Cortex-M3 Architecture: Introduction to the LPC1768 microcontroller

Index

- **1.1.** Introduction: Embedded Systems + Design of Digital Systems. The ARM Cortex-M3
- **1.2.** LPC1768: Block Diagram, Memory Map, Busses, Pinout
- **1.3.** LPC1768 exceptions: Types and Vectors, NVIC, Enter+Exit, Priority, CMSIS functions.
- **1.4.** LPC1768 system control modules: Clocking Features, RESET, FAULTs, SYSTICK, Power Management.

1.1. Embedded Systems

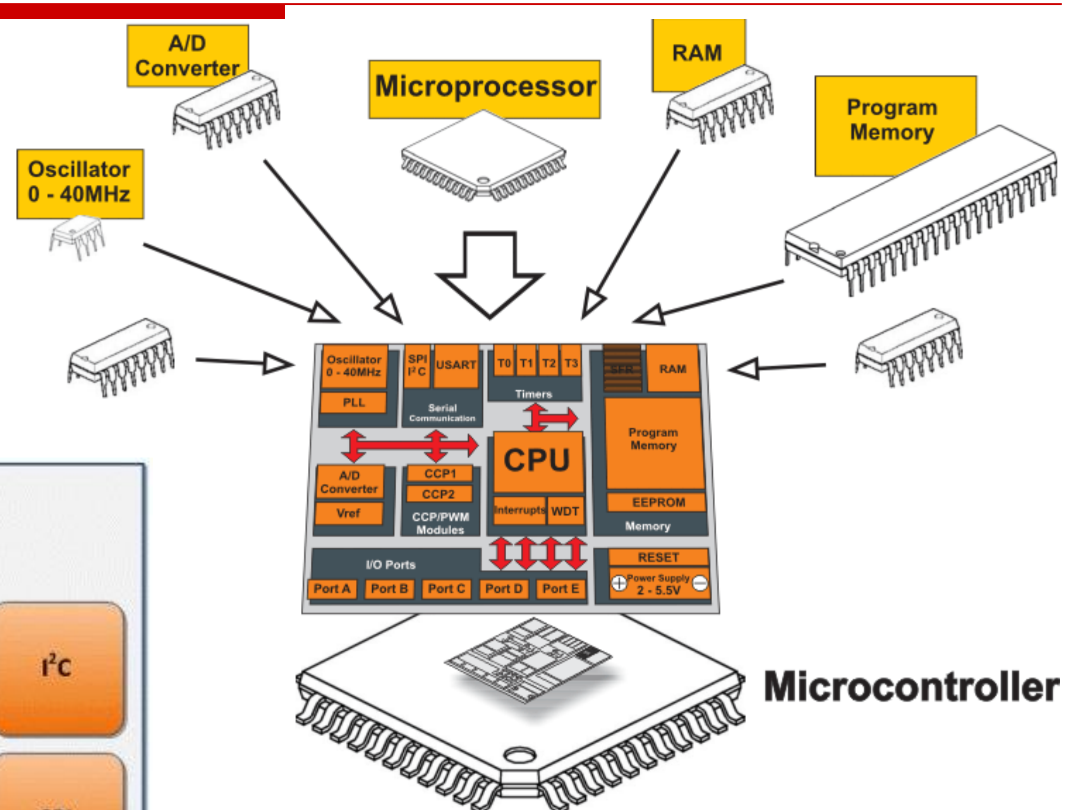
What is an Embedded System?

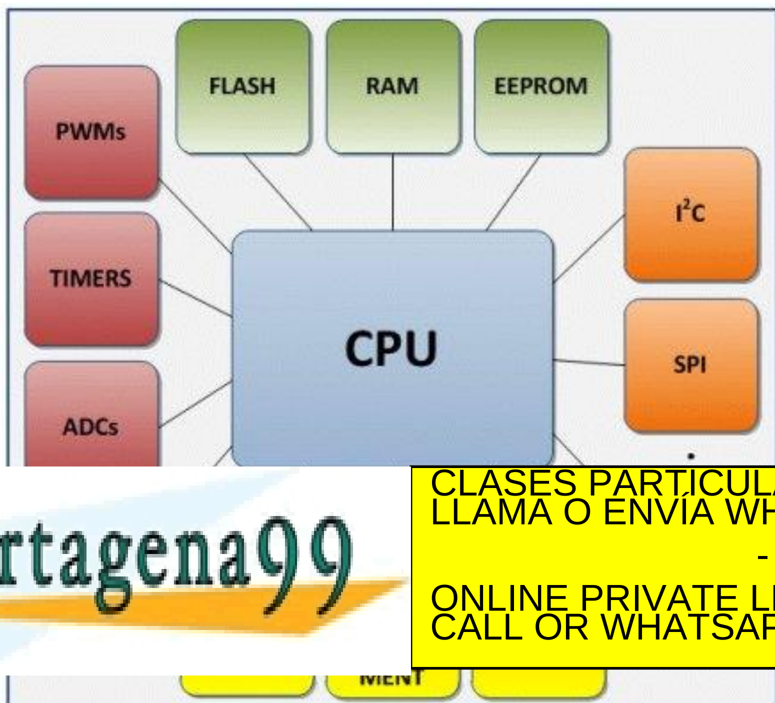
- Interacts with the environment
- Is divided into three stages: 1 INPUT + 2 PROCESS + 3 OUTPUT
  - The processing is based on:
  - Combinational logic and sequential circuits
  - Microprocessors (µP) - Microcontrollers (µC)
  - Digital Signal Processor (DSP)
  - Programmable Logic Devices (PLDs)
  - Programmable Logic Controllers (PLCs)
- **In general, they are real-time reactive systems:**
  - They react to external events and keep continuous interaction
  - They are continuously running

1.1. Design of Digital Systems

- A microcontroller (μC) is a chip **that includes in a single chip** all elements needed in a digital system
  - the processor (μP, CPU)
  - different types and amounts of memory and
  - various input/output interfaces and peripherals
  - All of them interconnected by uni/bidirectional busses
- Therefore:
  - Achieving more integration and lower price
  - Lower time to marked when implementing a project

1.1. Design of Digital Systems: What is Microcontroller?

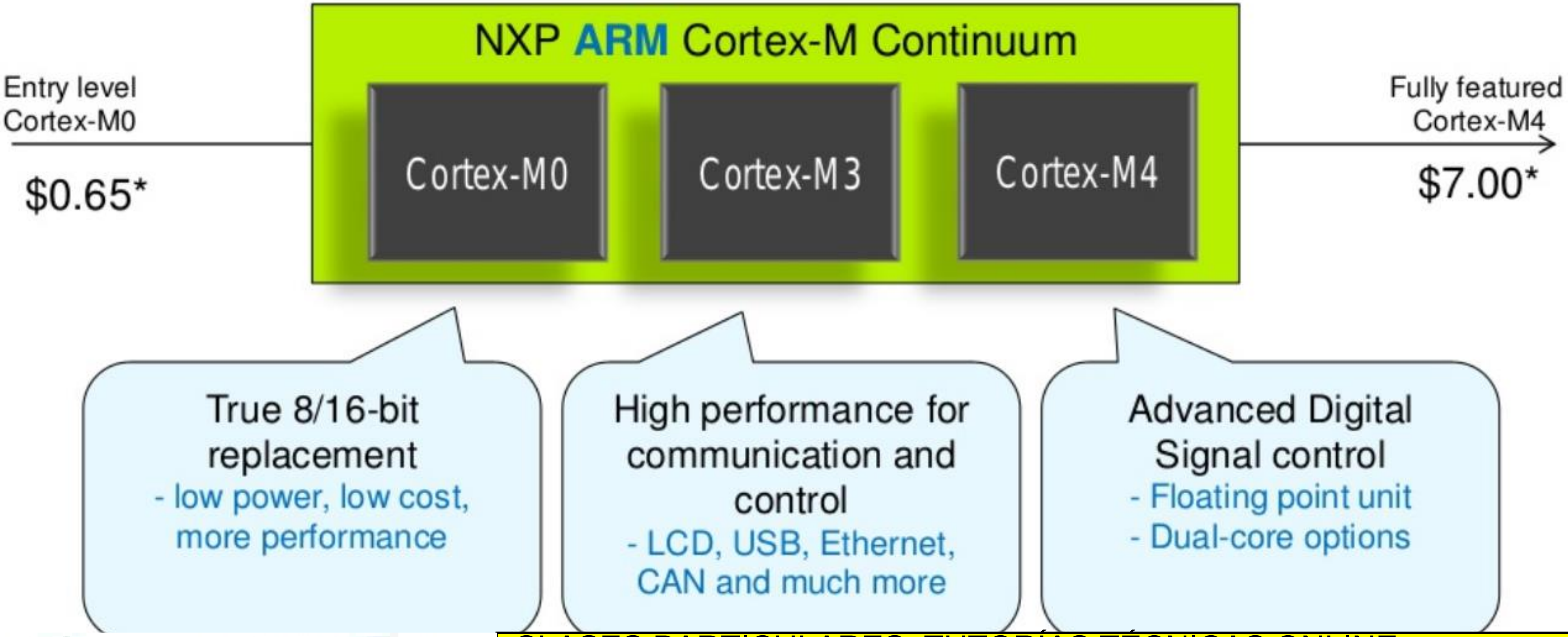




1.1. The ARM Cortex-M3

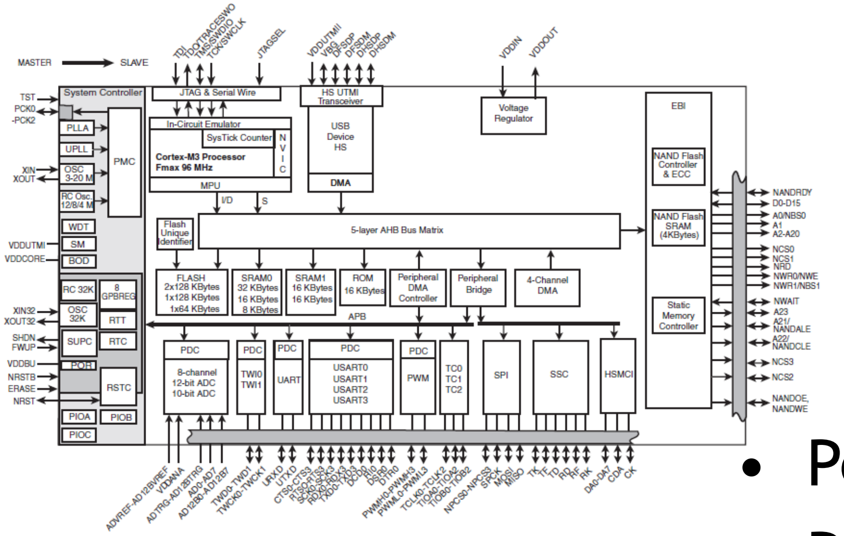
- ARM is a:
  - RISC µP
  - 17/18 32 bits visible registers in its programmer's model (37 total)
  - Caché Memory (upon version)
  - Von Neuman architecture (ARM7)
  - Harvard architecture (ARM9 and forward)
- The Cortex-M3 µC builds on the success of the ARM7
  - Nonmaskable interrupts for critical tasks
  - Deterministic nested vector exceptions
  - Atomic bit manipulation
  - Optional Memory Protection Unit (MPU)

