

# Diseño Basado en Microprocesadores

## Tema 2. Microcontroladores

- 2.1. Introducción a los microcontroladores
- 2.2. Entradas/Salidas Digitales
- 2.3. Temporizadores
- 2.4. Excepciones
- 2.5. Conversión Analógica/Digital
- 2.6. Comunicación serie RS232C
- 2.7. Teclado, conversión D/A y sonido
- 2.8. Interfaz I2C

## Tema 2.1. Introducción a los Microcontroladores

2.1.1. Introducción a los microcontroladores

2.1.2. Sistemas embebidos

2.1.3. Características de los microcontroladores

2.1.4. Aplicaciones de los microcontroladores

2.1.5. Familias de microcontroladores

2.1.6. Microcontroladores ARM

2.1.7. Arquitectura ARM

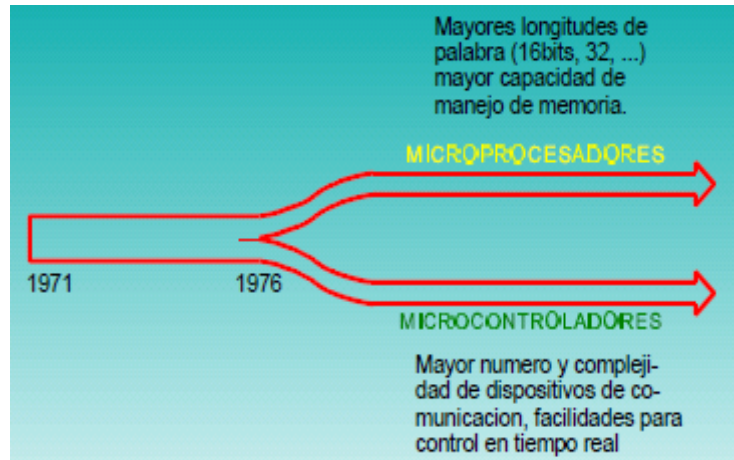
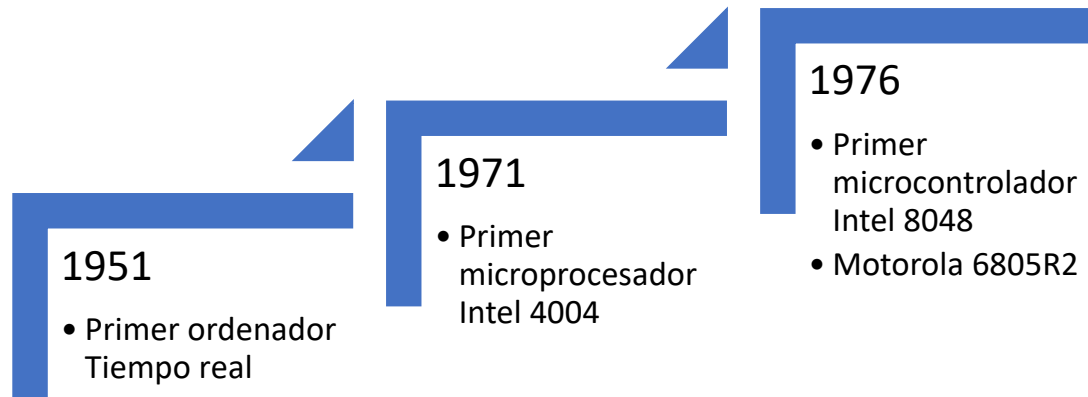
2.1.8. Características del LPC4088

