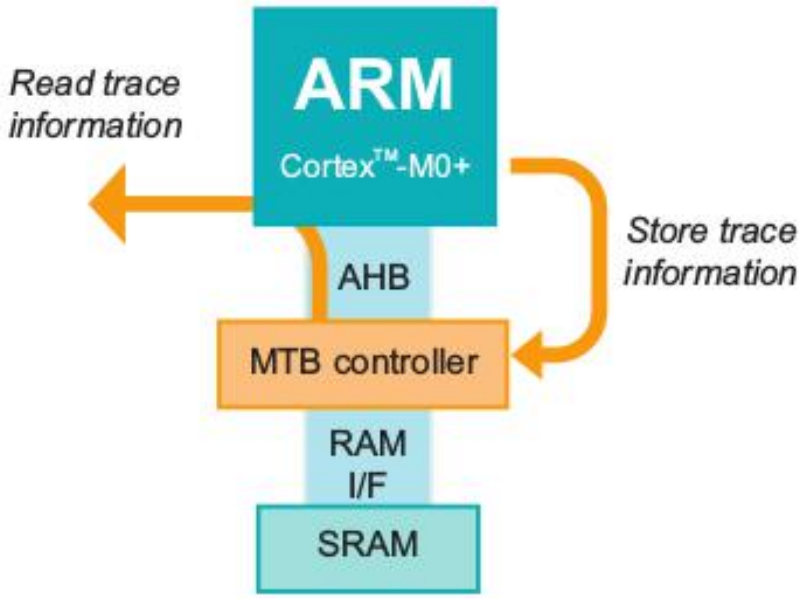
Cortex-M0+

Cortex-M0+



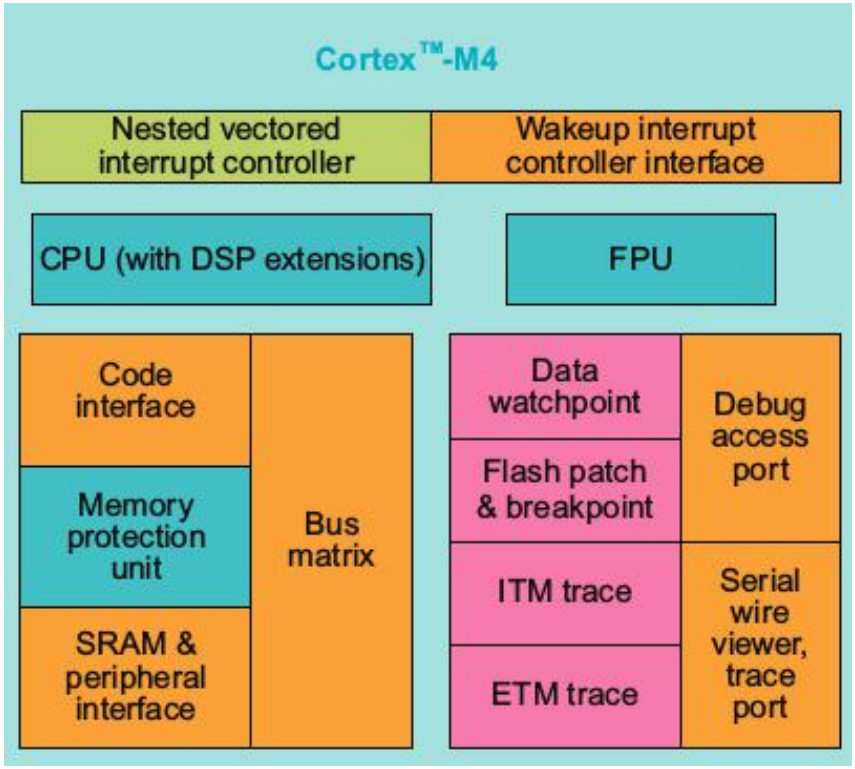
- Los Cortex-M0+ Agregan una interfaz de periféricos aparte del AHB.
- Permite acceder a perifericos y GPIO en un único ciclo de reloj.
- Es transparente a la interfaz de programa.

Cortex-M0+



- Los Cortex-M0+ tienen una interfaz de debug completa, como los M3 y M4.
- Agregan el controlador MTB (microtrace buffer). Que permite guardar una "cola" de instrucciones ejecutadas en SRAM.

Cortex-M4



- El Cortex-M0 se puede pensar como un M3 "recortado" mientras que el M4 se puede considerar como un M3 con DSP.
- Los Cortex-M4 están orientados a algoritmos de DSP como filtros, FFT, lazos PID.

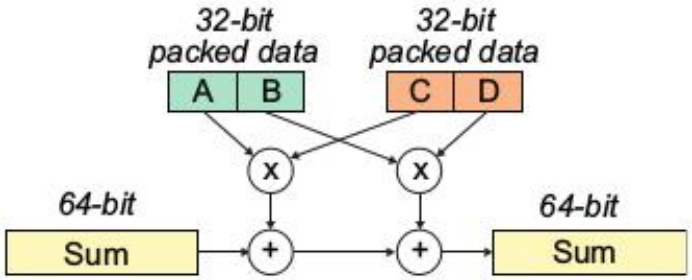
Cortex-M4.

Single Cycle MAC Instructions on the Cortex-M4

Operation	Instructions
16 x 16 = 32	SMULBB, SMULBT, SMULTB, SMULTT
16 x 16 + 32 = 32	SMLABB, SMLABT, SMLATB, SMLATT
16 x 16 + 64 = 64	SMLALBB, SMLALBT, SMLALTB, SMLALTT
16 x 32 = 32	SMULWB, SMULWT
(16 x 32) + 32 = 32	SMLAWB, SMLAWT
(16 x 16) ± (16 x 16) = 32	SMUAD, SMUADX, SMUSD, SMUSDX
(16 x 16) ± (16 x 16) + 32 = 32	SMLAD, SMLADX, SMLSD, SMLSDX
(16 x 16) ± (16 x 16) + 64 = 64	SMLALD, SMLALDX, SMLSLD, SMLSLDX
32 x 32 = 32	MUL
32 ± (32 x 32) = 32	MLA, MLS
32 x 32 = 64	SMULL, UMULL
(32 x 32) + 64 = 64	SMLAL, UMLAL
(32 x 32) + 32 + 32 = 64	UMAAL
32 ± (32 x 32) = 32 (upper)	SMMLA, SMMLAR, SMMLS, SMMLSR
(32 x 32) = 32 (upper)	SMMUL, SMMULR

- Los Cortex-M4 tienen el mismo NVIC que los M3, el mismo sistema de Debug y Buses.
- Agregan instrucciones de Multiplicar y acumular por hardware.
- Los Cortex-M4 agregan unidad de punto flotante (FPU).
- Entre las instrucciones de DSP agregan las SIMD.