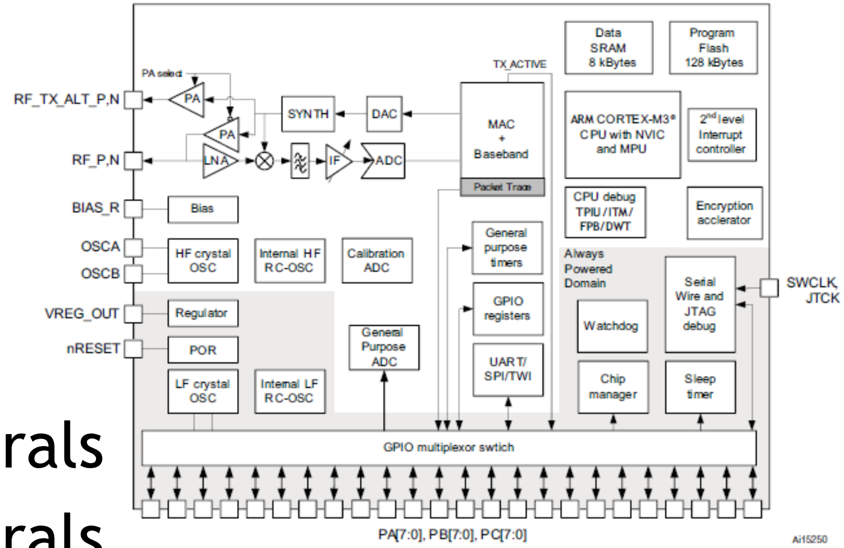
1.1. ARM Cortex Advanced Processors



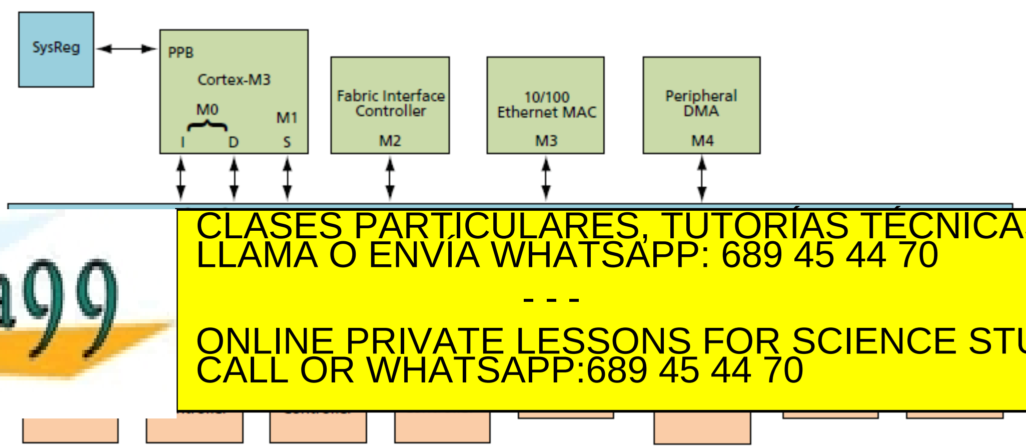
1.1. Cortex-M3: Manufacturers

1.1. Manufacturers that incorporate Cortex-M3: Differences?

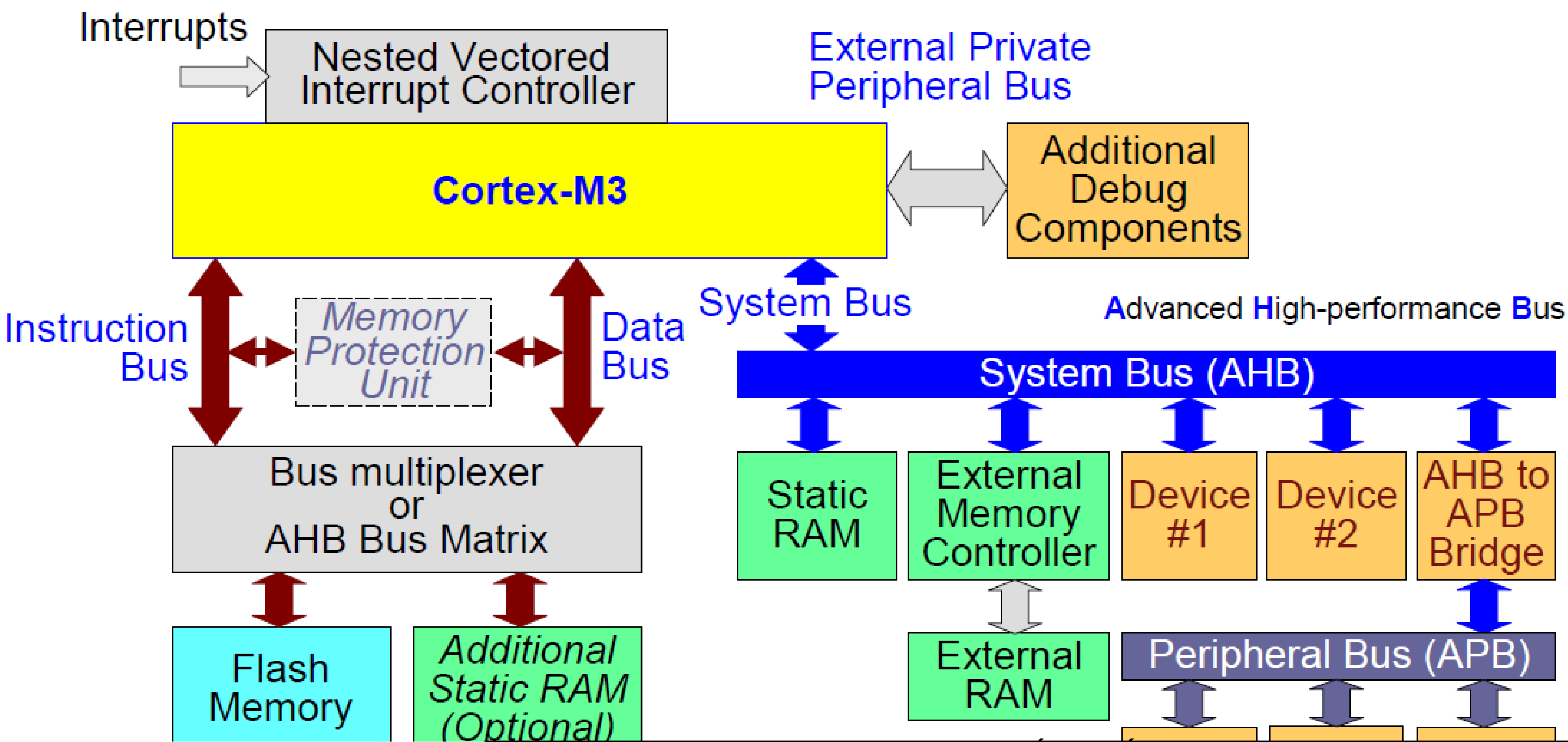




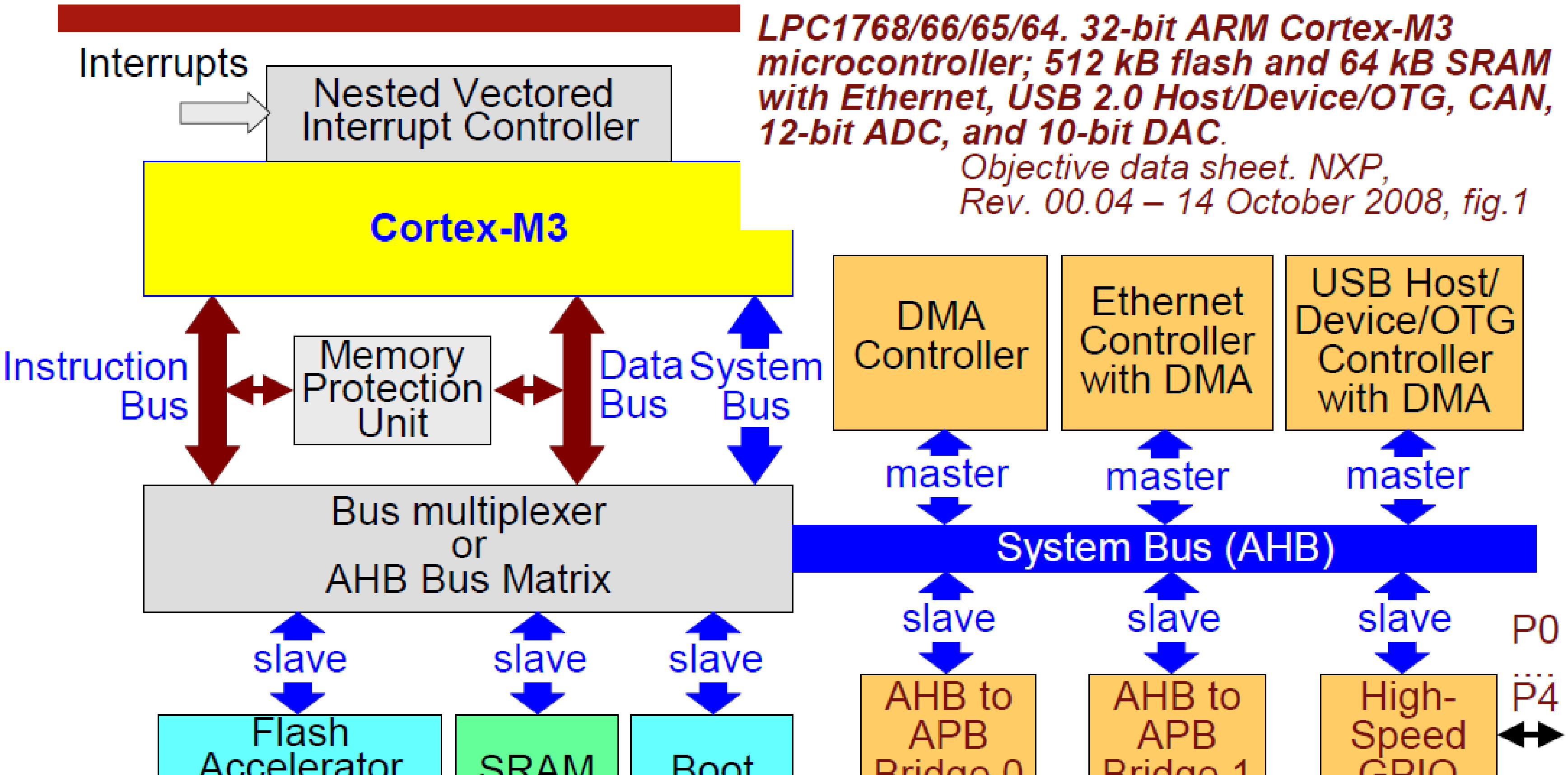
- Peripherals
- Peripherals
- Peripherals



1.2. Cortex-M3: Busses



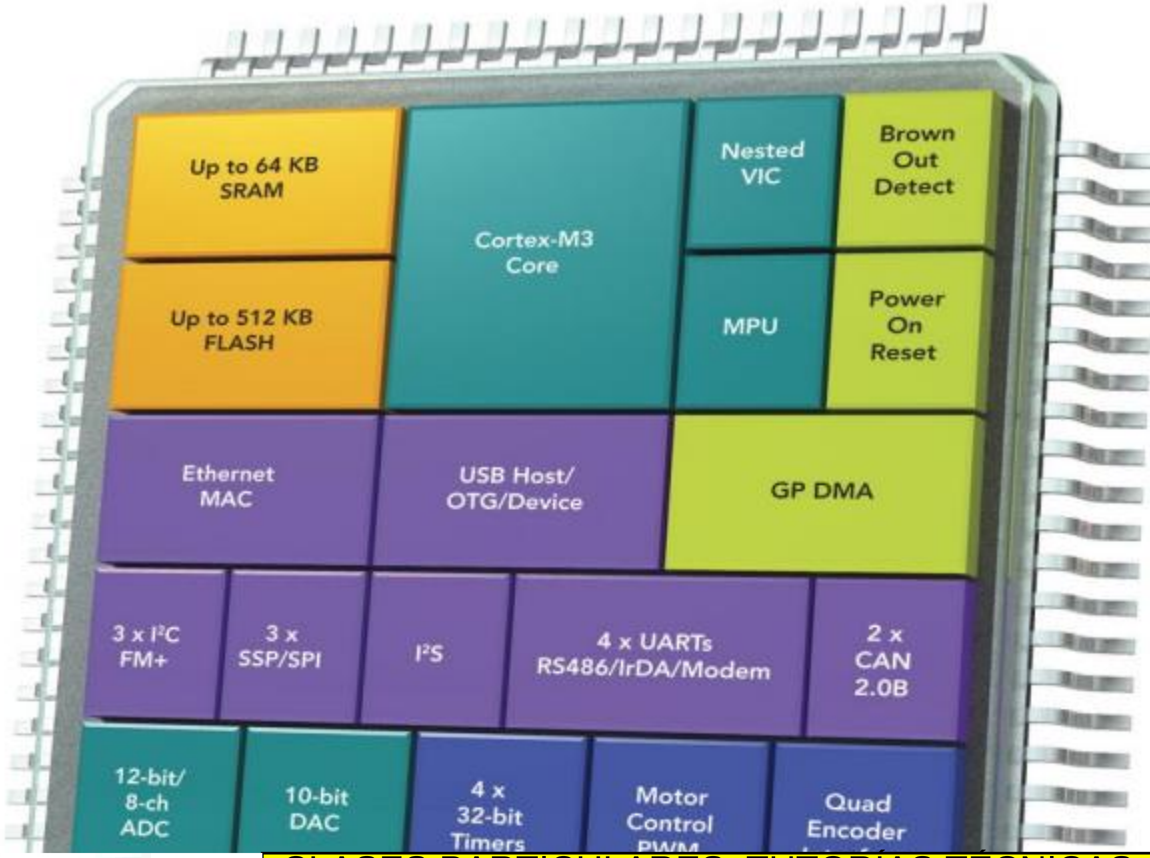
1.2. LPC17xx: Busses



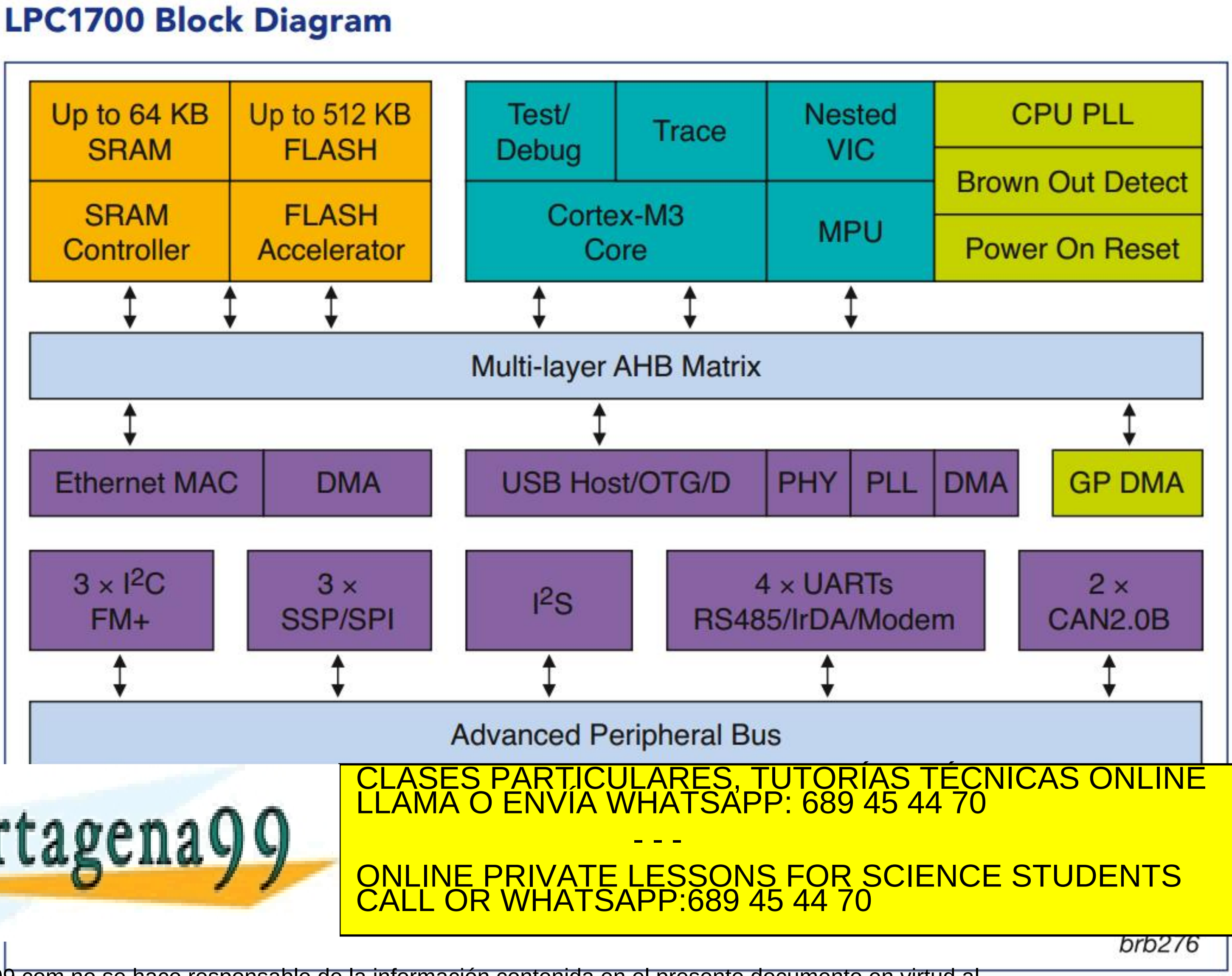
LPC1768/66/65/64. 32-bit ARM Cortex-M3 microcontroller; 512 kB flash and 64 kB SRAM with Ethernet, USB 2.0 Host/Device/OTG, CAN, 12-bit ADC, and 10-bit DAC.

Objective data sheet. NXP, Rev. 00.04 – 14 October 2008, fig.1

1.2. LPC17xx: Block Diagram



1.2. LPC17xx: Block Diagram



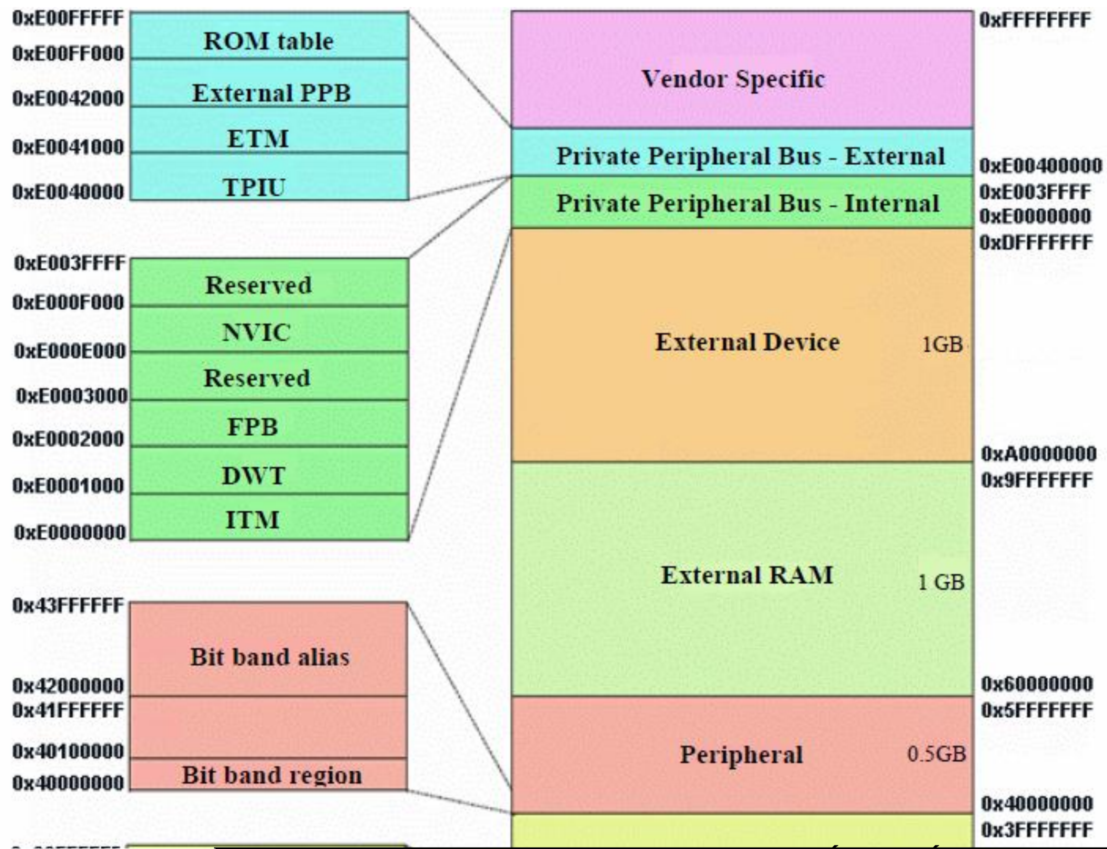
1.2. LPC17xx: Memory

- **On-chip Flash**
  - Maximum 512 KB.
  - Zero wait-state performance with Flash Accelerator.
- **On-Chip SRAM**
  - Maximun 64 KB:
    - 32 KB SRAM – accessible by the CPU and DMA controller on a higher speed bus.
    - Two additional 16 KB SRAM – separate slave port on the AHB multilayer matrix.
    - Allows CPU and DMA accesses to be spread over 3 separate RAMs that can be accessed simultaneously.
- **On-Chip ROM**

1.2. LPC17xx: Family

Part Number	Max Clock (MHz)	Flash (KB)	SRAM (KB)	Ethernet	USB	CAN	I²S	ADC	DAC	I²C	I/O Pins	Package
LPC1769	120	512	64	Y	Device/Host/OTG	2	Y	8	Y	3	70	LQFP100
LPC1768	100	512	64	Y	Device/Host/OTG	2	Y	8	Y	3	70	LQFP100
LPC1767	100	512	64	Y	None	0	Y	8	Y	3	70	LQFP100
LPC1766	100	256	64	Y	Device/Host/OTG	2	Y	8	Y	3	70	LQFP100
LPC1765	100	256	64	N	Device/Host/OTG	2	Y	8	Y	3	70	LQFP100
LPC1764	100	128	32	Y	Device	2	N	8	N	3	70	LQFP100
LPC1759	120	512	64	N	Device/Host/OTG	2	Y	6	Y	2	52	LQFP80
LPC1758	100	512	64	Y	Device/Host/OTG	2	Y	6	Y	2	52	LQFP80
LPC1756	100	256	32	N	Device/Host/OTG	2	Y	6	Y	2	52	LQFP80
LPC1754	100	128	32	N	Device/Host/OTG	1	N	6	Y	2	52	LQFP80
LPC1752	100	64	16	N	Device	1	N	6	N	2	52	LQFP80
LPC1751	100	32	8	N	Device	1	N	6	N	2	52	LQFP80