2.1.9. Memoria del LPC4088

2.1.10. Programación del LPC4088

2.1.11. Acceso a los registros de los periféricos

## Introducción a los microcontroladores. Definición

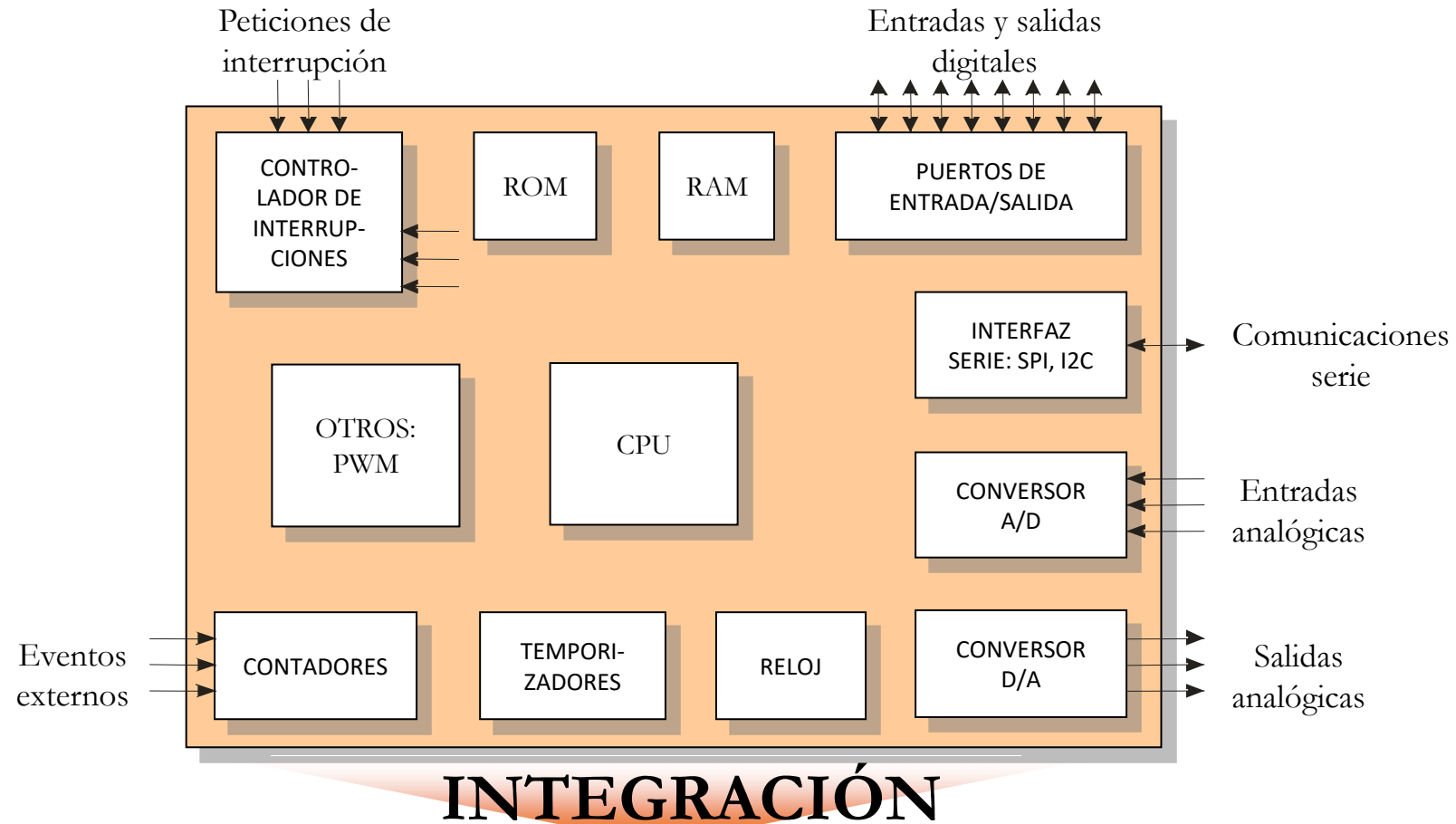


Un microcomputador es un microprocesador (CPU) más una serie de subsistemas en un mismo circuito integrado

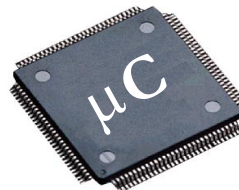


Un microcontrolador es un microcomputador especializado en tareas de monitorización y control de procesos construido sobre un único circuito integrado

## Diagrama de bloques de un microcontrolador



MICROCOMPUTADORES PARA APLICACIONES  
DE MONITORIZACIÓN Y CONTROL



## Periféricos integrados más comunes

### Oscilador de reloj

- Los microcontroladores incorporan en su interior el circuito generador de reloj que sincroniza todas las operaciones. Sólo son necesarios unos pocos componentes pasivos externos (cristal de cuarzo, resonador cerámico o red RC) para hacerlo funcionar.