Cortex-M4. SIMD.



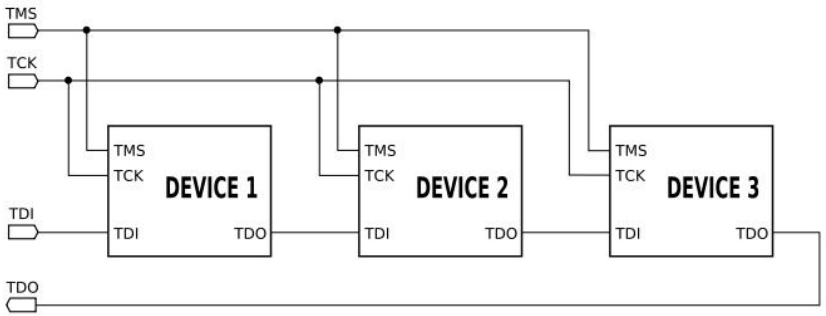
- Las instrucciones SIMD (single instruction mutiple data). Permiten hacer varias operaciones de 8 o 16 bits en un único ciclo de reloj.
- Las SIMD permiten optimizar algoritmos como los de filtros FIR.

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k]$$

Administración de Energía.

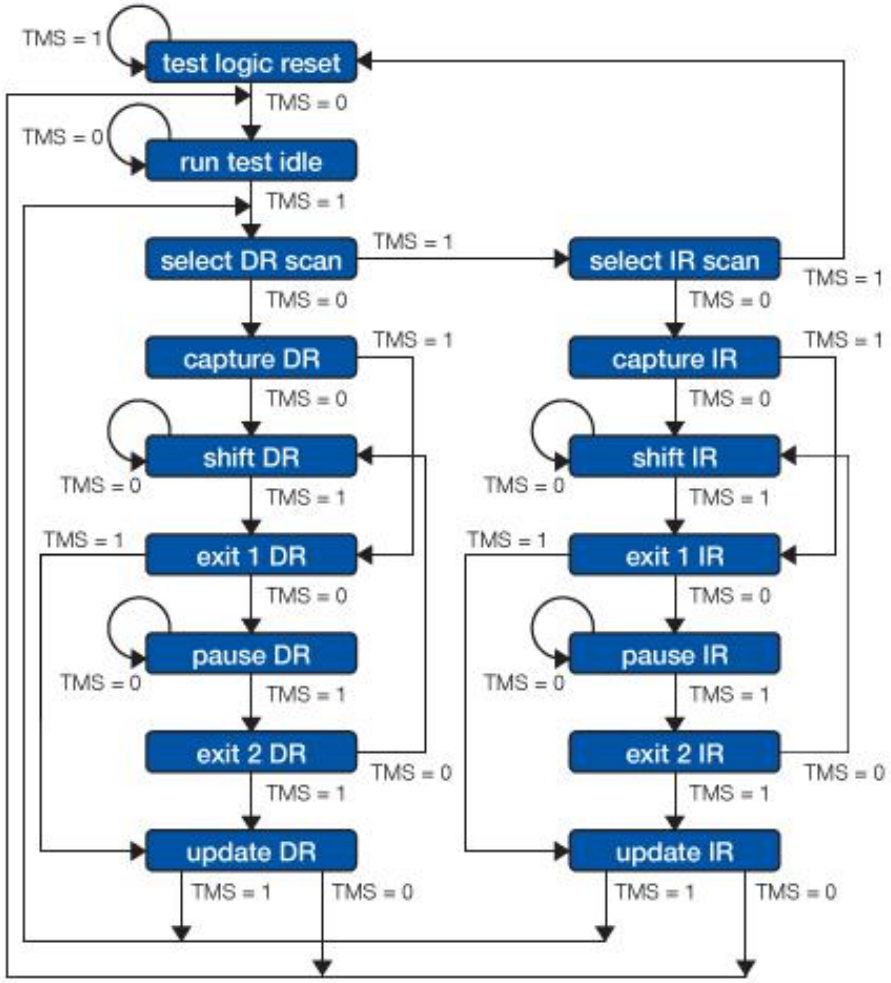
- Los procesadores Cortex-M pueden entrar en dos modos de bajo consumo. ***SLEEP*** y ***DEEPSLEEP***.
  - ***SLEEP.*** Detiene el reloj de la CPU pero no de los periféricos. Cualquiera de los periféricos puede despertar al CPU por medio de interrupciones.
  - ***DEEPSLEEP.*** Detiene el reloj del CPU y de la mayoría de los periféricos. Tipicamente se sostiene el reloj de la SRAM. Al no tener reloj el NVIC es necesario otro dispositivo para volver de este modo.
  - El Wakeup Interrupt Controller (WIC) es un dispositivo sencillo y de bajo consumo que permite volver de un DEEPSLEEP.

Debug. JTAG



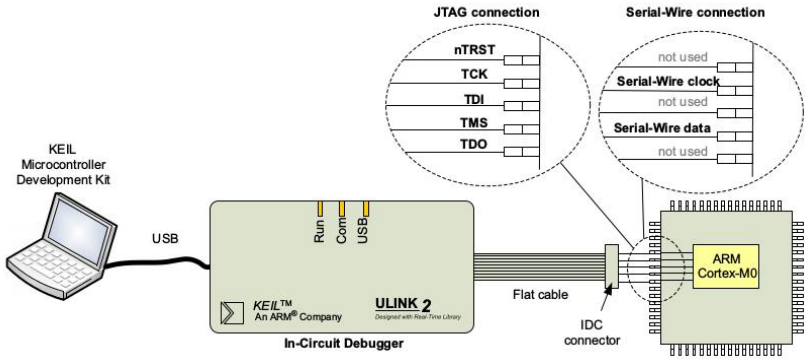
- JTAG es un protocolo de debug de 4 (lo más usual) o 5 pines.
- JTAG es un protocolo serie que permite "encadenar" varios dispositivos sobre el mismo puerto.
- JTAG es definido por la norma IEEE 1149.1-1990.

Debug. JTAG.



- JTAG define un registro de datos(DR) y un registro de instrucción (IR).
- JTAG define algunas instrucciones obligatorias
  - BYPASS
  - EXTEST
  - SAMPLE/PRELOAD
  - IDCODE (no obligatoria)
- Permite definir instrucciones propias (usadas por casi todos los fabricantes para programar dispositivos)

Debug. SerialWire



- El protocolo SerialWire usa dos pines SWCLK y SWDIO.
- La línea de datos es bidireccional y la de CLK es siempre manejada por el debugger.
- En los microcontroladores LPC se lo puede controlar totalmente por este protocolo.