1.2. Cortex-M3: Memory Map

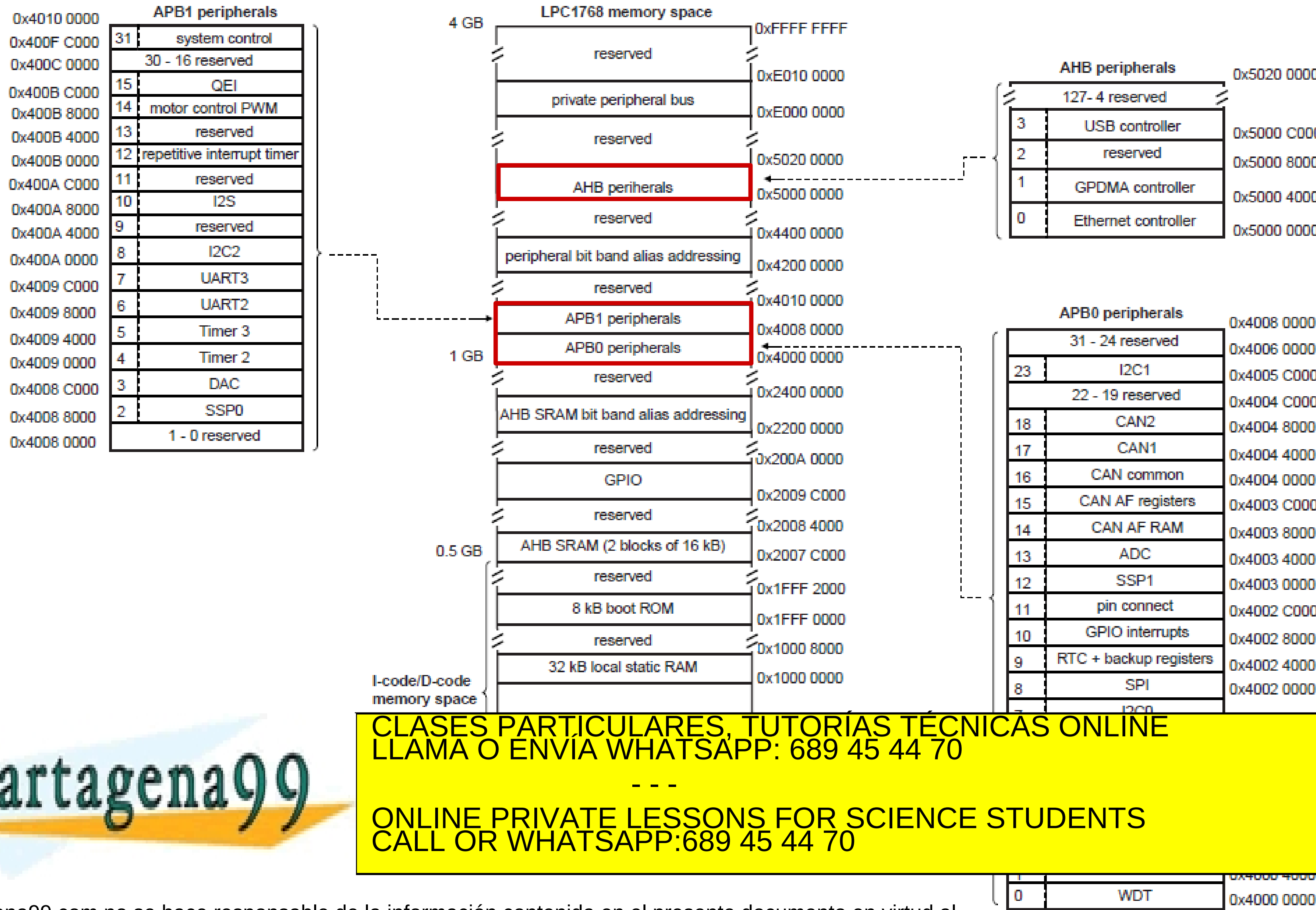


1.2. LPC17xx: Memory Map

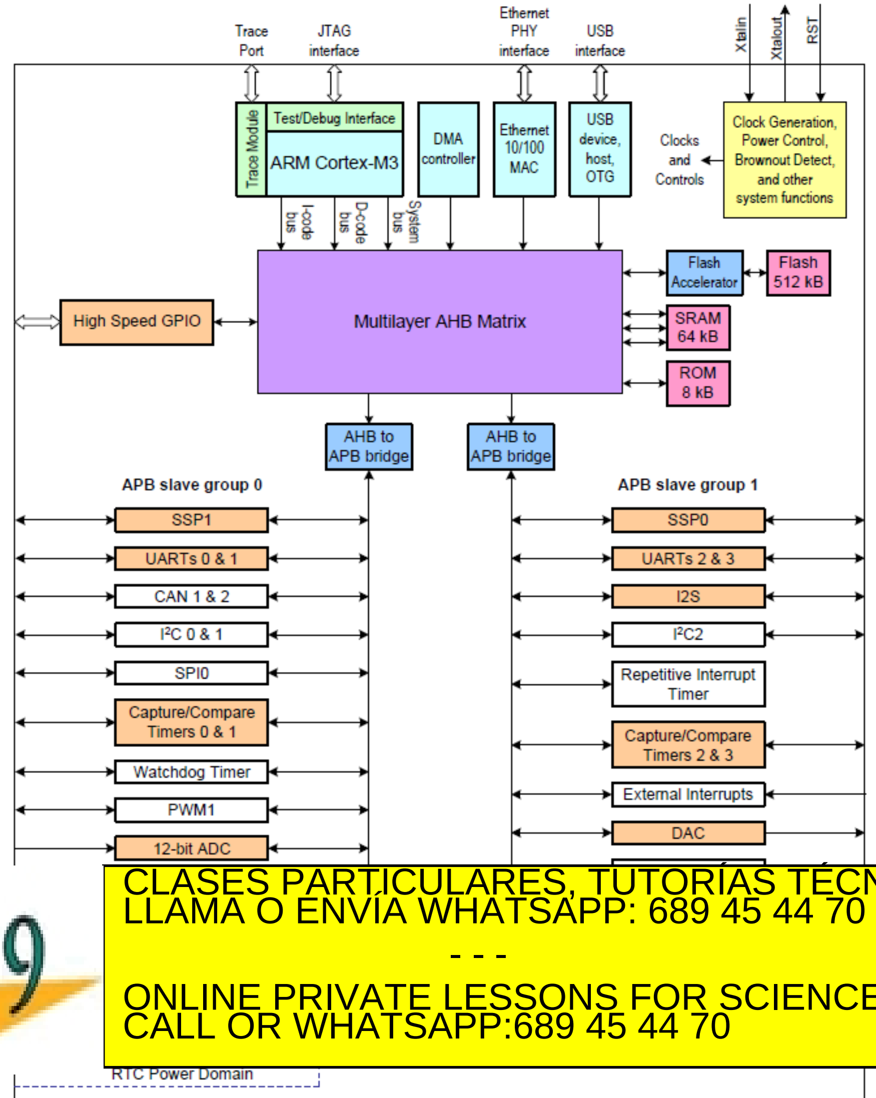
Table 3. LPC17xx memory usage and details

Address range	General Use	Address range details and description	
0x0000 0000 to 0x1FFF FFFF	On-chip non-volatile memory	0x0000 0000 - 0x0007 FFFF	For devices with 512 kB of flash memory.
		0x0000 0000 - 0x0003 FFFF	For devices with 256 kB of flash memory.
		0x0000 0000 - 0x0001 FFFF	For devices with 128 kB of flash memory.
		0x0000 0000 - 0x0000 FFFF	For devices with 64 kB of flash memory.
		0x0000 0000 - 0x0000 7FFF	For devices with 32 kB of flash memory.
	On-chip SRAM	0x1000 0000 - 0x1000 7FFF	For devices with 32 kB of local SRAM.
		0x1000 0000 - 0x1000 3FFF	For devices with 16 kB of local SRAM.
		0x1000 0000 - 0x1000 1FFF	For devices with 8 kB of local SRAM.
	Boot ROM	0x1FFF 0000 - 0x1FFF 1FFF	8 kB Boot ROM with flash services.
0x2000 0000 to 0x3FFF FFFF	On-chip SRAM (typically used for peripheral data)	0x2007 C000 - 0x2007 FFFF	AHB SRAM - bank 0 (16 kB), present on devices with 32 kB or 64 kB of total SRAM.
		0x2008 0000 - 0x2008 3FFF	AHB SRAM - bank 1 (16 kB), present on devices with 64 kB of total SRAM.
	GPIO	0x2009 C000 - 0x2009 FFFF	GPIO.
0x4000 0000 to 0x5FFF FFFF	APB Peripherals	0x4000 0000 - 0x4007 FFFF	APB0 Peripherals, up to 32 peripheral blocks, 16 kB each.
		0x4008 0000 - 0x400F FFFF	APB1 Peripherals, up to 32 peripheral blocks, 16 kB each.

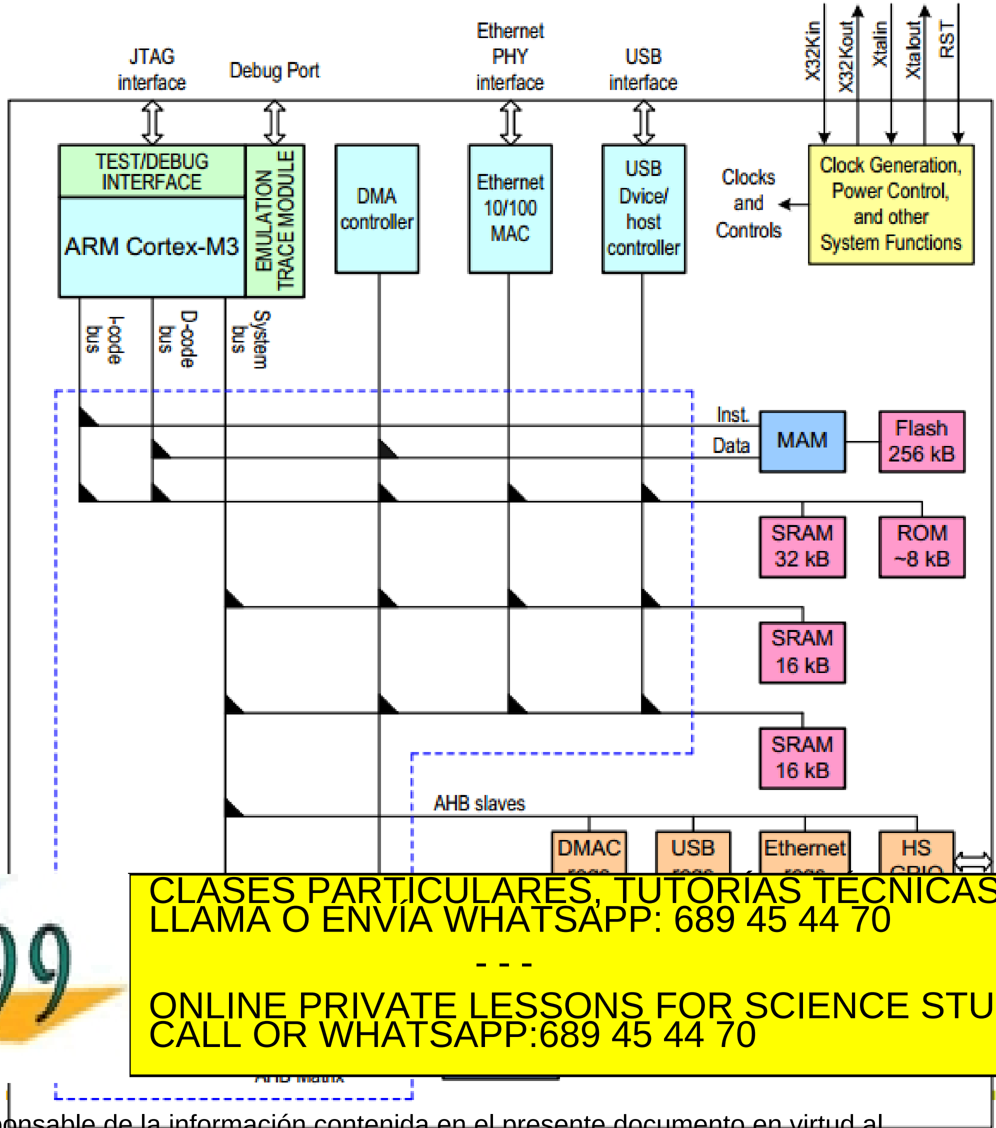
1.2. LPC1768: Memory (Peripherals map)



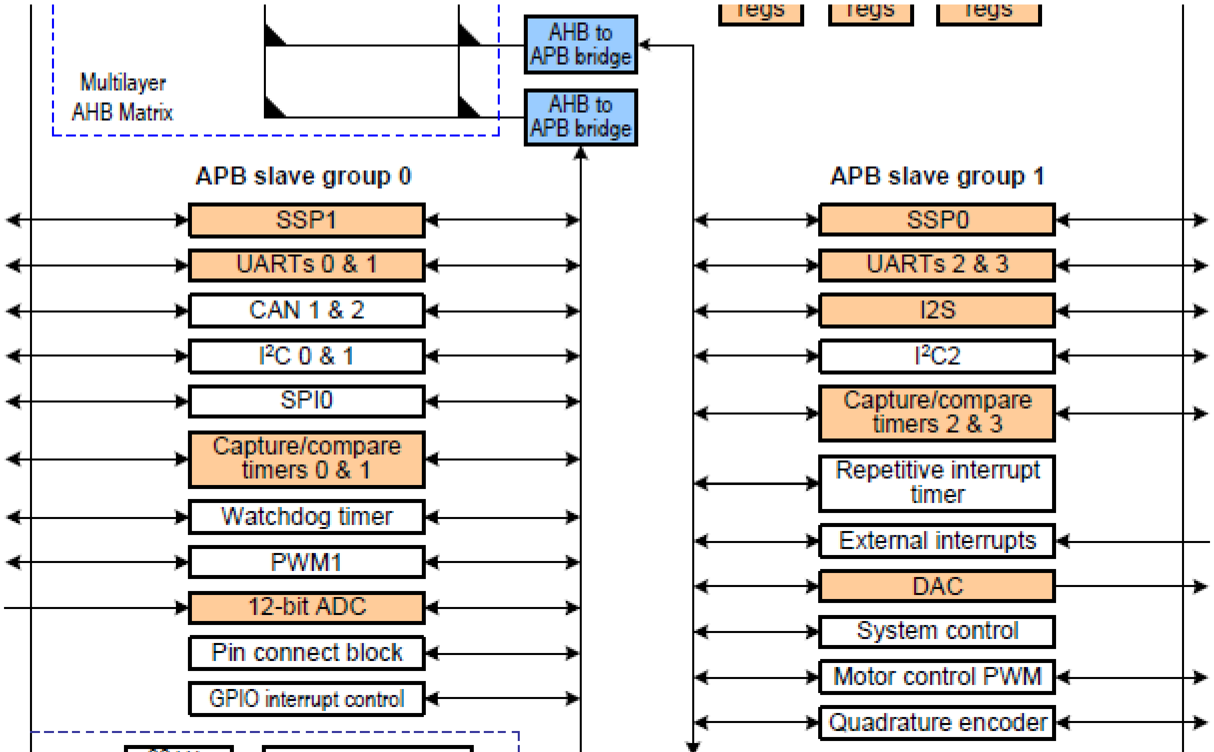
1.2. LPC1768: Bus structure



1.2. LPC1768: Bus structure (Multilayer AHB Matrix)



1.2. LPC1768: Bus structure (Peripheral)



1.2. LPC1768: Pins

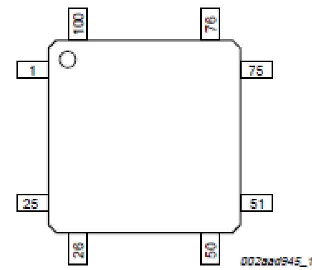
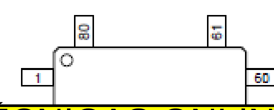