### Puertos de entradas/salidas digitales

- La mayor parte de las patillas del encapsulado de un microcontrolador se destinan a líneas de entrada salida que comunican al microcomputador interno con el mundo exterior.

### Timers

- Los timers pueden funcionar como temporizadores o como contadores.
- Como temporizadores, permiten generar o medir intervalos de tiempo. Se utilizan en la generación de referencias de tiempo internas o para la medida indirecta de magnitudes que dependen del tiempo (velocidad, aceleración).
- Como contadores permiten llevar la cuenta de sucesos externos. Los sucesos se transforman en pulsos que se aplican a patillas de conteo del microcontrolador. Muchos sensores generan pulsos que, contados, indican el valor de la magnitud medida.

### Interfaces de comunicaciones

- Permiten el intercambio de datos con otros dispositivos (otros microcontroladores, PCs, dispositivos de E/S) mediante conexiones punto a punto o en red empleando protocolos estándar tales como RS-232, RS-485, I2C, SPI, CAN, USB, Ethernet, etc.

## Periféricos integrados más comunes (2)

### Controlador de interrupciones

- Gestiona las peticiones de interrupción de dispositivos externos y de los subsistemas internos permitiendo la habilitación/deshabilitación selectiva de cada fuente de interrupción y la asignación de prioridades.

### Convertor analógico/digital (ADC)

- Un convertor analógico/digital convierte un intervalo continuo de tensiones analógicas de entrada en un conjunto discreto de códigos digitales de salida.
- La inclusión de un convertor analógico/digital en un microcontrolador le permite medir magnitudes analógicas externas (temperatura, presión, velocidad, etc.) previa transformación en un valor de tensión equivalente.

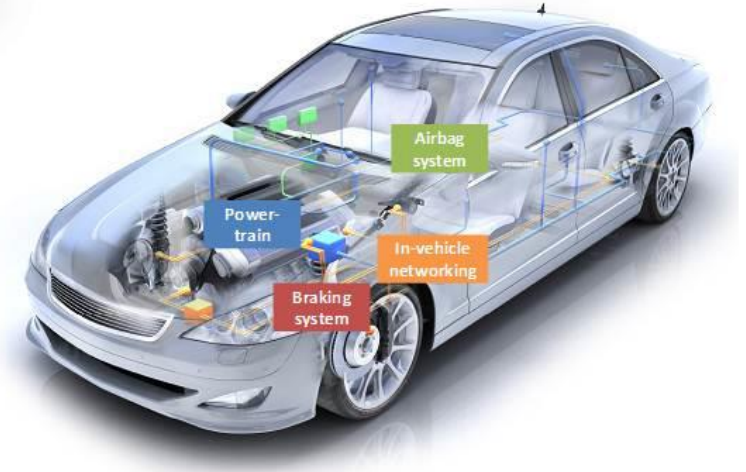
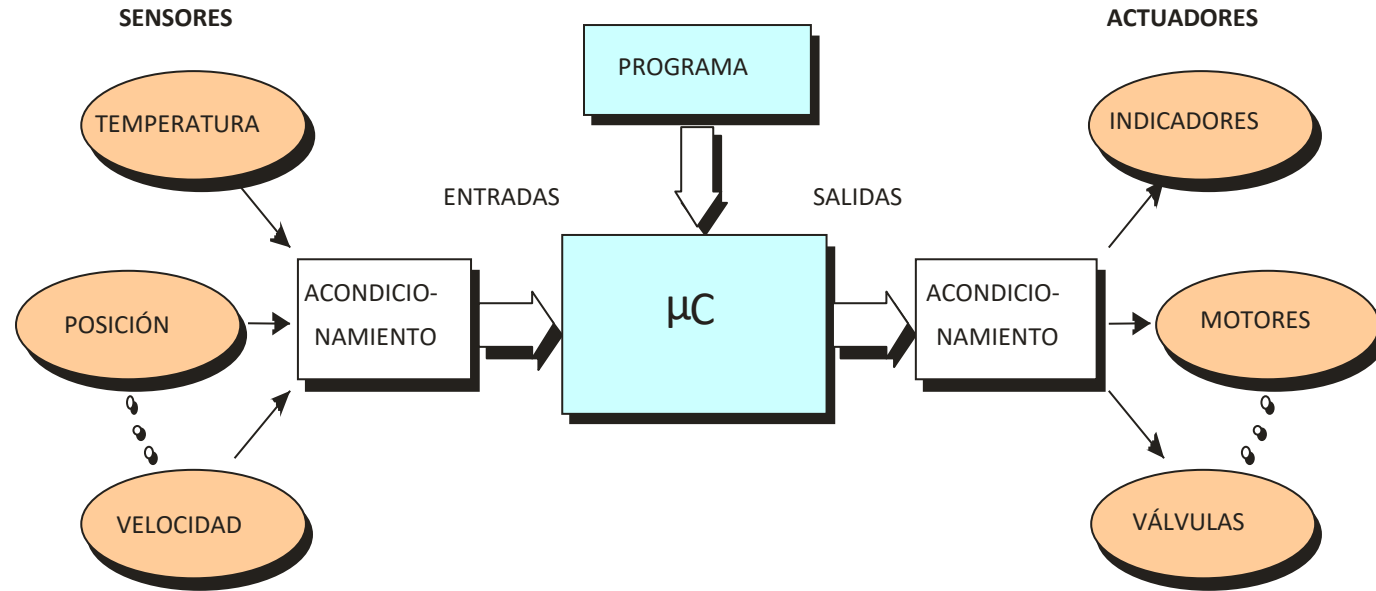
### Convertor digital/analógico (DAC)