Debug en Cortex-M3

- Los procesadores Cortex-M3 provee dos características de debug que pueden ser clasificadas en dos tipos:
  - Invasivo.
  - No invasivo.

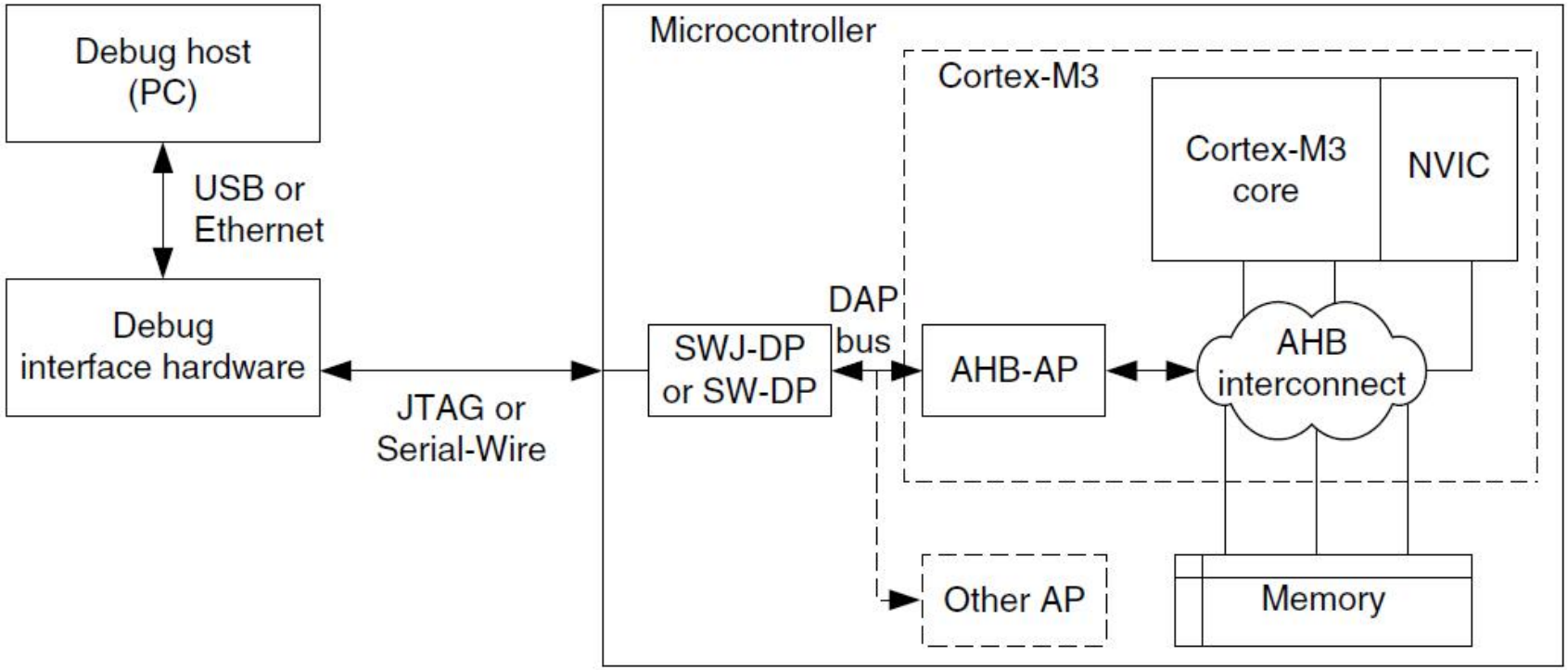
Debug en Cortex-M3

- Debug invasivo:
  - Ejecución por pasos.
  - Breakpoints por hardware.
  - Excepciones de debug.
  - Acceso a registros (lectura y escritura)
  - Acceso y modificación de datos en memoria.

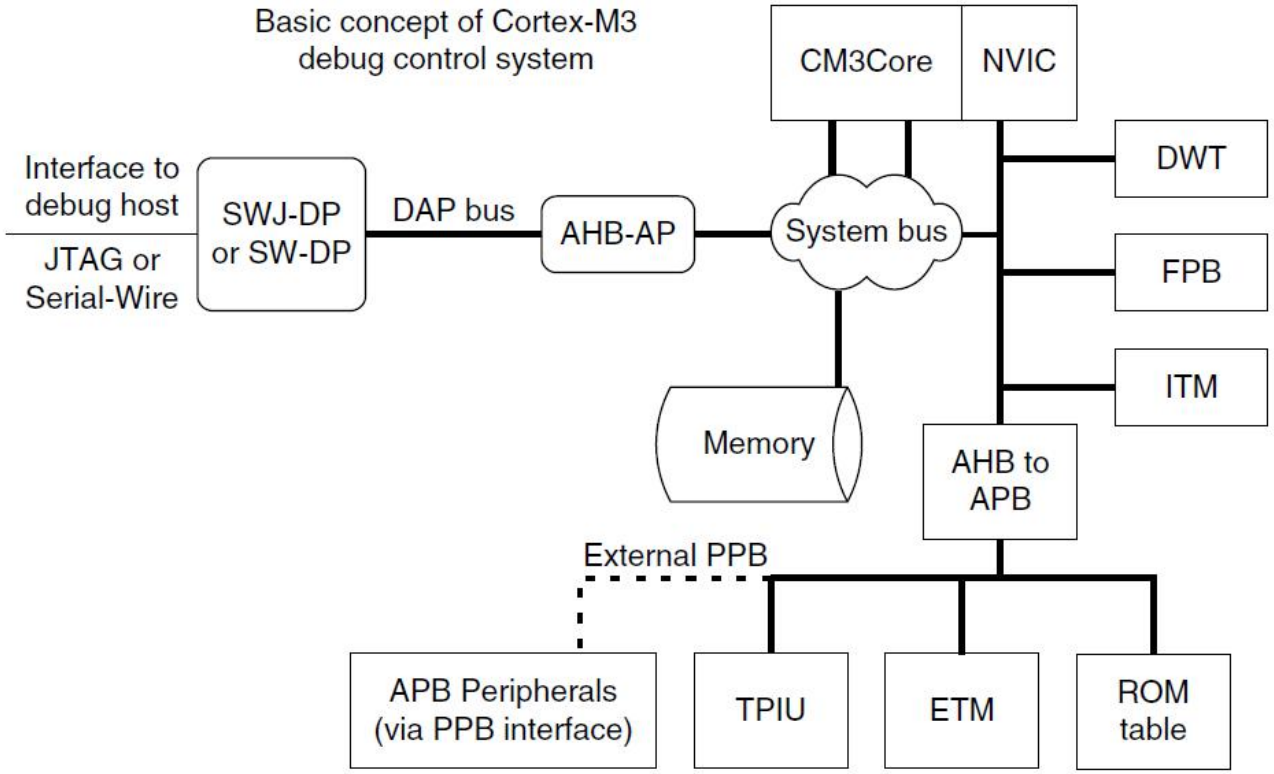
Debug en Cortex-M3

- Debug no invasivo:
  - Acceso a memoria (lectura de memoria).
  - Seguimiento:
    - Instrucciones
    - Datos.
  - Profiling.