Fig 14. LPC176x LQFP100 pin configuration



1.2. LPC1768: Pin Connect Block

LPC1768 Pin functions

Pin	Func 1	Func 2	Func 3	Func 4	Pin	Func 1	Func 2	Func 3	Func 4
1	TDO	SWO			51	P2.12	EINT2N	I2STX_WS	
2	TDI				52	P2.11	EINT1N	I2STX_CLK	
3	TMS	SWDIO			53	P2.10	EINT0N	NMI	
4	nTRST				54	VDD			
5	TCK	SWDCLK			55	VSS			
6	P0.26	AD0.3	AOUT	RXD3	56	P0.22	RTS1	CAN_TX1	
7	P0.25	AD0.2	I2SRX_SDA	TXD3	57	P0.21	RI1	CAN_RX1	
8	P0.24	AD0.1	I2SRX_WS	CAP3.1	58	P0.20	DTR1	SCL1	
9	P0.23	AD0.0	I2SRX_CLK	CAP3.0	59	P0.19	DSR1	MCICLK	SDA1
10	VDDA				60	P0.18	DCD1	MOSI0	MOSI
11	VSSA				61	P0.17	CTS1	MISO0	MISO
12	VREFP				62	P0.15	TXD1	SCK0	SCK
13	N.C.				63	P0.16	RXD1	SSEL0	SSEL
14	nRSTOUT				64	P2.9	USB_CONNECT	RXD2	
15	VREFN				65	P2.8	CAN_TX2	TXD2	
16	RTCX1				66	P2.7	CAN_RX2	RTS1	
17	nRESET				67	P2.6	PCAP1.0	RI1	TRACECLK
18	RTCX2				68	P2.5	PWM1.6	DTR1	TRACEDATA0
19	VBAT				69	P2.4	PWM1.5	DSR1	TRACEDATA1
20	P1.31	SCK1	AD0.5		70	P2.3	PWM1.4	DCD1	TRACEDATA2
21	P1.30	VBUS	AD0.4		71	VDD			
22	XTAL1				72	VSS			
23	XTAL2				73	P2.2	PWM1.3	CTS1	TRACEDATA3
24	P0.28	SCL0	USB_SCL		74	P2.1	PWM1.2	RXD1	
25	P0.27	SDA0	USB_SDA		75	P2.0	PWM1.1	TXD1	TRACECLK

Table 75. Pin function select register bits

1.2. LPC17xx: Pin Connect Block

Ex: **P0.26** is **AD0.3** (AIN3 in ADC)

Table 80. Pin function select register 1 (PINSEL1 - address 0x4002 C004) bit description

PINSEL1	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.16	GPIO Port 0.16	RXD1	SSEL0	SSEL	00
3:2	P0.17	GPIO Port 0.17	CTS1	MISO0	MISO	00
5:4	P0.18	GPIO Port 0.18	DCD1	MOSI0	MOSI	00
7:6	P0.19[1]	GPIO Port 0.19	DSR1	Reserved	SDA1	00
9:8	P0.20[1]	GPIO Port 0.20	DTR1	Reserved	SCL1	00
11:10	P0.21[1]	GPIO Port 0.21	RI1	Reserved	RD1	00
13:12	P0.22	GPIO Port 0.22	RTS1	Reserved	TD1	00
15:14	P0.23[1]	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0	00
17:16	P0.24[1]	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1	00
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3	00
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3	00
23:22	P0.27[1][2]	GPIO Port 0.27	SDA0	USB_SDA	Reserved	00
25:24	P0.28[1][2]	GPIO Port 0.28	SCL0	USB_SCL	Reserved	00


```
LPC_PINCON->PINSEL1|=(1<<20);
```

1.3. Cortex-M3: Exceptions

- What is an exception?
  - Any event (**internal or external**) that stops μP current process to switch to different task.
  - The exception priority is used to manage **when an it is attended** by the μP, and **to nest them**.
- In Cortex-M3
  - **System Exceptions** are numbered 1–15.
  - **External Interrupt** (to the μP, known as Interrupt Requests, IRQs) are numbered 16 and above.
  - Cortex-M3 chips can have different numbers of IRQs (from 1 to 240) and different numbers of priority levels according to them.
  - Cortex-M3 in LPC17XX has 35 vectored exceptions.
  - Most of the exceptions have programmable priority, and a few have fixed priority (normally 3, like in LPC17XX).

1.3. Cortex-M3: Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	-3 (Highest)	Reset
2	NMI	-2	Nonmaskable interrupt (external NMI input)
3	Hard Fault	-1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus Fault	Programmable	Bus error, like Prefetch abort
6	Usage Fault	Programmable	Exceptions due to program error or trying to access coprocessor
7-10	Reserved	N/A	[illegible]

1.3. Cortex-M3: Exceptions

13	Reserved	N/A	—
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System Tick Timer
16	**External Interrupt #0**	**Programmable**	**External Interrupt**
17	**External Interrupt #1**	**Programmable**	**External Interrupt**
…	**…**	**…**	**…**
255	**External Interrupt #239**	**Programmable**	**External Interrupt**

1.3. LPC17xx: Interrupt Number Definition

LPC17xx.h

```
typedef enum IRQn
{
/****** Cortex-M3 Processor Exceptions Numbers *******************
  Reset_IRQn                  = -15,      /*!< 1 Reset Vector, invo
  NonMaskableInt_IRQn         = -14,      /*!< 2 Non Maskable Inter
  HardFault_IRQn              = -13,      /*!< 3  Hard Fault, all c
  MemoryManagement_IRQn       = -12,      /*!< 4 Cortex-M3 Memory M
  BusFault_IRQn               = -11,      /*!< 5 Cortex-M3 Bus Faul
  UsageFault_IRQn             = -10,      /*!< 6 Cortex-M3 Usage Fa
  SVCall_IRQn                 = -5,       /*!< 11 Cortex-M3 SV Call
  DebugMonitor_IRQn           = -4,       /*!< 12 Cortex-M3 Debug M
  PendSV_IRQn                 = -2,       /*!< 14 Cortex-M3 Pend SV
  SysTick_IRQn                = -1,       /*!< 15 Cortex-M3 System

/****** LPC17xx Specific Interrupt Numbers ************************
  WDT_IRQn                    = 0,        /*!< Watchdog Timer Inter
  TIMER0_IRQn                 = 1,        /*!< Timer0 Interrupt
  TIMER1_IRQn                 = 2,        /*!< Timer1 Interrupt
  TIMER2_IRQn                 = 3,        /*!< Timer2 Interrupt
  TIMER3_IRQn                 = 4,        /*!< Timer3 Interrupt
  UART0_IRQn                  = 5,        /*!< UART0 Interrupt
  UART1_IRQn                  = 6,        /*!< UART1 Interrupt
  UART2_IRQn                  = 7,        /*!< UART2 Interrupt
  UART3_IRQn                  = 8,        /*!< UART3 Interrupt
  PWM1_IRQn                   = 9,        /*!< PWM1 Interrupt
  I2C0_IRQn                   = 10,       /*!< I2C0 Interrupt
  I2C1_IRQn                   = 11,       /*!< I2C1 Interrupt
  I2C2_IRQn                   = 12,       /*!< I2C2 Interrupt
  SPI_IRQn                    = 13,       /*!< SPI Interrupt
  SSP0_IRQn                   = 14,       /*!< SSP0 Interrupt
  SSP1_IRQn                   = 15,       /*!< SSP1 Interrupt                              */
  PLL0_IRQn                   = 16,       /*!< PLL0 Lock (Main PLL) Interrupt              */
  RTC_IRQn                    = 17,       /*!< Real Time Clock Interrupt                   */
  DMA_IRQn                    = 26,       /*!< General Purpose DMA Interrupt               */
```

Table 50. Connection of interrupt sources to the Vectored Interrupt Controller

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
0	16	0x40	WDT	Watchdog Interrupt (WDINT)
1	17	0x44	Timer 0	Match 0 - 1 (MR0, MR1) Capture 0 - 1 (CR0, CR1)
2	18	0x48	Timer 1	Match 0 - 2 (MR0, MR1, MR2) Capture 0 - 1 (CR0, CR1)
3	19	0x4C	Timer 2	Match 0-3 Capture 0-1
4	20	0x50	Timer 3	Match 0-3 Capture 0-1
5	21	0x54	UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
6	22	0x58	UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Control Change End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)

1.3. Cortex-M3: Exceptions (Vector Table)

Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
. . .		. . .	. . .
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
		0x0000	Initial SP value

- Vector Table contains **addresses** (vectors) of **exception handlers** and **ISRs**.
- **M**ain **S**tack **P**ointer initial value in location 0.
  - Set up by hardware during Reset.
- Vector Table can be relocated to SRAM.
  - Via the Vector Table Offset (Register contained in the NVIC)
- The **least-significant bit of each vector must be 1**, indicating that the exception handler is **Thumb**

1.3. LPC17xx: Vector Table definition

- **Startup_LPC17xx.s**

```
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors

__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     MemManage_Handler         ; MPU Fault Handler
                DCD     BusFault_Handler          ; Bus Fault Handler
                DCD     UsageFault_Handler        ; Usage Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     DebugMon_Handler          ; Debug Monitor Handler
                DCD     0                         ; Reserved
                DCD     PendSV_Handler            ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler

                ; External Interrupts
                DCD     WDT_IRQHandler            ; 16: Watchdog Timer
                DCD     TIMER0_IRQHandler         ; 17: Timer0
                DCD     TIMER1_IRQHandler         ; 18: Timer1
                DCD     TIMER2_IRQHandler         ; 19: Timer2
                DCD     TIMER3_IRQHandler         ; 20: Timer3
                DCD     UART0_IRQHandler          ; 21: UART0
                DCD     UART1_IRQHandler          ; 22: UART1
                DCD     UART2_IRQHandler          ; 23: UART2
                DCD     UART3_IRQHandler          ; 24: UART3
                DCD     PWM1_IRQHandler           ; 25: PWM1
                DCD     I2C0_IRQHandler           ; 26: I2C0
                DCD     I2C1_IRQHandler           ; 27: I2C1
                DCD     I2C2_IRQHandler           ; 28: I2C2
                DCD     EINT3_IRQHandler          ; 37: External Interrupt 3
```

1.3. LPC17xx: Vector Table definition

Example of program with exceptions

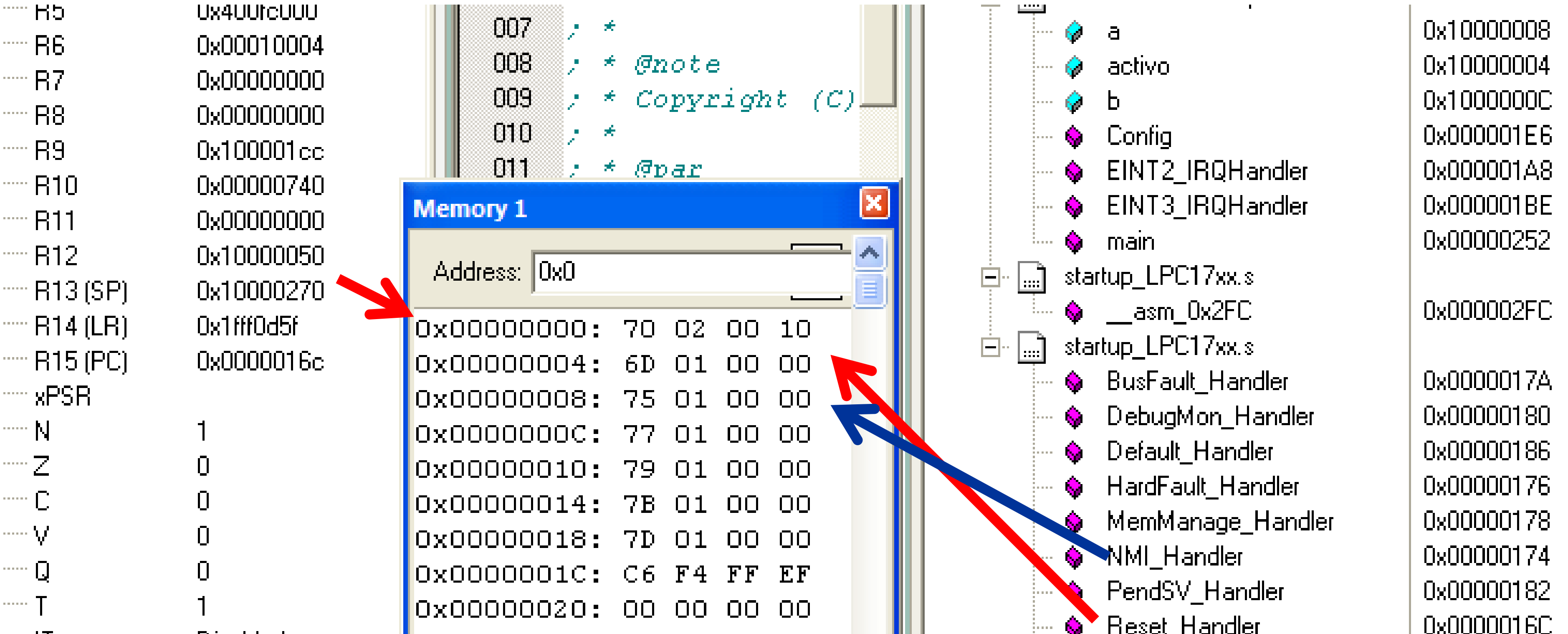
```
// Programa principal
int main(void)
{
  //Funcion de inicializacion
  Config();

  // Programa principal
  while (1) {

  }
}
```

```
void EINT2_IRQHandler(void)
{
  // Borrar el falg de la EINT3 --> EXTINT.3
  LPC_SC->EXTINT = 1 << 2;
  activo ++;
}

void EINT3_IRQHandler(void)
{
```

```
// Función de inicialización
void Config(void)
{
  // Configuracion del P2.13 como EINT3    --> PINSEL
  LPC_PINCON->PINSEL4 |= 1 << (12*2);

  // Configurar el pin P1.29 como salida --> GPIO1 F
  LPC_GPIO1->FIODIR |= 1<<29;

  // Interrupción activa por flanco de bajada    -->
  LPC_SC->EXTMODE |= 1<< 2;

  // Configuramos prioridad 1 a la interrupcion EINT
  NVIC->IP[EINT2_IRQn] = 0x01 << 3;
  NVIC->IP[EINT3_IRQn] = 0x02 << 3;

  // Habilitar la interrupcion EINT3  --> ISER0.21

  NVIC->ISER[0] = 1 << EINT2_IRQn; /* enable interru
  NVIC->ISER[0] = 1 << EINT3_IRQn; /* enable interru

  LPC_GPIOINT->IO0IntEnR |= 3<< 21; // P0.21 y  P0.2
  LPC_GPIOINT->IO0IntEnF |= 3<< 21;  //P0.21 y  P0.2

}
```

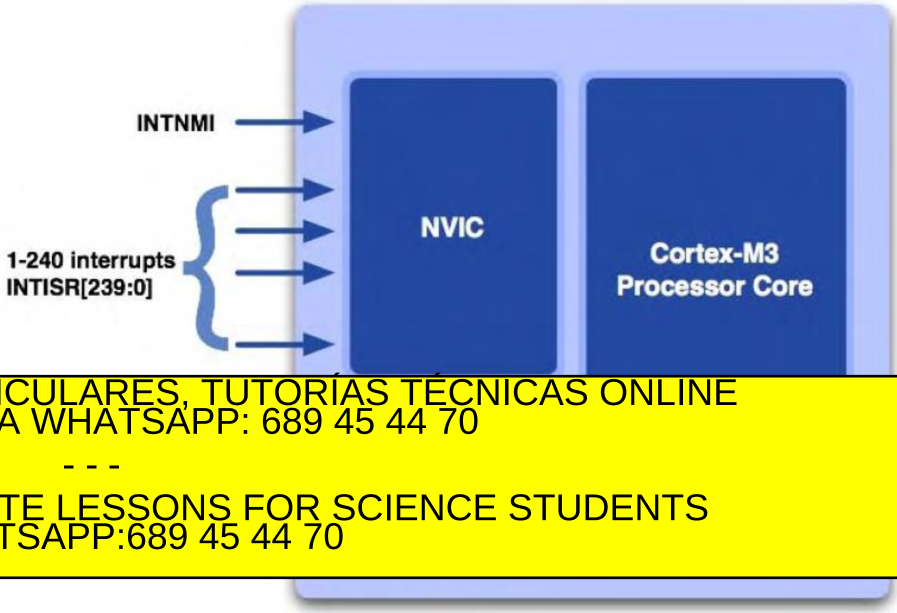
1.3. LPC17xx: Vector Table definition (keil example)

- Keil example vectors inicializ.