- Un convertor digital/analógico convierte un dato digital de entrada en una tensión (o corriente) analógica de salida.
- Mediante el convertidor digital/analógico el microcontrolador puede generar señales analógicas, por ejemplo, señales de audio.

### Watchdog (Perro guardián)

- Tipo especial de temporizador que una vez puesto en marcha provoca el reset del microcontrolador si el programa no lo “refresca” periódicamente.
- Permite que el microcontrolador salga automáticamente de situaciones de bloqueo del programa.

## Sistemas embebidos



Suelen actuar como controladores embebidos (embedded controller)

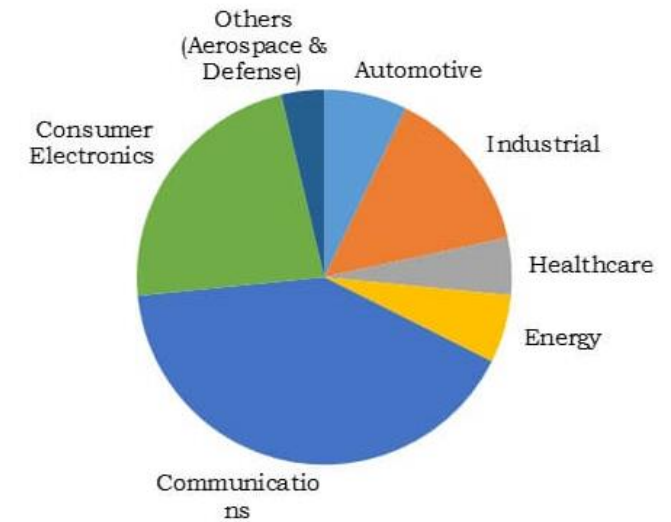
- Microcontrolador dedicado exclusivamente a una tarea e incorporado dentro del producto o sistema que gobierna sin que el usuario sea consciente de su presencia
- (*Sistema Empotrado, Incrustado*)