Interfaz de Debug.



Interfaz de Debug (2)



Modos de debug.

- Los procesadores Cortex-M3 soportan dos modos de debug.
  - Halt. El procesador detiene su ejecución completamente.
  - Debug monitor. El procesador ejecuta una excepción de debug.

Modos de debug. Halt.

- La ejecución del procesador ***se detiene***.
- El SYSTICK se detiene.
- Las interrupciones son demoradas y/o enmascaradas.

Modos de Debug. Halt.

Table 15.1 Debug Halting Control and Status Register (0xE000EDF0)

Bits	Name	Type	Reset Value	Description
31:16	KEY	W	—	Debug key; value of 0xA05F must be written to this field to write to this register, otherwise the write will be ignored
25	S_RESET_ST	R	—	Core has been reset or being reset; this bit is clear on read
24	S_RETIRE_ST	R	—	Instruction is completed since last read; this bit is clear on read
19	S_LOCKUP	R	—	When this bit is 1, the core is in a locked-up state
18	S_SLEEP	R	—	When this bit is 1, the core is in sleep mode
17	S_HALT	R	—	When this bit is 1, the core is halted
16	S_REGRDY	R	—	Register read/write operation is completed
15:6	Reserved	—	—	Reserved
5	C_SNAPSTALL	R/W	0*	Use to break a stalled memory access
4	Reserved	—	—	Reserved
3	C_MASKINTS	R/W	0*	Mask interrupts while stepping; can only be modified when the processor is halted
2	C_STEP	R/W	0*	Single step the processor; valid only if C_DEBUGEN is set
1	C_HALT	R/W	0*	Halt the processor core; valid only if C_DEBUGEN is set
0	C_DEBUGEN	R/W	0*	Enable halt mode debug

** The control bit in DHCSR is reset by power on reset. System reset (for example, by the Application Interrupt and Reset Control register of NVIC) does not reset the debug controls.*

Modos de debug. Debug monitor

- El procesador ejecuta la interrupción tipo 12.
- El SYSTICK si corriendo.
- Las nuevas interrupciones serán atendidas o demoradas en función de la configuración del NVIC.
- El contenido de la memoria puede ser alterado por el handler de debug.

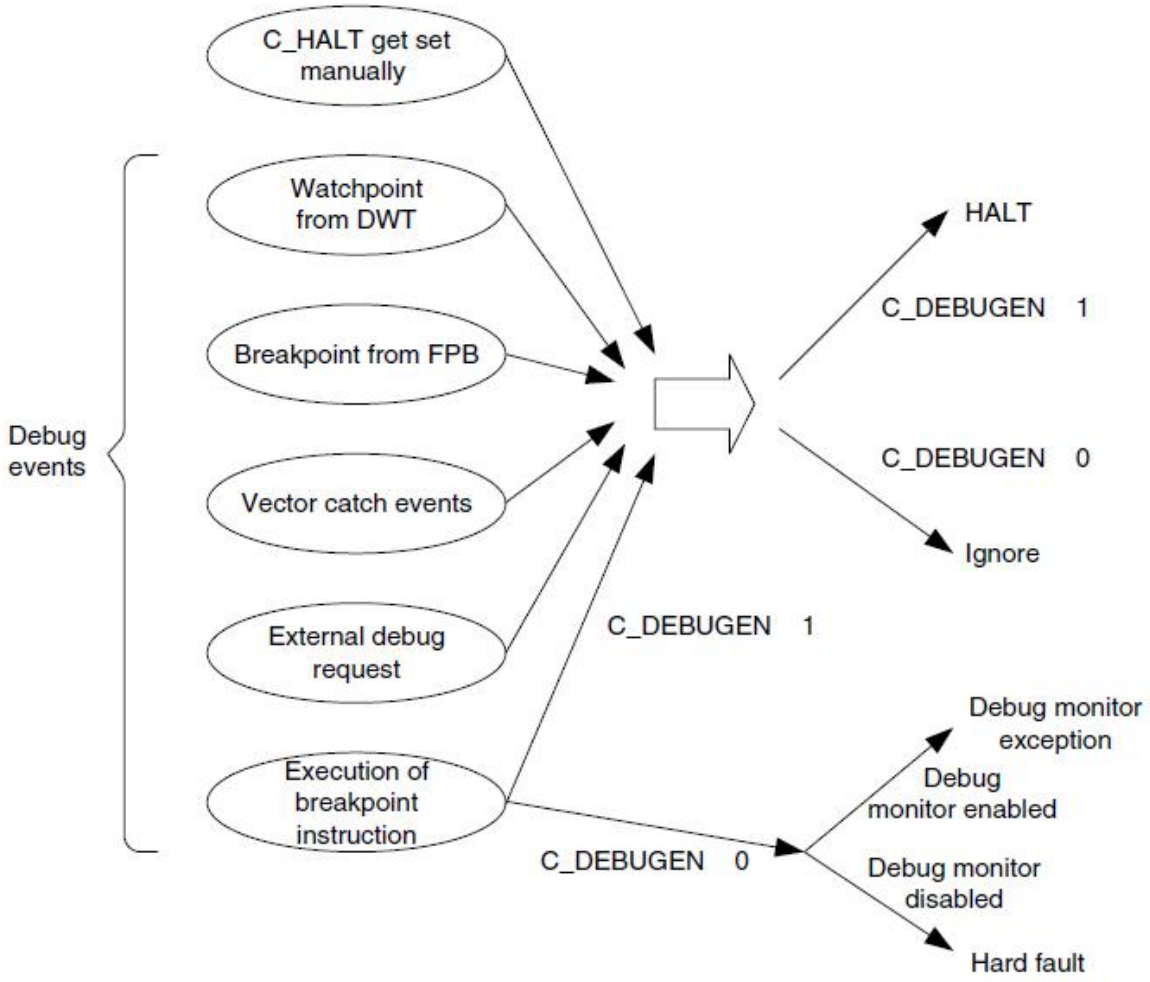
Modos de Debug. Debug Monitor.