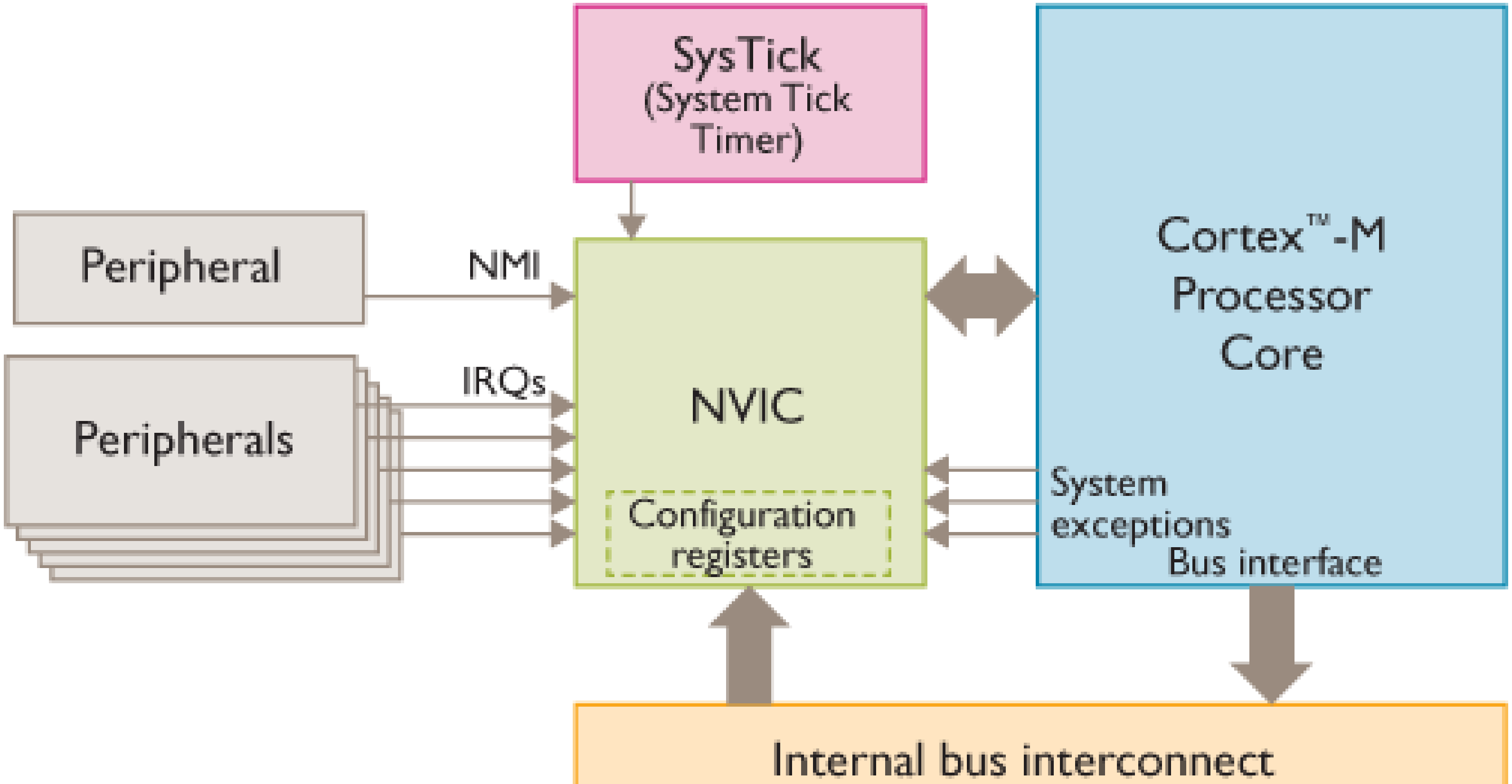


1.3. NVIC: Characteristics

- Nested Vector Interrupt Controller.
- NVIC is integrated into Cortex-M3 core.
- Supports 240 interrupt sources.
- 256 priority levels for each interrupt.
- NVIC hardware supports nested interrupts.
- Fast context switch.
- 12-cycle typical.
- Advanced features:
  - Priority pre-emption
  - Tail-chaining



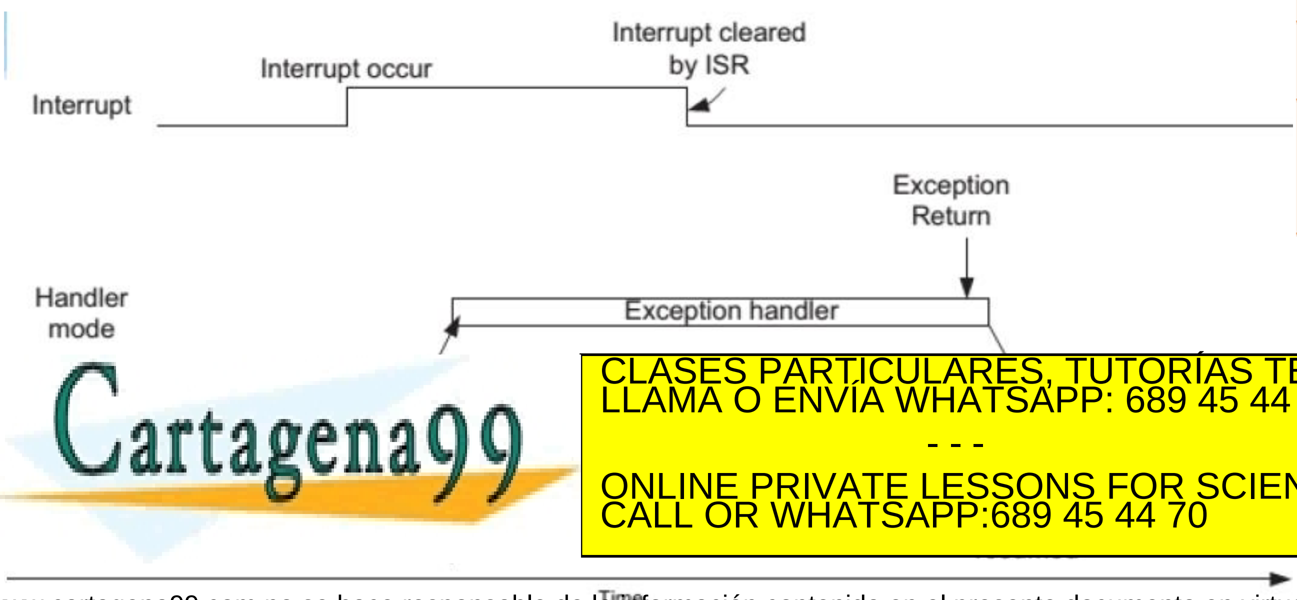
1.3. NVIC: Characteristics



1.3. NVIC: Interrupt sequence (enter + exit)

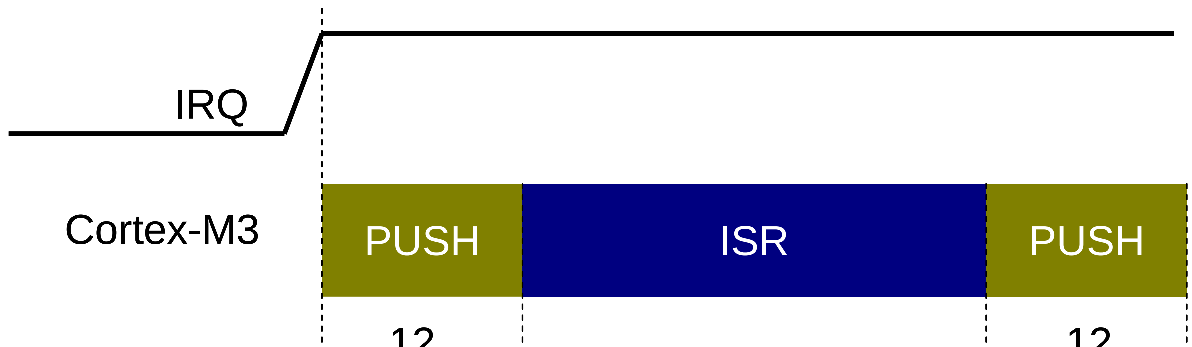
- Pre-emption & Exit
  - Stacking – Unstacking
    - PC, xPSR, r0-r3, r12, LR
  - Vector fetch.
  - Register update:
    - SP,PSR,PC,LR

Pre SP(N)	-
N-4	xPSR
N-8	PC
N-12	LR
N-16	R12
N-20	R3
N-24	R2
N-28	R1
New SP (N-32)	R0



1.3. NVIC: Interrupt Latency

- Deterministic interrupt latency
  - Cortex-M3 has an interrupt latency of 12 cycles and 12 cycles to return from servicing.
    - ARM7 does not have deterministic interrupt latency (24 to 42 cycles)
  - Latency includes stacking the registers, vector fetch, and fetching instructions for the interrupt handler.



1.3. NVIC: Exception States

- **Inactive:**
  - The exception is not active and not pending.
- **Pending:**
  - The exception is waiting to be serviced by the processor.
  - An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
- **Active:**
  - An exception that is being serviced by the processor but has not completed.
  - An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.
- **Active and pending**

1.3. NVIC: Registers (I)

- Each interrupt input has several registers to control it:
  - **Enable/Disable Bit (ISERn/ICERn)**
    - Enable or disable the interrupt
    - Can be set, cleared or read
  - **Pending Bit (ISPRn/ICPRn)**
    - If the pending bit is set, then the interrupt is pending
    - An interrupt can be "pended" by setting the pending bit
    - Pending bit can be set, cleared or read
  - **Active Bit (IABRn)**
    - A bit is set if the interrupt is executing or "active-stacked"
    - Active register is normally read only

1.3. NVIC: Registers (II)

Address	Name	Type	Description
0xE000E004	ICTR	RO	Interrupt Controller Type Register
0xE000E100 - 0xE000E10B	NVIC_ISER0 - NVIC_ISER2	RW	Interrupt Set-Enable Registers
0xE000E180 - 0xE000E18B	NVIC_ICER0 - NVIC_ICER2	RW	Interrupt Clear-Enable Registers
0xE000E200 - 0xE000E20B	NVIC_ISPR0 - NVIC_ISPR2	RW	Interrupt Set-Pending Registers
0xE000E280 - 0xE000E28B	NVIC_ICPR0 - NVIC_ICPR2	RW	Interrupt Clear-Pending Registers
0xE000E300 -	NVIC_IABR0 -	RO	Interrupt Active Bit Register

1.3. NVIC: Registers (III)

- To disable exceptions:
  - **PRIMASK**
    - A 1-bit register, when this is set, it allows nonmaskable interrupt (NMI) and the hard fault exception; all other interrupts are disabled.
  - **FAULTMASK**
    - A 1-bit register, when this is set, it allows only the NMI and all other interrupts and fault handling exceptions are disabled.
  - **BASEPRI**
    - A register of up to 8-bits (depending on the bit-width implemented for the priority level). It defines the masking

1.3. Cortex-M3: Exceptions (Priority Level)

- A higher-priority exception can preempt a lower-priority exception.
- **Reset**, **NMI**, and **hard fault** have fixed priority levels.
- Supports 256 levels of programmable priority.
- The reduction of priority levels can be implemented by **cutting out several lowest bits** of the priority configuration register:

  - **3 bits** of priority level

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented			Not implemented, read as zero				

  - **4 bits** of priority level

1.3. System Control Block (SCB)

- **Exception enables**.
- **Setting** or **clearing** exceptions to/from the **pending state**.
- **Exception status** (Inactive, Pending, or Active). Inactive is when an exception is neither Pending nor Active.
- **Priority setting** (for configurable system exceptions)
- The **exception number** of the **currently executing** code and highest pending exception.

1.3. Cortex-M3: Exceptions (Priority Level Register)

- This register is further divided into two parts: **preempt priority** and **subpriority**.
- Using a **Priority Group** register, the priority-level configuration register can be divided into two halves, i.e., the upper half (preempt priority) and the lower half (subpriority).
  - **Preempt priority**: an interrupt or exception with a **higher preempt priority can preempt one with a lower** preempt priority.
  - **Subpriority**: the order when **multiple interrupts or exceptions** with the **same preempt priority** occur at the same time.

Priority Group	Preempt Priority Field	Subpriority Field
0	Bit [7:1]	Bit [0]
1	Bit [7:2]	Bit [1:0]
2	Bit [7:3]	Bit [2:0]
3	Bit [7:4]	Bit [3:0]
4	Bit [7:5]	Bit [4:0]

1.3. NVIC: LPC17xx

- LPC17xx supports 35 vectored interrupts.
  - ARM allows manufacturers flexibility to implement fewer than 240
- 32 interrupt priority levels.
  - ARM allows flexibility to implement fewer than 256 levels
- Priority.
  - A programmable priority level of 0-31 for each interrupt.
  - A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
  - Grouping of priority values into group priority and sub-priority fields.
- Stack Operations.
  - The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead.
- An External Non-Maskable Interrupt (NMI).
- Includes Wake-up Interrupt Controller (WIC).

1.3. LPC17xx: Config. Interrupts priority

- Each priority register is divided into four eight bit priority fields, each field being assigned to an individual interrupt vector.
- LPC176x Interrupt Vectors starts at ISR #16 (vector offset 0x40)
- **Lower numbers are higher priority.**
- The LPC17xx only uses **5 bits** of this field to implement 32 levels of priority.
  - However, you should note that the active priority bits are in the upper 5 bits of each priority field.
- By default the priority field defines levels of priority with level zero the highest and 32 the lowest.
- Format the priority field into priority groups (pre-emption) and subgroups (sub-levels).

1.3. LPC17xx: Config. Interrupts priority

- Interrupt source has an **5-bits** interrupt priority value.
- The 5 bits are divided into **preempting priority** levels and **non-preempting** "sub-priority" levels.
  - Sub-priority levels only have an effect if the pre-empting priority levels are the same.
  - The software programmable **PRIGROUP** register field of the NVIC chooses how many of the 5-bits are used for "group-priority" and how many are used for "sub-priority".
  - Group priority is the pre-empting priority.

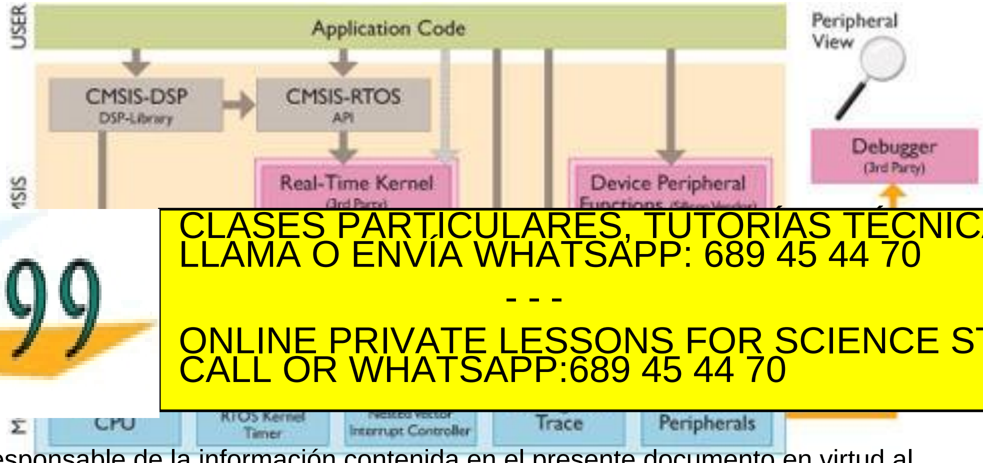
	Interrupt priority level value, PRI_N[7:0]			Number of	
PRIGROUP	Binary point[1]	Group priority bits	Subpriority bits	Group priorities	Subpriorities
b010	bxxxxx.000	[7:3]	none	32	1
b011	bxxxx.y000	[7:4]	[3]	16	2
b100	bxxx.yy000	[7:5]	[4:3]	8	4

- CMSIS is a vendor-independent hardware abstraction layer (HAL) for the Cortex-M processor series.

- CMSIS enables consistent and simple software interfaces to the processor and the peripherals, simplifying software re-use, reducing the learning curve for new microcontroller developers and reducing the time to market for new devices.

- Standardizing the software interfaces across all Cortex-M silicon vendor products.

- The CMSIS consists of the following components:
  - **CMSIS-CORE:** provides an interface to Cortex-M0, Cortex-M3, Cortex-M4, SC000, and SC300 processors and peripheral registers.
  - **CMSIS-DSP:** DSP library with over 60 functions in fixed-point (fractional q7, q15, q31) and single precision floating-point (32-bit) implementation.
  - **CMSIS-RTOS API:** standardized programming interface for real-time operating systems for thread control, resource, and time management.
  - **CMSIS-SVD:** System View Description XML files that contain the programmer's view of a complete microcontroller system including peripherals.



1.3. CMSIS: Cortex Microcontroller Software Interface Standard

- Hardware Abstraction Layer (HAL) for Cortex-M processor registers:
  - **NVIC, MPU**
- Standardized system exception names. For example:
  - **void SVC_Handler()**
  - **void UART0_IRQHandler()**
- Standardized method of header file organization.
- Common method for system initialization.
  - **SystemInit()**
- Standardized intrinsic functions. For example:
  - **void __disable_irq(void), void __enable_irq(void)**
- Common access functions for communication.
  - **SystemCoreClock**, in Keil !!!!!!