## Aplicaciones de los $\mu$ controladores



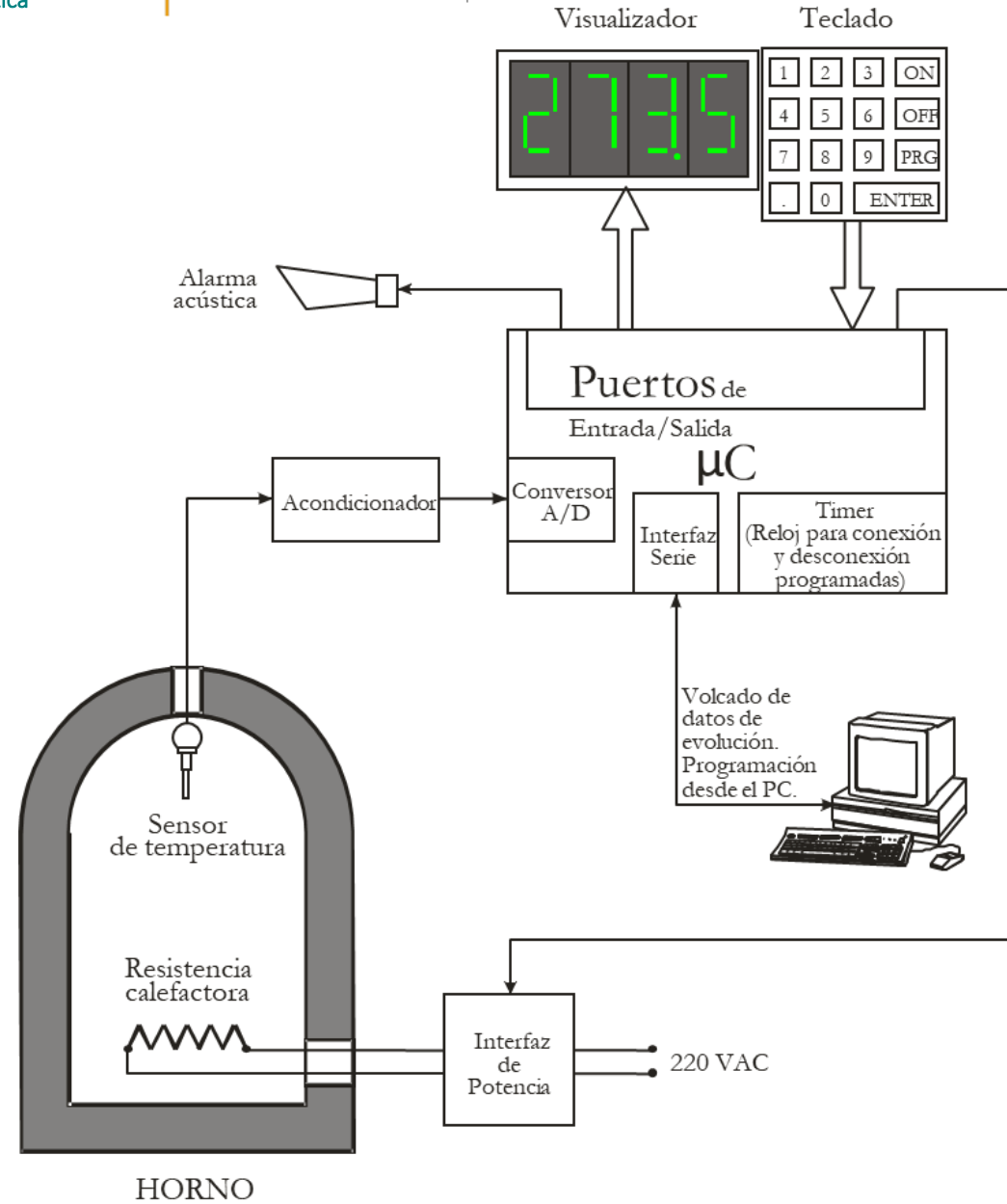
<https://www.alliedmarketresearch.com/embedded-computing-market>