Table 15.2 Debug Exception and Monitor Control Register (0xE000EDFC)

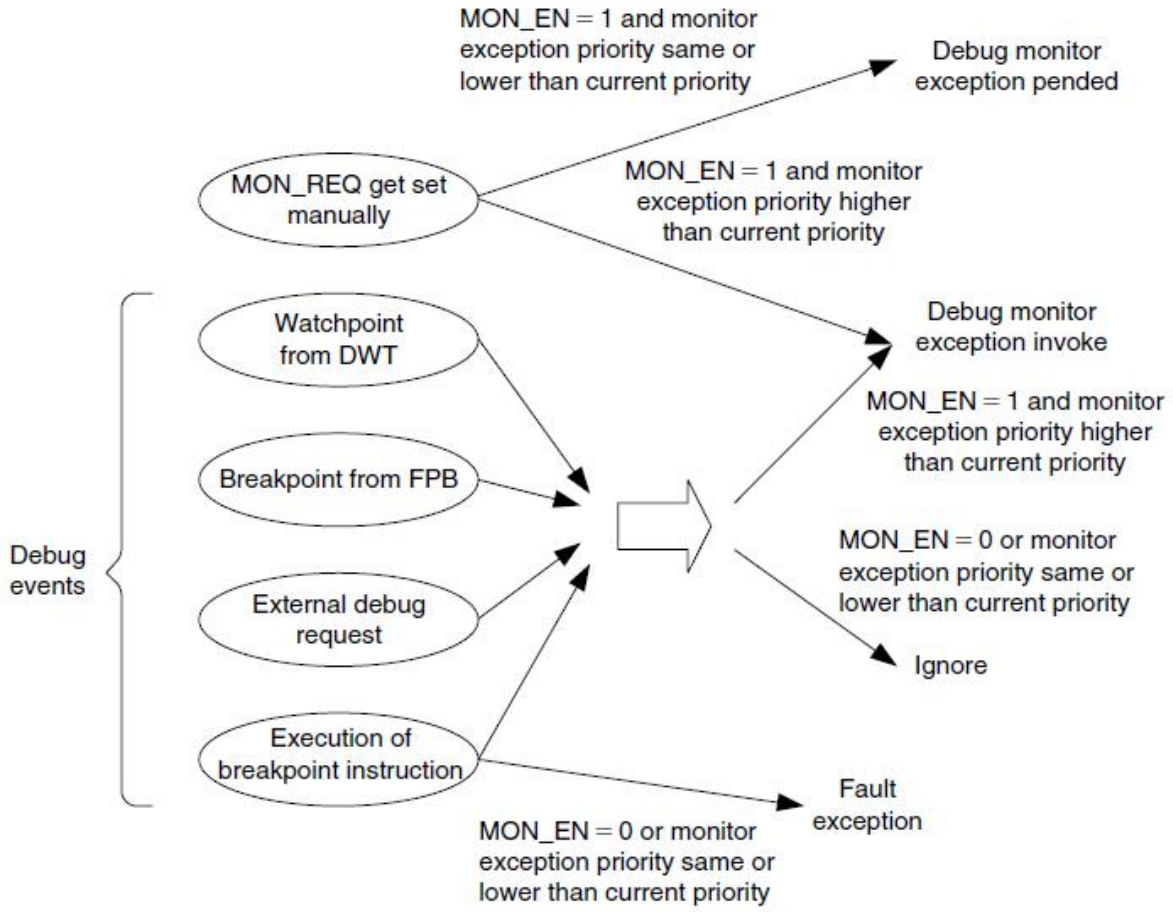
Bits	Name	Type	Reset Value	Description
24	TRCENA	R/W	0*	Trace system enable; to use DWT, ETM, ITM, and TPIU, this bit must be set to 1
23:20	Reserved	—	—	Reserved
19	MON_REQ	R/W	0	Indication that the debug monitor is caused by a manual pending request rather than hardware debug events
18	MON_STEP	R/W	0	Single step the processor; valid only if MON_EN is set
17	MON_PEND	R/W	0	Pend the monitor exception request; the core will enter monitor exceptions when priority allows
16	MON_EN	R/W	0	Enable the debug monitor exception
15:11	Reserved	—	—	Reserved
10	VC_HARDERR	R/W	0*	Debug trap on hard faults
9	VC_INTERR	R/W	0*	Debug trap on interrupt/exception service errors
8	VC_BUSERR	R/W	0*	Debug trap on bus faults
7	VC_STATERR	R/W	0*	Debug trap on usage fault state errors
6	VC_CHKERR	R/W	0*	Debug trap on usage fault-enabled checking errors (e.g., unaligned, divide by zero)
5	VC_NOCPERR	R/W	0*	Debug trap on usage fault, no coprocessor errors
4	VC_MMERR	R/W	0*	Debug trap on memory management fault
3:1	Reserved	—	—	Reserved
0	VC_CORERESET	R/W	0*	Debug trap on core reset

** The control bit in DHCSR is reset by power on reset. System reset (for example, by the Application Interrupt and Reset Control register of NVIC) does not reset the debug controls.*

Debug. Eventos en Halt.



Debug. Eventos en debug monitor.



Debug. Lectura y escritura de registros.

Table 15.3 Debug Core Register Selector Register (0xE000EDF4)

Bits	Name	Type	Reset Value	Description
16	REGWnR	W	—	Direction of data transfer: Write = 1, Read = 0
15:5	Reserved	—	—	—
4:0	REGSEL	W	—	Register to be accessed: 00000 = R0 00001 = R1 . . . 01111 = R15 10000 = xPSR/flags 10001 = Main Stack Pointer (MSP) 10010 = Process Stack Pointer (PSP) 10100 = Special registers: [31:24] Control [23:16] FAULTMASK [15:8] BASEPRI [7:0] PRIMASK Other values are reserved

Table 15.4 Debug Core Register Data Register (0xE000EDF8)