1.3. CMSIS: Accessing Core Registers

Function Definition	Core Register	Description
void __enable_irq (void)	PRIMASK = 0	Global Interrupt enable (using the instruction **CPSIE i**)
void __disable_irq (void)	PRIMASK = 1	Global Interrupt disable (using the instruction **CPSID i**)
void __set_PRIMASK (uint32_t value)	PRIMASK = value	Assign value to Priority Mask Register (using the instruction **MSR**)
uint32_t __get_PRIMASK (void)	return PRIMASK	Return Priority Mask Register (using the instruction **MRS**)
void __set_CONTROL (uint32_t value)	CONTROL = value	Set CONTROL register value (using the instruction **MSR**)
uint32_t __get_CONTROL (void)	return CONTROL	Return Control Register Value (using the instruction **MRS**)
void __set_PSP (uint32_t TopOfProcStack)	PSP = TopOfProcStack	Set Process Stack Pointer value (using the instruction **MSR**)
uint32_t __get_PSP (void)	return PSP	Return Process Stack Pointer (using the instruction **MRS**)

1.3. CMSIS: NVIC functions

- In addition, the CMSIS provides a number of functions for NVIC control:

Only Ext. Interrupt. ⇒

Function Name	Parameter	Description
void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)	Priority Grouping Value	Set the Priority Grouping (Groups . Subgroups)
void NVIC_EnableIRQ (IRQn_Type IRQn)	IRQ Number	Enable IRQn
void NVIC_DisableIRQ (IRQn_Type IRQn)	IRQ Number	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)	IRQ Number	Return 1 if IRQn is pending else 0
void NVIC_SetPendingIRQ (IRQn_Type IRQn)	IRQ Number	Set IRQn Pending
void NVIC_ClearPendingIRQ (IRQn_Type IRQn)	IRQ Number	Clear IRQn Pending Status
void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)	IRQ Number, Priority	Set Priority for IRQn
uint32_t NVIC_GetPriority (IRQn_Type IRQn)	IRQ Number	Get Priority for IRQn
uint32_t NVIC_EncodePriority (uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority)	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)	IRQ Number, Priority, pointer to Priority Group, pointer to Preemptive Priority, pointer to Sub Priority	Decode given priority to group, preemptive and sub priority
void NVIC_SystemReset (void)	(void)	Resets the System

1.3. NVIC access: CMSIS examples (I)

```
#include "LPC17xx.h"

uint32_t priorityGroup;                              /* Variables to store priority group and priority */
uint32_t priority;
uint32_t preemptPriority;
uint32_t subPriority;


int main (void)  {

  NVIC_SetPriorityGrouping(5);                       /* Set priority group to 5:
                                                        Bit[7..6] preempt priority Bits,
                                                        Bit[5..3] subpriority Bits
                                                        (valid for five priority bits) */

  priorityGroup =  NVIC_GetPriorityGrouping();       /* Get used priority grouping */

  priority = NVIC_EncodePriority(priorityGroup, 1, 6);   /* Encode priority with 6 for subpriority and 1 for preempt priority
                                                        Note: priority depends on the used priority grouping */

  NVIC_SetPriority(UART0_IRQn, priority);            /* Set new priority */

  priority =  NVIC_GetPriority(UART0_IRQn);          /* Retrieve priority again */

  NVIC_DecodePriority(priority, priorityGroup, &preemptPriority, &subPriority);
```

1.3. NVIC access: CMSIS examples (II)

- CMSIS NVIC functions NVIC_EnableIRQ(), NVIC_GetActive()

```
#include "LPC17xx.h"

uint32_t active;                                         /* Variable to store interrupt active state */


void TIMER0_IRQHandler(void)  {                          /* Timer 0 interrupt handler  */

  if (LPC_TIM0->IR & (1 << 0))  {                        /* Check if interrupt for match channel 0 occured */
    LPC_TIM0->IR |= (1 << 0);                            /* Acknowledge interrupt for match channel 0 occured */
  }
  active = NVIC_GetActive(TIMER0_IRQn);                  /* Get interrupt active state of timer 0 */
}


int main (void) {
                                                         /* Set match channel register MR0 to 1 millisecond */
  LPC_TIM0->MR0 = (((SystemCoreClock / 1000) / 4) - 1);  /* 1 ms? */

  LPC_TIM0->MCR = (3 << 0);                              /* Enable interrupt and reset for match channel MR0 */

  NVIC_EnableIRQ(TIMER0_IRQn);                           /* Enable NVIC interrupt for timer 0 */
```

1.3. CMSIS: Peripheral Access

- Describes naming conventions, requirements, and optional features for accessing peripherals.
- Each peripheral provides a data type definition with a name that is composed of a prefix **<device abbreviation>_** and the **<peripheral name>_**
  - for example, **LPC_UART** for the device **LPC** and the peripheral **UART**.
  - The intention is to avoid name collisions caused by short names.
  - If more peripherals exist of the same type, identifiers have a postfix consisting of a digit or letter, for example **LPC_UART0**, **LPC_UART1**.
- The data type definition uses the standard C data types from the ANSI C header file <stdint.h>.
  - IO Type Qualifiers are used to specify the access to peripheral variables.
  - IO Type Qualifiers are indented to be used for automatic generation of debug information of peripheral registers and are defined as shown below:

1.3. CMSIS: Peripheral Access (example)

- The following *typedef* is an example for a UART.
  - **<device abbreviation>_UART_TypeDef**: defines the generic register layout for all UART channels in a device.

```
typedef struct
{
  union {
  __I  uint8_t  RBR;                /* Offset: 0x000 (R/ )  Receiver Buffer Register        */
  __O  uint8_t  THR;                /* Offset: 0x000 ( /W)  Transmit Holding Register       */
  __IO uint8_t  DLL;                /* Offset: 0x000 (R/W)  Divisor Latch LSB               */
       uint32_t RESERVED0;
  };
  union {
  __IO uint8_t  DLM;                /* Offset: 0x004 (R/W)  Divisor Latch MSB               */
  __IO uint32_t IER;                /* Offset: 0x004 (R/W)  Interrupt Enable Register       */
  };
  union {
  __I  uint32_t IIR;                /* Offset: 0x008 (R/ )  Interrupt ID Register           */
  __O  uint8_t  FCR;                /* Offset: 0x008 ( /W)  FIFO Control Register           */
  };
  __IO uint8_t  LCR;                /* Offset: 0x00C (R/W)  Line Control Register           */
       uint8_t  RESERVED1[7];
  __I  uint8_t  LSR;                /* Offset: 0x014 (R/ )  Line Status Register            */
       uint8_t  RESERVED2[7];
  __IO uint8_t  SCR;                /* Offset: 0x01C (R/W)  Scratch Pad Register            */
       uint8_t  RESERVED3[3];
  __IO uint32_t ACR;                /* Offset: 0x020 (R/W)  Autobaud Control Register       */

} LPC_UART_TypeDef;
```

1.3. CMSIS: Peripheral Access (ex.: cont.)

- To access the registers of the UART defined above, pointers to a register structure are defined.

- In this example **<device abbreviation>_UART#** are two pointers to UARTs defined with above register structure:
  - #define LPC_UART2 ((LPC_UART_TypeDef *) LPC_UART2_BASE)
  - #define LPC_UART3 ((LPC_UART_TypeDef *) LPC_UART3_BASE)

- The registers in the various UARTs can now be referred in the user code as shown below:
  - **LPC_UART1->DR** // is the data register of UART1.

1.3. CMSIS: Peripheral access (min. requirements)

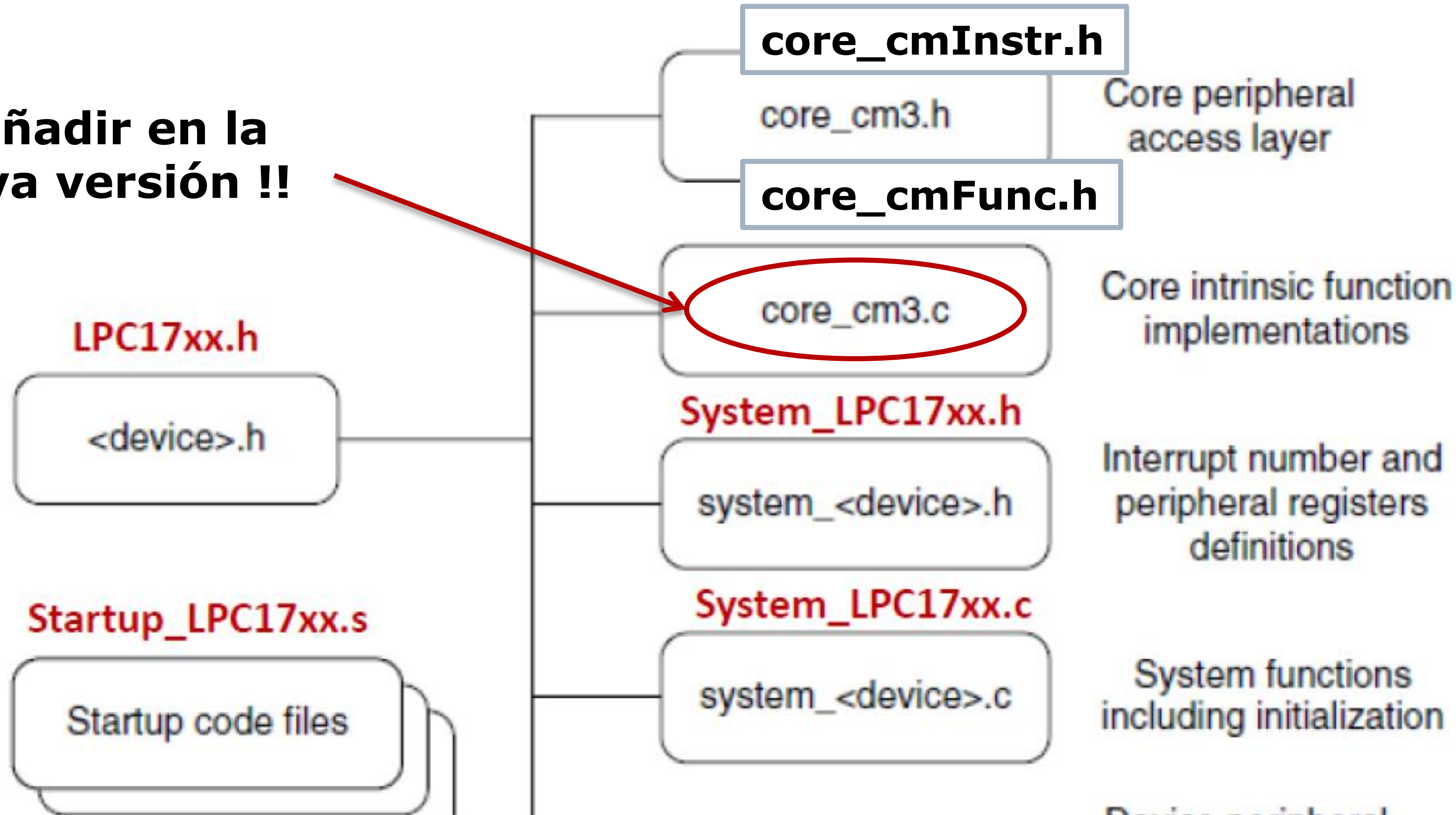
- To access the peripheral registers and related function in a device, the files **device.h** and **core_cm3.h** define as a minimum:
  - The **Register Layout Typedef** for each peripheral that defines all register names. **RESERVED** is used to introduce space into the structure for adjusting the addresses of the peripheral registers.

```
typedef struct
{
  __IO uint32_t CTRL;                    /* Offset: 0x000 (R/W)  SysTick Control and Status Register */
  __IO uint32_t LOAD;                    /* Offset: 0x004 (R/W)  SysTick Reload Value Register       */
  __IO uint32_t VAL;                     /* Offset: 0x008 (R/W)  SysTick Current Value Register      */
  __I  uint32_t CALIB;                   /* Offset: 0x00C (R/ )  SysTick Calibration Register        */
} SysTick_Type;
```

  - **Base Address** for each peripheral.
    - #define SysTick_BASE (SCS_BASE + 0x0010) /* SysTick Base Address */
  - **Access Definitions** for each peripheral. In case of multiple peripherals that are using the same **register layout typdef**, multiple access definitions exist (LPC_UART0, LPC_UART2).
    - #define SysTick ((SysTick_Type *) Systick_BASE) /* SysTick access definition */

1.3. CMSIS: Files for the LPC17xx (NXP)

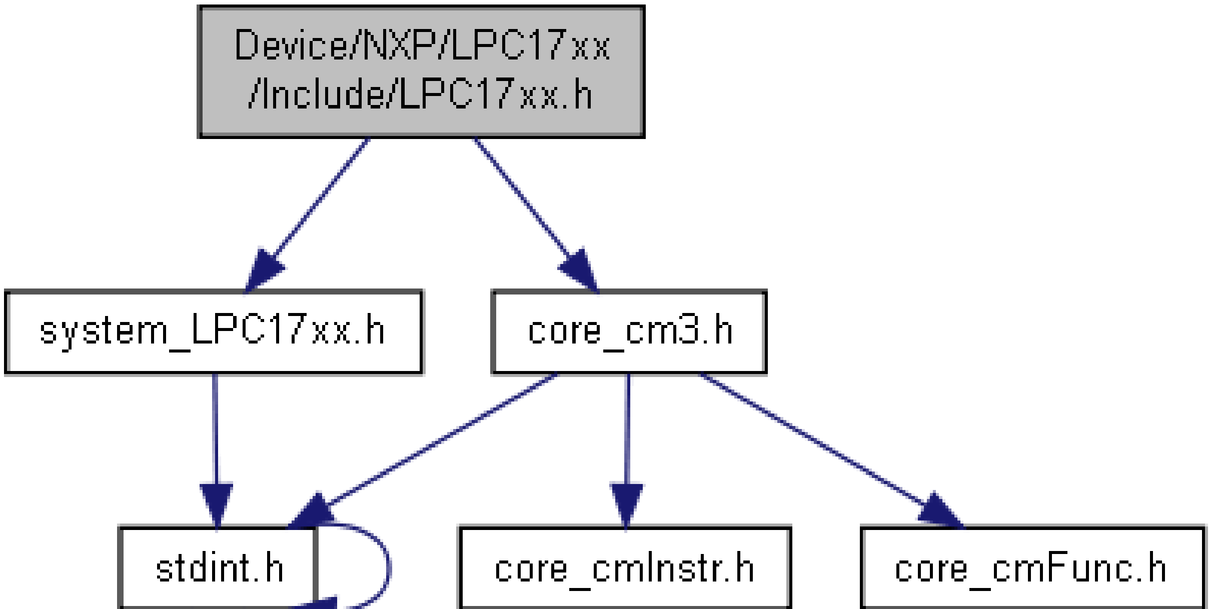
No añadir en la nueva versión !!



1.3. CMSIS: Files for LPC17xx (NXP)

```
#include "core_cm3.h"
#include "system_LPC17xx.h"
```

Include dependency graph for LPC17xx.h:



1.3. CMSIS: core_cm3.h file

- As the LPC17xx is CortexM3 based, the core files are "**core_cm3.h**" and "core_cm3.c".
- The files "**core_cm3.h**" and "core_cm3.c" are standard across all vendor devices that have the CortexM3 at their core.
- A fragment of "**core_cm3.h**" is shown in the figure:

```
/** \brief  Set Interrupt Priority

    The function sets the priority of an interrupt.

    \note The priority cannot be set for every core interrupt.

    \param [in]      IRQn  Interrupt number.
    \param [in]  priority  Priority to set.
 */
__STATIC_INLINE void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
{
```

5 bits (LPC17xx)

1.3. CMSIS: startup_LPC17xx.s

- The key variations in **startup_LPC17xx.s** for the different compilers:

	IVT definition	default ISR handlers	Import	Export
arm	DCD SysTick_Handler	SysTick_Handler PROC EXPORT SysTick_Handler [WEAK] B . ENDP	IMPORT SystemInit	EXPORT __Vectors
iar	DCD SysTick_Handler	PUBWEAK SysTick_Handler SECTION .text:CODE:REORDER(1) SysTick_Handler B SysTick_Handler	EXTERN SystemInit	PUBLIC __vector_table
gcc	.long SysTick_Handler	.weak SysTick_Handler .type SysTick_Handler, %function SysTick_Handler: B . .size SysTick_Handler, . - SysTick_Handler	EXTERN SystemInit	.globl __cs3_interrupt_vector_cortex_m

Also in `startup_LPC17xx.s` we have the Reset Handlers. Shown below are the examples for ARM:

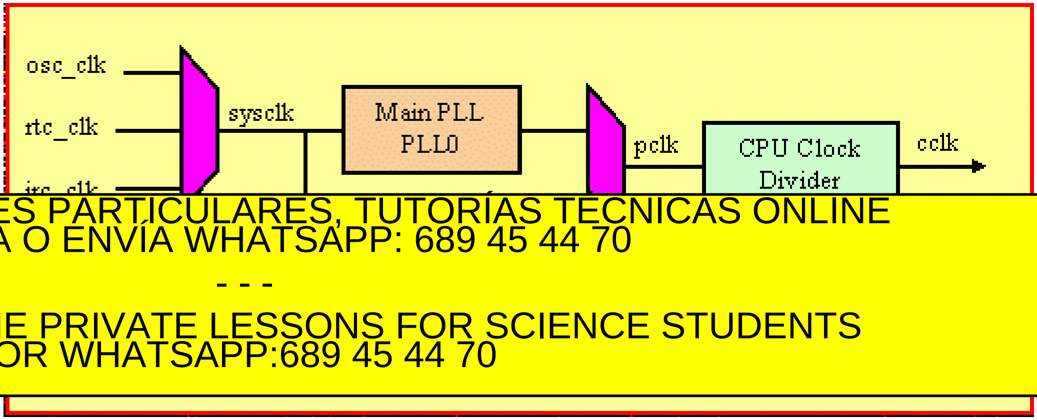
```
Reset_Handler    PROC
                 EXPORT  Reset_Handler            [WEAK]
                 IMPORT  SystemInit
                 ENDP
```

1.4. Clocking Features (I)

- Operating Frequency - 100 MHz
- The LPC1700 includes three independent oscillators (same as LPC236x).
  - Main Oscillator
  - Internal RC oscillator (Default after **Reset**)
  - RTC oscillator
- Any of the three clock sources can be chosen by software to drive the main PLL and ultimately the CPU.