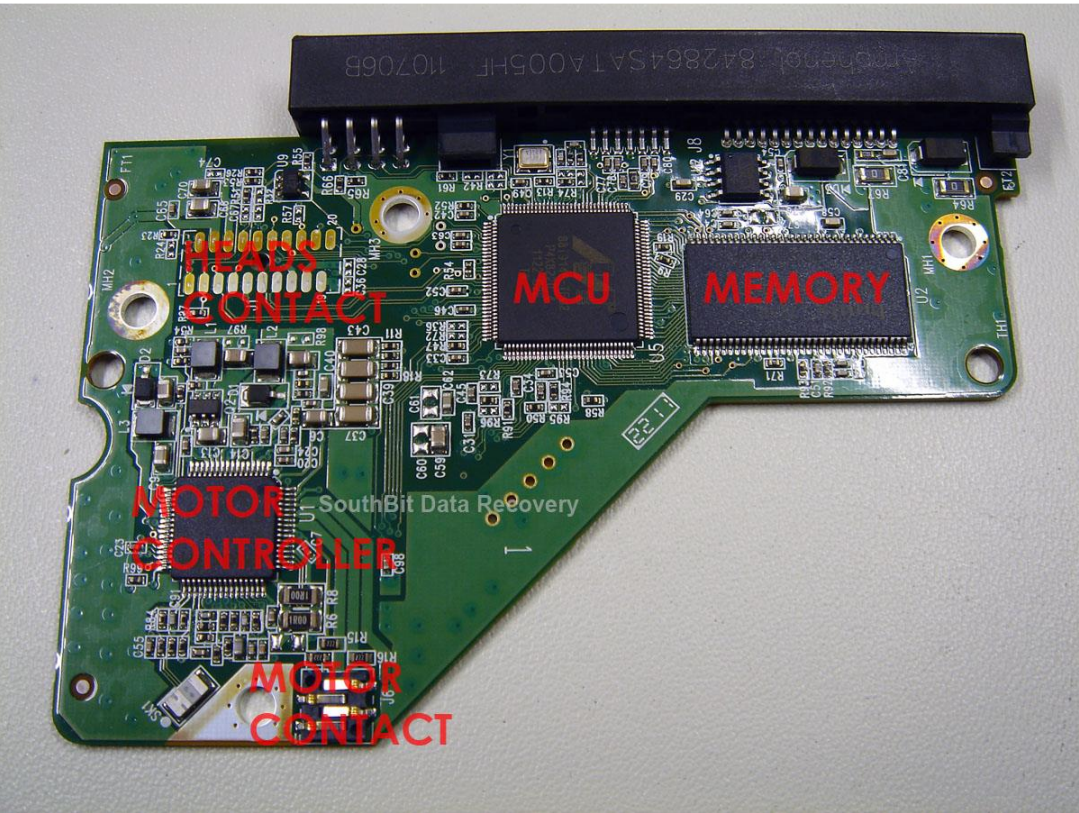
## Aplicaciones de los $\mu$ controladores (2)

### Control de la temperatura de un horno



# Aplicaciones de los $\mu$ controladores (3)

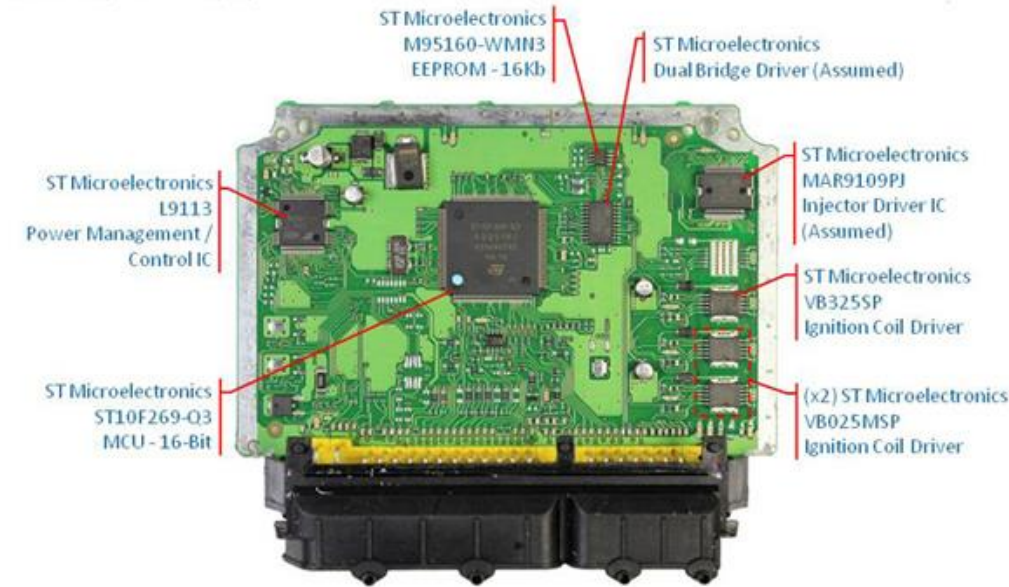
## Disco duro



## Unidad de control del motor de un automóvil

### Magneti Marelli IAW 4AC Engine Control Unit

Disassembly – Main PCB, Top



Source: IHS

## Características de los $\mu$ controladores

---

Estructura de sistema integrada: CPU + memoria + subsistemas de apoyo (periféricos)

---

Repertorio de instrucciones adaptados a funciones de control

---

Gestión eficiente de entrada/salida

---

Gran capacidad para atender las interrupciones (baja latencia): rapidez y nivel de prioridad