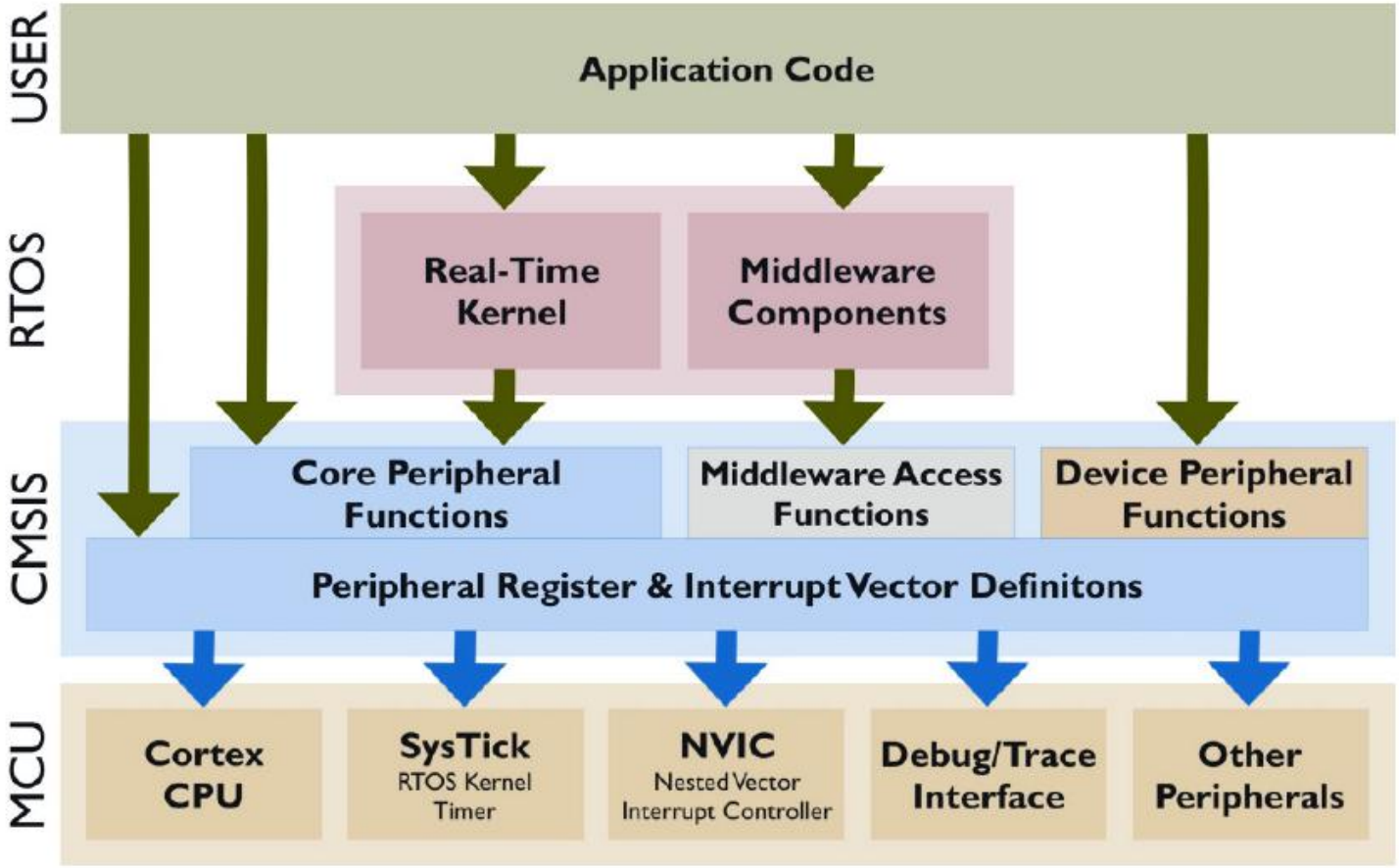
Bits	Name	Type	Reset Value	Description
31:0	Data	R/W	—	Data register to hold register read result or to write data into selected register

CMSIS

- CMSIS: Cortex Microcontroller Software Interface Standard. La CMSIS es una capa de abstracción de hardware para los procesadores Cortex-M .

- La CMSIS es una biblioteca de software escalable, por lo que se definen los siguientes componentes.

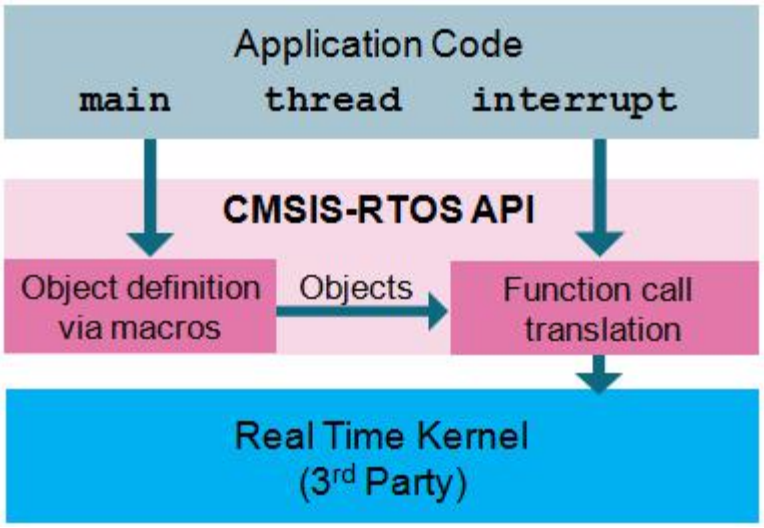
CMSIS


USER
Application Code
RTOS
Real-Time Kernel
Middleware Components
CMSIS
Core Peripheral Functions
Middleware Access Functions
Device Peripheral Functions
Peripheral Register & Interrupt Vector Definitons
MCU
Cortex CPU
SysTick
RTOS Kernel Timer
NVIC
Nested Vector Interrupt Controller
Debug/Trace Interface
Other Peripherals

CMSIS

- ***CMSIS-CORE***. Provee interfaz con los registros y perifericos (NVIC, Systick, Interfaz de DEbug) de los procesadores M0, M3 y M4.
- ***CMSIS-DSP***. Es una biblioteca de software que provee funciones (más de 60) de procesamiento de señales en punto fijo y punto flotante (32 bits).
- ***CMSIS-RTOS API***. Provee una interfaz para control de hilos, procesos y recursos para SOs.
- ***CMSIS-SVD***. System View Description XML. Archivos que contienen la descripción del microcontrolador desde el punto de vista del programador. Formaliza con gran detalle periféricos.

CMSIS. RTOS



- Define de manera mínima el control de hilos y sincronización. El resto debe proveerlo el RTOS en uso.

CMSIS. Archivos.