1.4. Clocking Features (II)

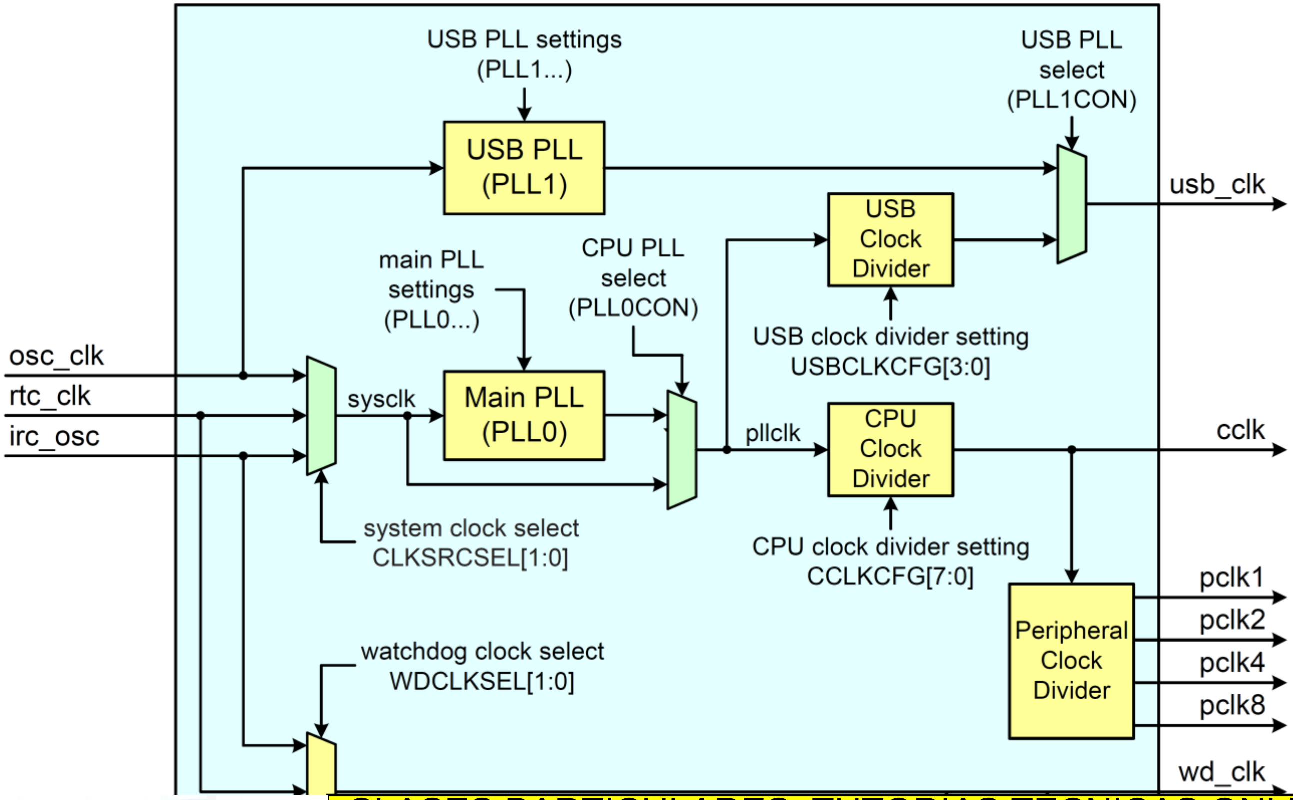
- **Internal RC oscillator**
  - Clock source for the WDT, and/or as the clock that drives the PLL and subsequently the CPU.
  - The nominal IRC frequency is 4 MHz ± 1% accuracy over the entire temp and voltage range.
- **Main oscillator**
  - Clock source for the CPU, with or without using the PLL.
  - The main oscillator also provides the clock source for USB PLL.
  - Operates at frequencies of 1 MHz to 24 MHz.
- **RTC oscillator**
  - Clock source for the RTC block, the main PLL, and subsequently the CPU.
  - 1 Hz clock to RTC.



1.4. Clocking Features (III)

- Main PLL (PPL0)
  - Input clock frequency in the range of 32 kHz to 50 MHz.
  - May run from the main oscillator, the internal RC oscillator, or the RTC oscillator.
- Second PLL (PLL1)
  - Dedicated to provide clocking for the USB interface to allow added flexibility for the main PLL settings.
- Peripheral Clock Selection Register(s)
  - Used to control the rate of the clock signal that will be supplied to the individual peripheral(s).
  - Each Peripheral can have its own clock setting where it can be individually set equal to CPU clock or divided down.
- Clock output function

1.4. Clocking Features (III)



1.4. System Clock: PLL0 Config. registers

Registros de configuración del PLL0

Name	Description	Access	Reset value	address
PLL0CON	PLL0 Control Register. Holding register for updating PLL0 control bits. Values written to this register do not take effect until a valid PLL0 feed sequence has taken place.	R/W	0	0x400FC080
PLL0CFG	PLL0 Configuration Register. Holding register for updating PLL0 configuration values. Values written to this register do not take effect until a valid PLL0 feed sequence has taken place.	R/W	0	0x400FC084
PLL0STAT	PLL0 Status Register. Read-back register for PLL0 control and configuration information. If PLL0CON or PLL0CFG have been written to, but a PLL0 feed sequence has not yet occurred, they will not reflect the current PLL0 state. Reading this register provides the actual values controlling the PLL0, as well as the PLL0 status.	RO	0	0x400FC088
PLL0FEED	PLL0 Feed Register. This register enables loading of the PLL0 control and configuration information from the PLL0CON and PLL0CFG registers into the shadow registers that actually affect PLL0 operation.	WO	NA	0x400FC08C

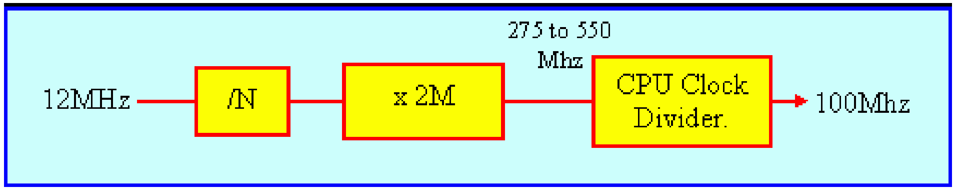
1.4. System Clock: PLL0 Config. registers

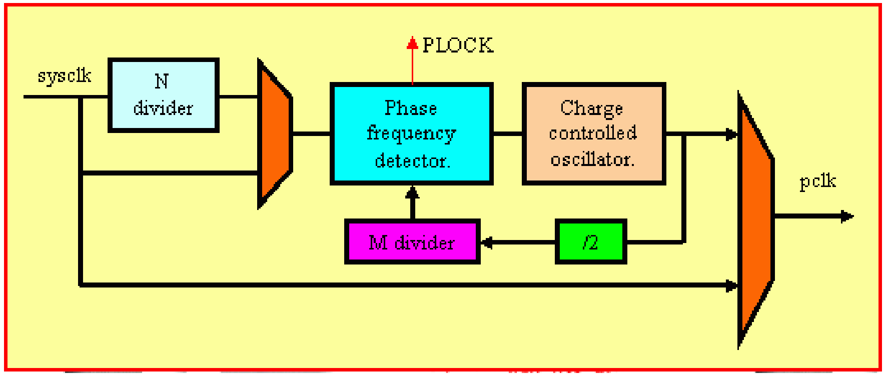
- **CCLKCFG:** Registro de configuración del reloj de la CPU

Bit	Symbol	Value	Description	Reset Value
7:0	CCLKSEL		Selects the divide value for creating the CPU clock (CCLK) from the PLL0 output.	0x00
		0 to 1	Not allowed, the CPU clock will always be greater than 100 MHz.	
		2	PLL0 output is divided by 3 to produce the CPU clock.	
		3	PLL0 output is divided by 4 to produce the CPU clock.	
		4	PLL0 output is divided by 5 to produce the CPU clock.	
		..	..	
		255	PLL0 output is divided by 256 to produce the CPU clock.	
31:8	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

1.4. System Clock: PLL0 Config. registers

- **PLL0CFG:** Registro de configuración del PLL0





Bit	Simbol	Description	Reset value
14:0	MSEL0	PLL0 Multiplier value. Supplies the value “M” in PLL0 frequency calculations. **The value stored here is M - 1**. Supported values for M are 6 through 512. **Note:** Not all values of M are needed, and therefore some are not supported by hardware.	0
15	-	Reserved, user software should not write ones to reserved bits. Thevalue read from a reserved bit is not defined.	NA

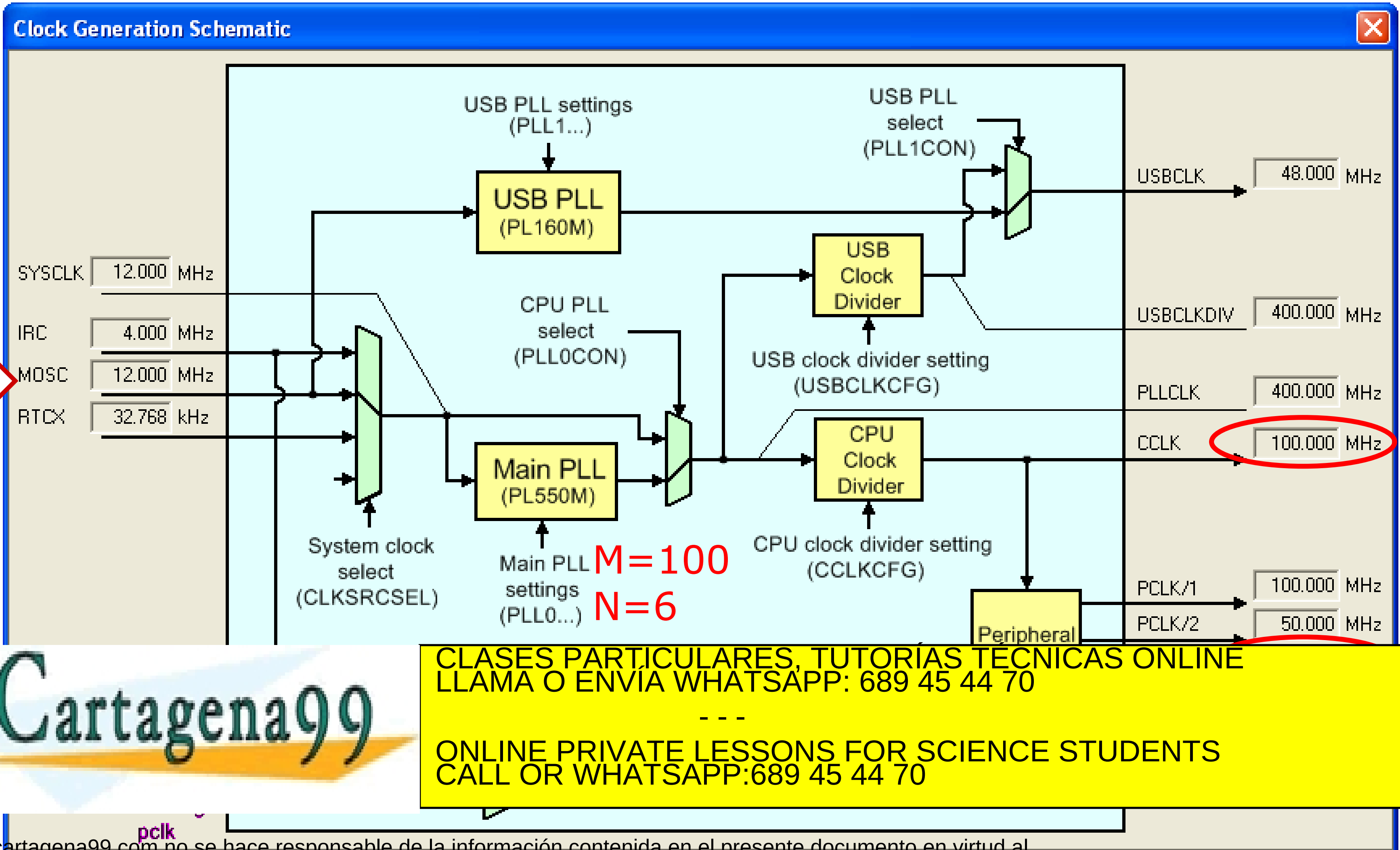
1.4. System Clock: Keil Config.

- system_LPC17xx.c

```
/*----------------------------------------------------------------------------
  Define clocks
 *----------------------------------------------------------------------------*/
#define XTAL        (12000000UL)        /* Oscillator frequency               */
#define OSC_CLK     (      XTAL)        /* Main oscillator frequency          */
#define RTC_CLK     (   32768UL)        /* RTC oscillator frequency           */
#define IRC_OSC     ( 4000000UL)        /* Internal RC oscillator frequency   */

/* F_cco0 = (2 * M * F_in) / N  */
#define __M               (((PLLOCFG_Val      ) & 0x7FFF) + 1)
#define __N               (((PLLOCFG_Val >> 16) & 0x00FF) + 1)
#define __FCCO(__F_IN)    ((2ULL * __M * __F_IN) / __N)
#define __CCLK_DIV        (((CCLKCFG_Val      ) & 0x00FF) + 1)

/* Determine core clock frequency according to settings */
 #if (PLL0_SETUP)
    #if   ((CLKSRCSEL_Val & 0x03) == 1)
        #define __CORE_CLK (__FCCO(OSC_CLK) / __CCLK_DIV)
    #elif ((CLKSRCSEL_Val & 0x03) == 2)
        #define __CORE_CLK (__FCCO(RTC_CLK) / __CCLK_DIV)
    #else
        #define __CORE_CLK (__FCCO(IRC_OSC) / __CCLK_DIV)
    #endif
 #else
 ...
 #endif
```

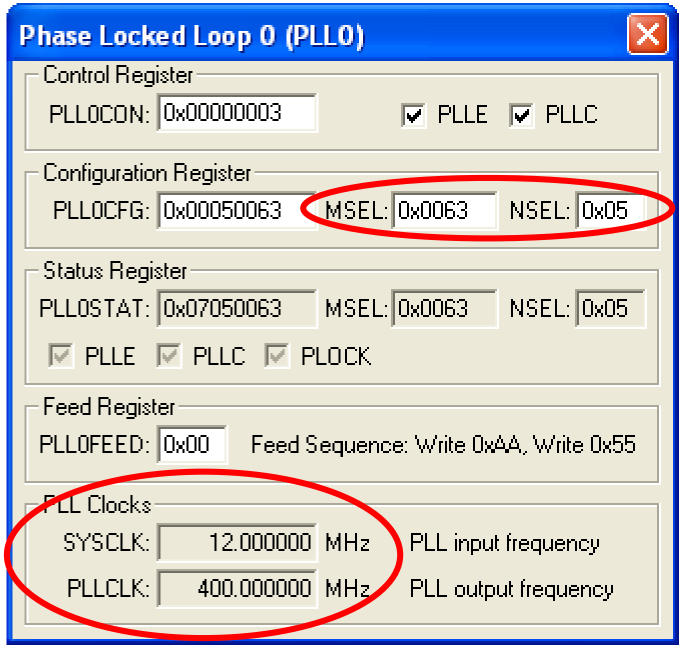
```
uint32_t SystemCoreClock = __CORE_CLK; /*!< System Clock Frequency (Core Clock)*/
```