---

Arquitectura de multiprocesamiento

---

Controladores de periféricos inteligentes para aplicaciones específicas

---

Memoria RAM y ROM interna de gran capacidad y facilidad de implementación externa

---

Versiones de bajo consumo para aplicaciones especiales

---

Protección de los programas internos a prueba de intrusismo

---

Inmunidad al ruido eléctrico

---

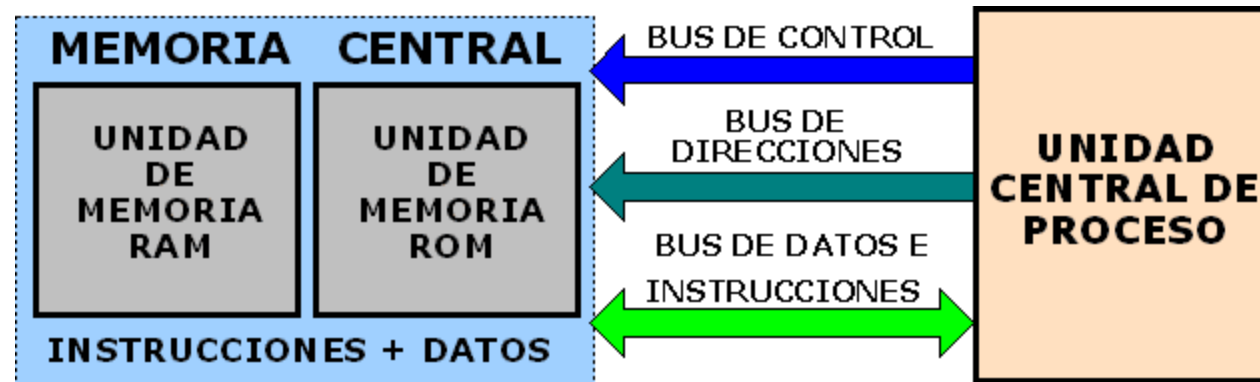
Márgenes amplios de tensiones de alimentación

---

## Clasificación de los microprocesadores (1)

### Arquitectura de Von Neumann

- Se caracteriza por disponer de una única memoria principal en la que se almacenan los datos y las instrucciones. A esta memoria se accede a través de un sistema de buses único (Bus de datos, Bus de direcciones, Bus de control)



## Clasificación de los microprocesadores (2)

### Arquitectura Harvard

- Se caracteriza por disponer de dos memorias, Memoria de datos y Memoria de Programa. Además, cada memoria dispone de su respectivo bus



## Microcontroladores ARM

---

ARM (Advanced RISC Machines) es una compañía fundada en 1990

---

Nació a partir de la empresa Acorn que diseñó las primeras versiones de la arquitectura

---

Diseña microprocesadores, pero no los fabrica

---

ARM licencia sus diseños a otras compañías que fabrican microprocesadores y otros sistemas