File	Provider	Description
device.h	Device specific (provided by silicon partner)	Defines the peripherals for the actual device. The file may use several other include files to define the peripherals of the actual device.
core_cm0.h	ARM (for RealView ARMCC, IAR, and GNU GCC)	Defines the core peripherals for the Cortex-M0 CPU and core peripherals.
core_cm3.h	ARM (for RealView ARMCC, IAR, and GNU GCC)	Defines the core peripherals for the Cortex-M3 CPU and core peripherals.
core_cm0.c	ARM (for RealView ARMCC, IAR, and GNU GCC)	Provides helper functions that access core registers.
core_cm3.c	ARM (for RealView ARMCC, IAR, and GNU GCC)	Provides helper functions that access core registers.
startup_*device*	ARM (adapted by compiler partner / silicon partner)	Provides the Cortex-M startup code and the complete (device specific) Interrupt Vector Table
system_*device*	ARM (adapted by silicon partner)	Provides a device specific configuration file for the device. It configures the device initializes typically the oscillator (PLL) that is part of the microcontroller device

CMSIS. Definición de Interrupciones.

```
typedef enum IRQn
{
/******  Cortex-M3 Processor Exceptions/Interrupt Numbers ***********************************************/
  NonMaskableInt_IRQn             = -14,      /*!< 2 Non Maskable Interrupt                                */
  HardFault_IRQn                  = -13,      /*!< 3 Cortex-M3 Hard Fault Interrupt                        */
  MemoryManagement_IRQn           = -12,      /*!< 4 Cortex-M3 Memory Management Interrupt                 */
  BusFault_IRQn                   = -11,      /*!< 5 Cortex-M3 Bus Fault Interrupt                         */
  UsageFault_IRQn                 = -10,      /*!< 6 Cortex-M3 Usage Fault Interrupt                       */
  SVCall_IRQn                     = -5,       /*!< 11 Cortex-M3 SV Call Interrupt                          */
  DebugMonitor_IRQn               = -4,       /*!< 12 Cortex-M3 Debug Monitor Interrupt                    */
  PendSV_IRQn                     = -2,       /*!< 14 Cortex-M3 Pend SV Interrupt                          */
  SysTick_IRQn                    = -1,       /*!< 15 Cortex-M3 System Tick Interrupt                      */
/******  STM32 specific Interrupt Numbers ***************************************************************/
  WWDG_STM_IRQn                   = 0,        /*!< Window WatchDog Interrupt                               */
  PVD_STM_IRQn                    = 1,        /*!< PVD through EXTI Line detection Interrupt               */
  :
  :
  } IRQn_Type;
```

CMSIS. Funciones para inicialización.

Function Definition	Description
void SystemInit (void)	Setup the microcontroller system. Typically this function configures the oscillator (PLL) that is part of the microcontroller device. For systems with variable clock speed it also updates the variable SystemCoreClock. SystemInit is called from startup_*device* file.
void SystemCoreClockUpdate (void)	Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution. SystemCoreClockUpdate() evaluates the clock register settings and calculates the current core clock.

Variable Definition	Description
uint32_t SystemCoreClock	Contains the system core clock (which is the system clock frequency supplied to the SysTick timer and the processor core clock). This variable can be used by the user application to setup the SysTick timer or configure other parameters. It may also be used by debugger to query the frequency of the debug timer or configure the trace clock speed. SystemCoreClock is initialized with a correct predefined value. The compiler must be configured to avoid the removal of this variable in case that the application program is not using it. It is important for debug systems that the variable is physically present in memory so that it can be examined to configure the debugger.

CMSIS. Systick

Name	Parameter	Description
uint32_t SysTickConfig (uint32_t ticks)	ticks is SysTick counter reload value	Setup the SysTick timer and enable the SysTick interrupt. After this call the SysTick timer creates interrupts with the specified time interval. Return: 0 when successful, 1 on failure.

CMSIS. NVIC

Name	Core	Parameter	Description
void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)	M3	Priority Grouping Value	Set the Priority Grouping (Groups . Subgroups)
uint32_t NVIC_GetPriorityGrouping (void)	M3	(void)	Get the Priority Grouping (Groups . Subgroups)
void NVIC_EnableIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Enable IRQn
void NVIC_DisableIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Return 1 if IRQn is pending else 0
void NVIC_SetPendingIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Set IRQn Pending
void NVIC_ClearPendingIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Clear IRQn Pending Status
uint32_t NVIC_GetActive (IRQn_Type IRQn)	M3	IRQ Number	Return 1 if IRQn is active else 0
void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)	M0, M3	IRQ Number, Priority	Set Priority for IRQn (not threadsafe for Cortex-M0)
uint32_t NVIC_GetPriority (IRQn_Type IRQn)	M0, M3	IRQ Number	Get Priority for IRQn
uint32_t NVIC_EncodePriority (uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority)	M3	IRQ Number, Priority Group, Preemptive Priority, Sub Priority	Encode priority for given group, preemptive and sub priority
NVIC_DecodePriority (uint32_t Priority, uint32_t PriorityGroup, uint32_t* pPreemptPriority, uint32_t* pSubPriority)	M3	IRQ Number, Priority, pointer to Priority Group, pointer to Preemptive Priority, pointer to Sub Priority	Deccode given priority to group, preemptive and sub priority
void NVIC_SystemReset (void)	M0, M3	(void)	Resets the System

Consideraciones para codificar en C sobre procesadores ARM.

- Existen muchas consideraciones básicas de programación para procesadores ARM, la documentación del compilador usado aporta gran detalle sobre diversos aspectos importantes (y sutiles) a considerar.
- Las consideraciones de codificación son muy importantes cuando se procura optimizar el rendimiento y ocupar cantidades de memoria limitadas.
- Es útil conocer el assembler del procesador y el assembler generado por el compilador para aprender acerca de la eficiencia y lo que hace el compilador.
- Hay que balancear claridad en la codificación, optimizaciones e información para debug de acuerdo a las condiciones particulares de cada proyecto.

Consideraciones de lazos.

- La gran mayoría de los programas embebidos se pueden modelar con la regla 90/10: el 90% del tiempo se ejecuta el 10% del código. La mayor parte del tiempo el programa se encuentra ejecutando lazos.
- Una optimización (posible) puede ser la siguiente:

```
int i;
for (i=0;i<n;i++);

            MOVS    r1, #0x00
            B       test
lazo:       ADDS    r0, r0 , #1
test:       CMP     r1, r0
            BLT     lazo
```

```
char i;
for (i=0;i<n;i++);

            MOVS    r1, #0x00
            B       test
lazo:       ADDS    r2,r0,#1
            UXTB    r0,r2
test:       CMP     r1,r0
            BLT     lazo
```

Asignación de registros y alias de punteros.

- En ARM las variables pueden alocarse en registros en lugar de memoria (con gran ahorro en rendimiento y tamaño de código).
- Por la consistencia de valor de la variable, el compilador debe asegurarse que el valor de la variable en memoria no cambia por una referencia de puntero en otros puntos del programa, por lo tanto una variable alocada en un registro debe:
  - Ser una **variable local** o un **parámetro de función.**
  - No tienen su dirección tomada **(&var_name)** o asignada a otra variable.
- Al crear un puntero a una variable, el compilador no sabe si va a cambiar su valor en otro punto, por lo que no puede copiarla a un registro.