- Peripherals->Clocking & Power Control

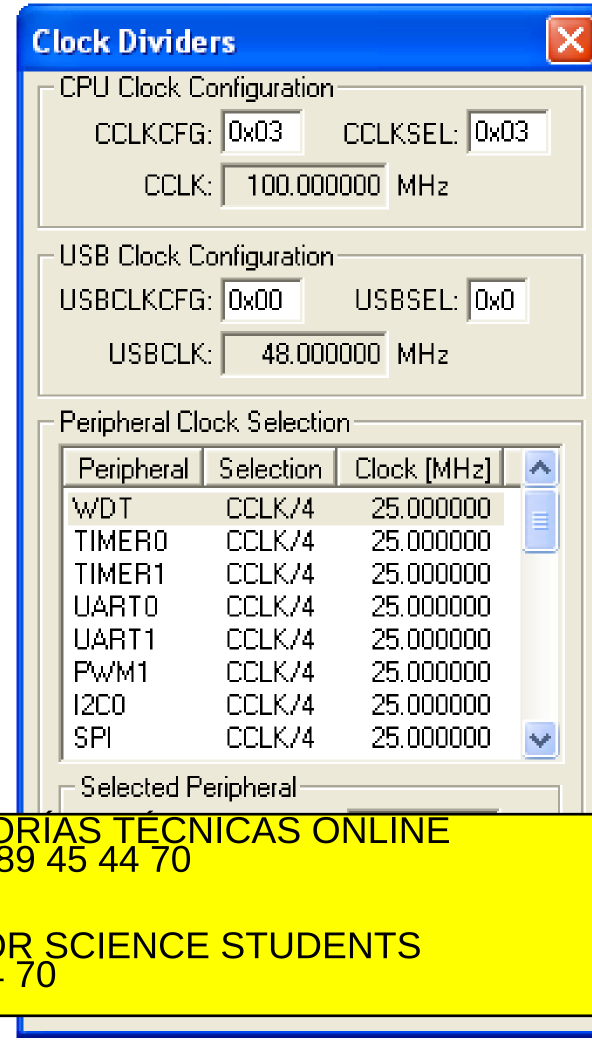


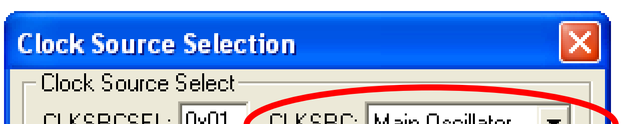
1.4. System Clock: Keil debug windows

- Peripherals->Clocking & Power Control



M=100
N=6





1.4. System Clock: system_LPC17xx.c file

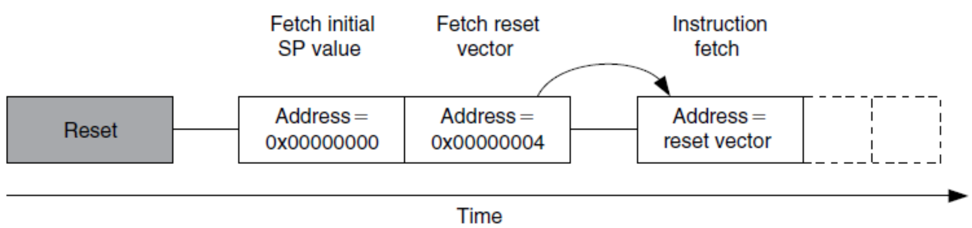
system_LPC17xx.c

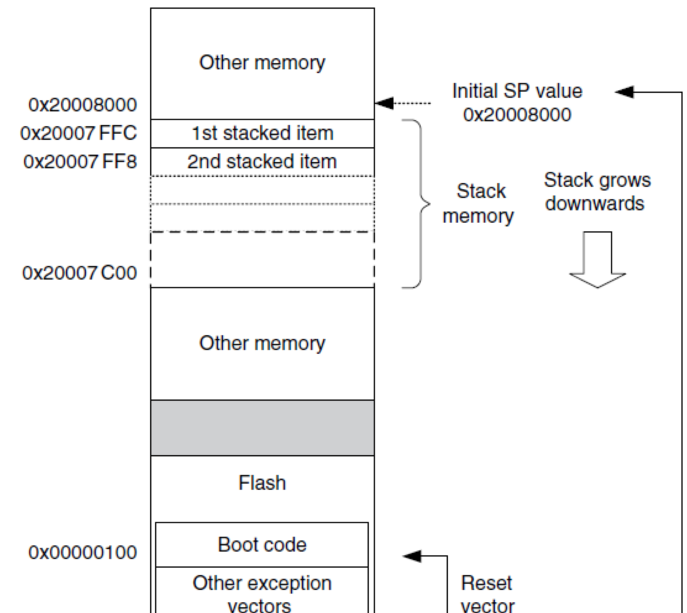
```
/*----------------------------------------------------------------------------
  Define clocks
 *----------------------------------------------------------------------------*/
#define XTAL        (12000000UL)        /* Oscillator frequency               */
#define OSC_CLK     (      XTAL)        /* Main oscillator frequency          */
#define RTC_CLK     (   32768UL)        /* RTC oscillator frequency           */
#define IRC_OSC     ( 4000000UL)        /* Internal RC oscillator frequency   */

/* F_cco0 = (2 * M * F_in) / N  */
#define __M               (((PLL0CFG_Val      ) & 0x7FFF) + 1)
#define __N               (((PLL0CFG_Val >> 16) & 0x00FF) + 1)
#define __FCCO(__F_IN)    ((2ULL * __M * __F_IN) / __N)
#define __CCLK_DIV        (((CCLKCFG_Val      ) & 0x00FF) + 1)

/* Determine core clock frequency according to settings */
 #if (PLL0_SETUP)
    #if   ((CLKSRCSEL_Val & 0x03) == 1)
        #define __CORE_CLK (__FCCO(OSC_CLK) / __CCLK_DIV)
    #elif ((CLKSRCSEL_Val & 0x03) == 2)
        #define __CORE_CLK (__FCCO(RTC_CLK) / __CCLK_DIV)
    #else
        #define __CORE_CLK (__FCCO(IRC_OSC) / __CCLK_DIV)
    #endif
 #else
 ...
 #endif
```

uint32_t **SystemCoreClock** = __CORE_CLK; /*!< System Clock Frequency (Core Clock)*/

1.4. Reset (I)

- **The most urgent exception in any µP.**
  - Reset Sequence: µP will read 2 words from the Vector Table:
    - Address **0x00000000**: Starting value of R13 (the MSP)
    - Address **0x00000004**: Reset Vector (VN 1), starting value of R15 (the PC)





- **After the Reset Vector is fetched:**
  - Cortex-M3 begins normal operations starting the program from the Reset Vector (VN 1)
  - It is necessary to have the SP initialized from that moment, as some of the exceptions

1.4. Reset (II)

- Reset Types and Signals.

Reset Type	Reset Signal on the Cortex-M3 Processor	Description
Power on reset	PORESETn	Reset that should be asserted when the device is powered up; resets processor core, peripherals, and debugging system Activate by power up sequence of the device
System reset	SYSRESETn	System reset; affects the whole system including processor core, NVIC (except debug control registers), MPU, peripherals but not the debugging system; activate by power up sequence of the device, reset request from debugger through NVIC register “AIRCR”
Processor reset	VECTRESET bit in the NVIC AIRCR register	Reset processor core only; affect the processor system including processor core, NVIC (except debug control registers), MPU, but not the debugging system; activate reset request from debugger through NVIC register “AIRCR”—intended to be used by debugger

1.4. Fault Exceptions

- Can be used during software development to determine the causes of errors in the program and correct them.
- What to do with them in final running systems? Fault-Handling methods:
  - **Recovery:** In some cases, it might be possible to resolve the problem that caused the fault. For example, in the case of coprocessor instructions, the problem can be resolved using coprocessor emulation software.
  - **Task termination:** In systems that run an OS, the offending applications could be terminated and restarted if needed.
  - **Reset:** In some other cases, the system might need a reset, with SYSRESETREQ or VECTRESET (depending on the part of the system wanted to reset)
- **Types of Fault Exceptions** (ordered by VN, from 3 to 6):

usage

1.4. Hard Fault Exception

- The hard fault handler can be caused by:
  - Usage faults, bus faults, and memory management faults **if their handler cannot be executed**.
  - A bus fault during vector fetch.

- **Hard Fault Status Register** (HFSR) is used in the exception handler to determine the cause of the fault.

1.4. Bus Fault Exception

- Bus faults are produced when an error response is received during a transfer on the AHB interfaces.
- Bus fault due to:
  - 1. Instruction prefetch abort.
  - 2. Data read/write abort.
  - 3. Stack PUSH in the beginning of interrupt processing.
  - 4. Stack POP at the end of interrupt processing.
  - 5. Reading of an interrupt vector address when the processor starts the interrupt-handling sequence.
- **Bus Fault Status Register** (BFSR) is used in the

1.4. Memory Management Fault Exception

- Common memory manage faults include:
  - 1. Access to memory regions not defined in MPU setup.
  - 2. Execute code from nonexecutable memory regions.
  - 3. Writing to read-only regions.
  - 4. An access in the user state to a region defined as privileged access only.

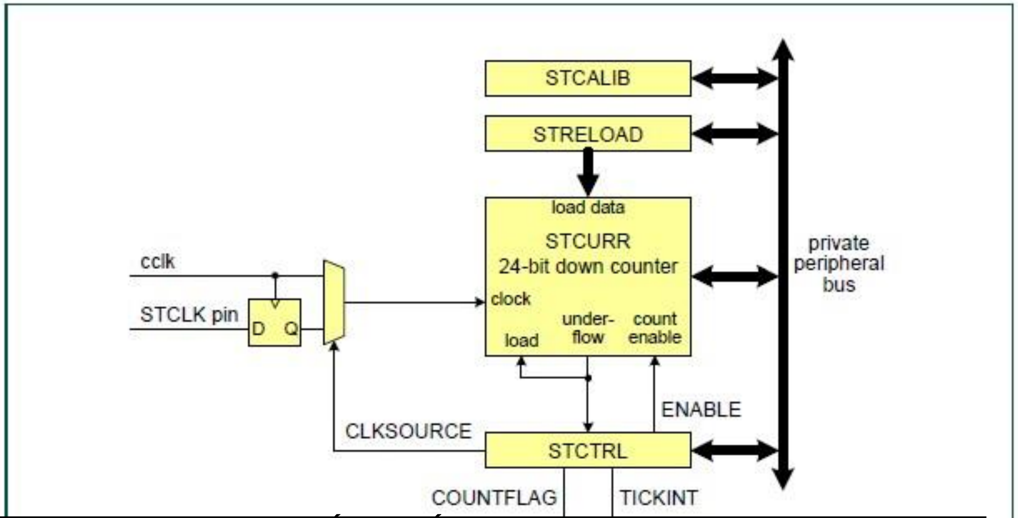
- **Memory Management Fault Status Register** (MFSR) is used in the exception handler to determine the cause of the fault.

1.4. Usage Fault Exception

- Usage faults can be caused by:
  - 1. Undefined instructions.
  - 2. Coprocessor instructions.
  - 3. Trying to switch to the ARM state (This can happen if you load a new value to PC with the LSB equal to 0).
  - 4. Invalid interrupt return (Link Register contains invalid/incorrect values)
  - 5. Unaligned memory accesses using multiple load or store instructions.
  - 6. Divide by zero.
- **Usage Fault Status Register** (UFSR) is used in

1.4. SYSTICK Timer

- SYSTICK timer is a **24-bit** down counter. The counter loads the reload value from the **RELOAD** register when it reach zero.
- It always run until the enable bit in the **SYSTICK Control** and **Status** register is cleared.
- **2** configurable Clock sources.
- Suitable for Real Time OS or other scheduled tasks.



1.4. SYSTICK configuration (I)

- **Initialization (3 steps)**
  - Step1: Specify **the RELOAD value**.
  - Step2: **Clear the counter** via NVIC_CURRENT
  - Step3: Set **CLK_SRC=1,** interrupt action (**INTEN),** and enable counter **(ENABLE)** via NVIC_CTRL.

Address	31-24	23-17	16	15-3	2	1	0	Name
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

- **CMSIS** function:
  - SysTick_Config(uint32_t ticks)

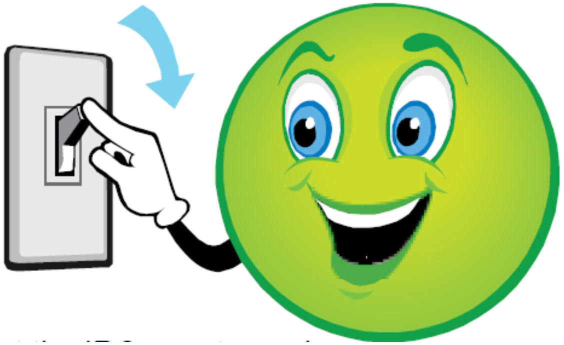
1.4. SYSTICK configuration (II)

core_cm3.h:

```
__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if ((ticks - 1) > SysTick_LOAD_RELOAD_Msk)  return (1);      /* Reload value impossible */

  SysTick->LOAD  = ticks - 1;                                  /* set reload register */
  NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) - 1);  /* set Priority for Systick Interrupt */
  SysTick->VAL   = 0;                                          /* Load the SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
                   SysTick_CTRL_ENABLE_Msk;                    /* Enable SysTick IRQ and SysTick Timer */
  return (0);                                                  /* Function successful */
}
```

```
#define Ftick 100          // SYSTICK Frequency interrupt
```

1.4. Power Modes (I)

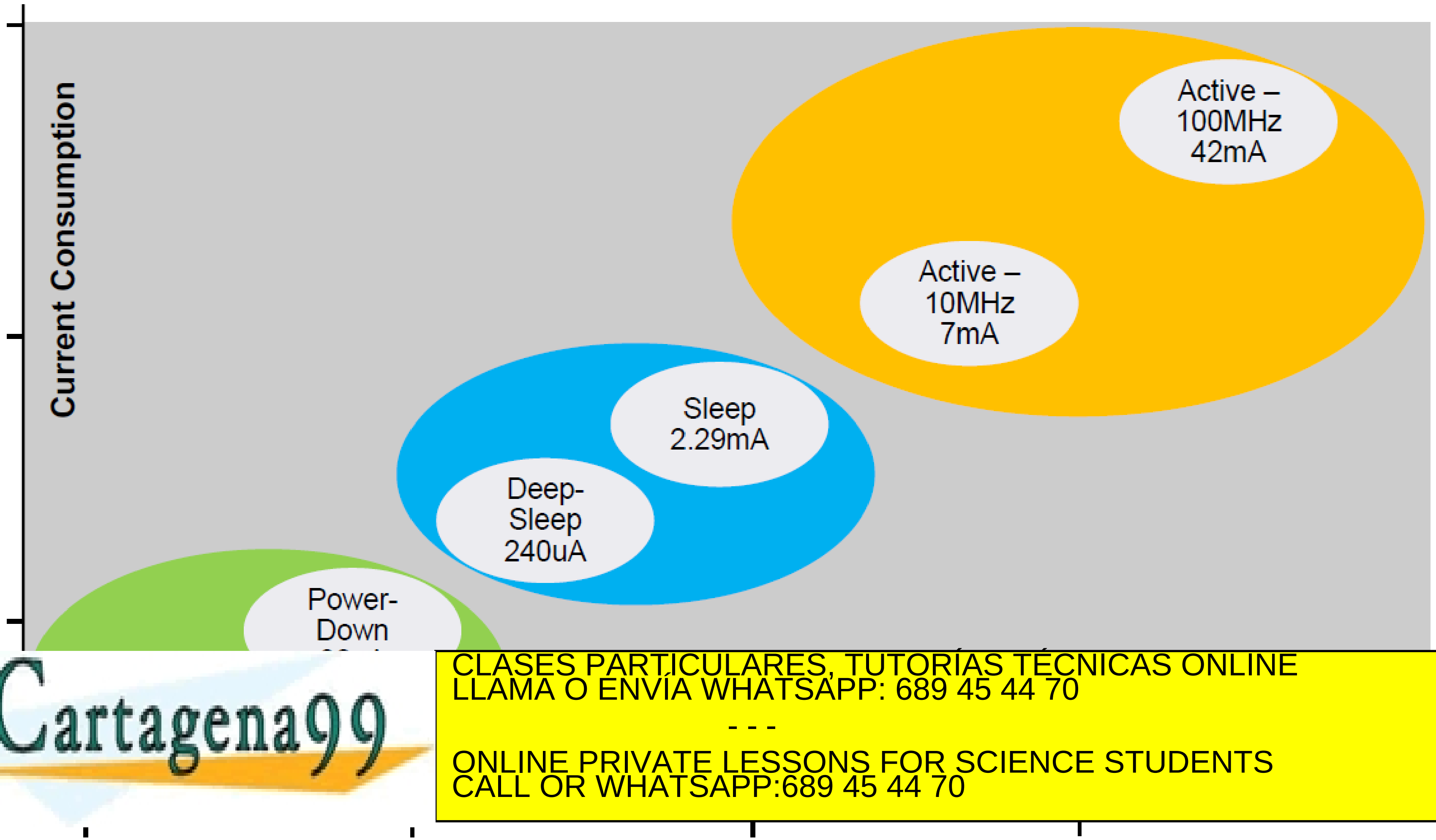


- **Sleep**
  - **CPU execution is suspended.**
  - **Peripherals continue running.**
  - **Ireg = 2.29 mA (@ 25º C)**
  - **State is preserved.**
- **Deep-Sleep***
  - **Main oscillator and all internal clocks except the IRC are stopped.**
  - **Flash memory is in standby, ready for immediate use.**
  - **Ireg = 240 μA (@ 25º C)**
  - **State is preserved.**
- **Power-down***
  - **Same as Deep-Sleep mode except Flash and IRC are shut down**
  - **Ireg = 30 μA (@ 25º C)**
  - **State is preserved.**
- **Deep power-down**
  - **All clocks including IRC are stopped. Internal voltage is turned off.**

***BOD Disable**

1.4. Power Modes (II)

LPC1700 Current Consumption Profile



1.4. Power Modes (III)

- **Wake-Up Controller (WIC)**
  - Allows "automatic" system wakeup from any priority interrupt that can occur while the clocks are stopped in any Power-down mode.
    - Does not require a separate enable for wakeup and interrupt.
  - When the CPU enters Power Down, Sleep or Deep Sleep modes by executing the **WFI** (Wait For Interrupt) instruction, the NVIC sends a mask of the current interrupt situation to the WIC.
  - This mask includes all of the interrupts that are both enabled and of sufficient priority to be serviced immediately
    - The WIC simply has to notice that one of these interrupts occurred and then wake up the CPU
    - Will also wake from CAN activity or USB activity
  - Eliminates the need to periodically wake up the controller and poll the