Asignación de registros y alias de punteros.

- Las variables globales no pueden alocarse en registros pues sus valores no son privados a un módulo determinado.
- Ejemplo de asignación de alias de puntero:
  - **int prog1 (int var1) {funcion(&var1);}** En este caso el compilador no puede determinar si un valor de registro va a cambiar o no, entonces va a usar paso por memoria.
  - **int prog1 (int var2) {int local1 = var2; funcion(&local1); var2=local1;}** En este caso al usar la variable intermedia, puede hacer el pasaje por registro.

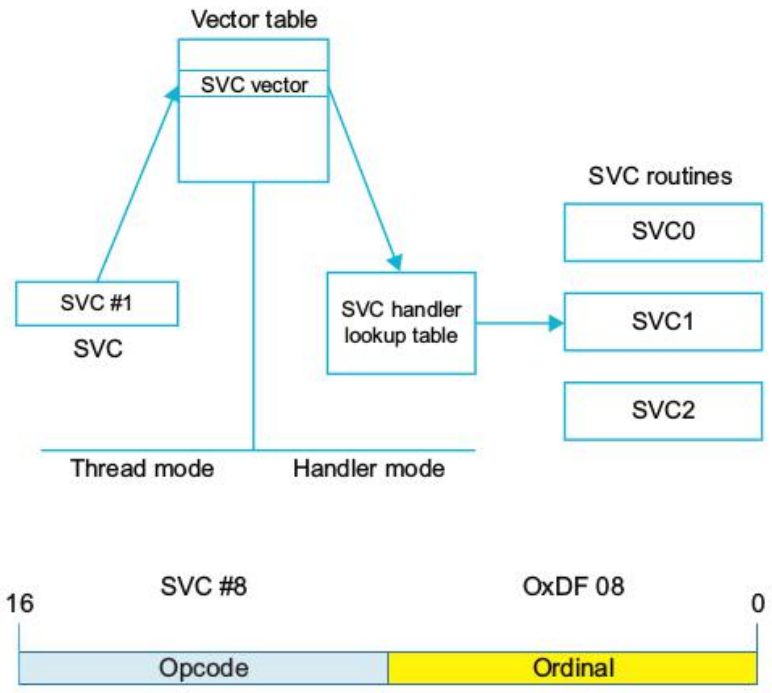
Tipos de datos en ARM.

- Los procesadores ARM son procesadores de 32bits, por lo que todo lo que se haga con este tipo de datos va a utilizar menos código que con el uso de otros tipos de datos.
  - ***int masuno(int i) { return i++; }***
  - Generará:
    - ADDS r1,r0,#1
    - BX lr
  - ***short masuno(short i) { return i++; }***
  - Generará:
    - MOV r1,r0
    - ADDS r0,r1,#1
    - SXTH r0,r0
    - MOV r1,r0
    - BX lr

Cortex-M. Modos de operación.

Modes (Thread out of reset)		Operations (Privilege out of reset)	Stacks (Main out of reset)
	Handler - Processing of exceptions	Privileged execution full control	Main stack used by OS and exceptions
	Thread - No exception is being processed - Normal code execution	Privileged or unprivileged	Main or process

Cortex-M. Supervisor Call (SVC)

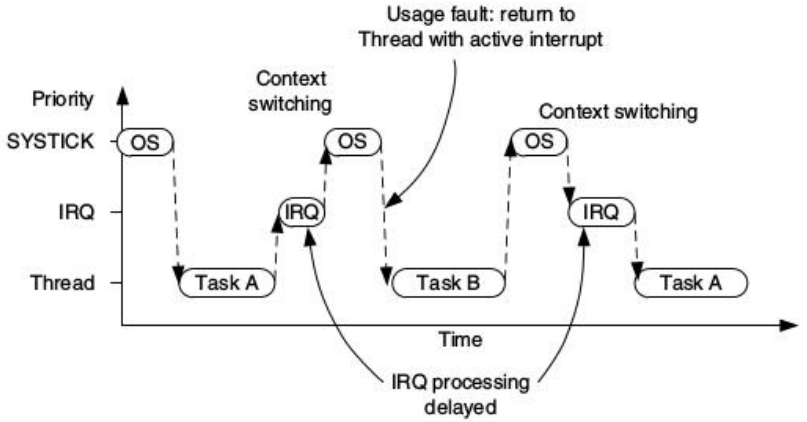


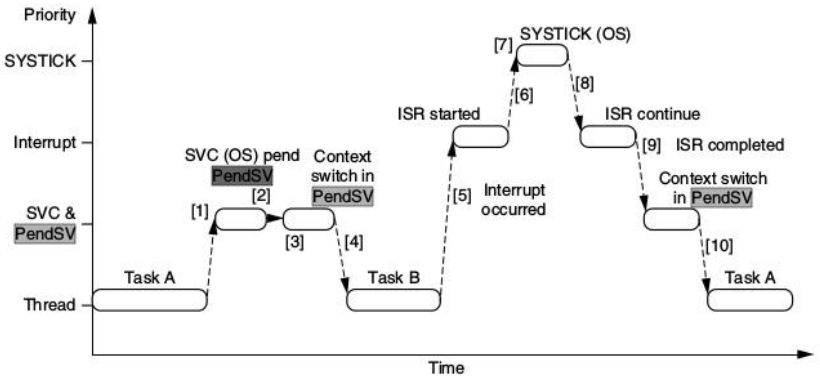
- Una vez configurado en modo thread sin privilegios, el único camino para volver a obtenerlos es una excepción.
- La supervisor CALL es este camino.
- En su código de operación permite codificar un número de 8 bits, lo que permite parametrizar la llamada.

Cortex-M. Pendable Service Call

- La Pend_SVC es una excepción que se tendría que ejecutar si ninguna otra excepción tiene lugar.
- El uso normal de la Pend_SVC es con sistemas operativos para generar los cambios de contexto.
- La idea detrás de esta excepción es minimizar la cantidad de cambios de contexto del procesador.

Cortex-M. Pendable Service Call





Bibliografía.

- The Definitive Guide to the ARM Cortex-M3, Second Edition – Joseph Yiu – Newnes – 2009.
- The Definitive Guide to the ARM Cortex-M0. Joseph Yiu. Elsevier. 2011.
- The Designer's Guide to the Cortex-M Processor Family. Trevor Martin. Elsevier. 2013
- LPC17xx User manual.
- ARM®v7-M Architecture Reference Manual.
- Cortex™-M3 Revision: r1p1 Technical Reference Manual.