---

También diseña periféricos, arquitecturas de bus y herramientas de desarrollo





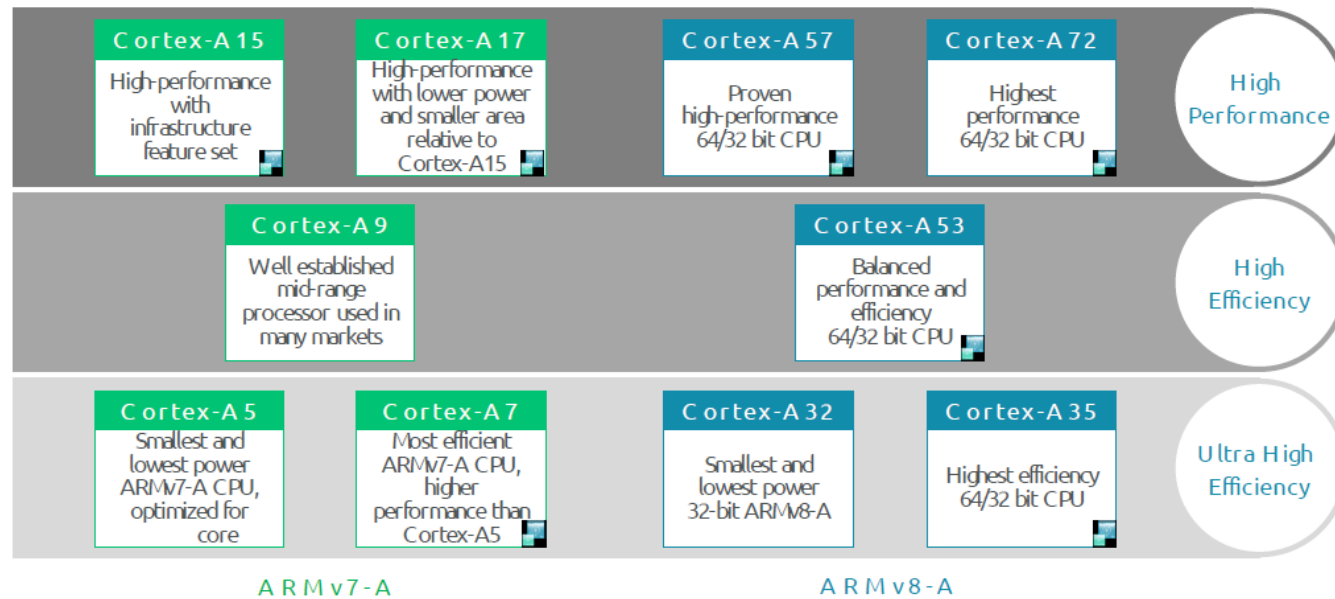




# Arquitectura ARM (1)

## Cortex-A

Altas prestaciones para aplicaciones y sistemas operativos complejos (Linux, Windows).  
Smartphones, ordenadores portátiles, SmartTV, servidores.

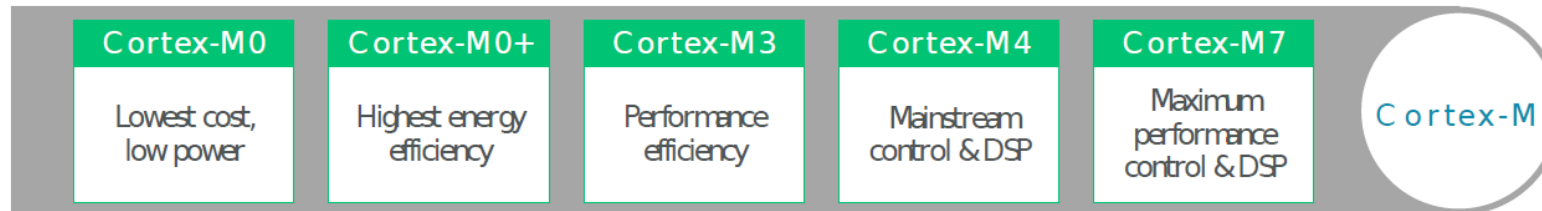


## Arquitectura ARM (2)

### Cortex-M

CPUs para microcontroladores y dispositivos de señal mixta. Equilibrio entre prestaciones, consumo y coste.

Control industrial, automoción, instrumentación, electrodomésticos, conectividad, periféricos de ordenador.

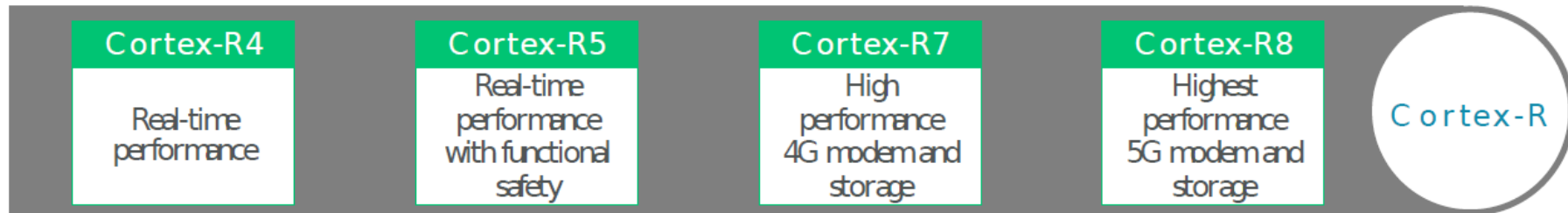


## Arquitectura ARM (3)

### Cortex-R

CPUs de altas prestaciones para aplicaciones de alta fiabilidad, tolerancia a fallos y respuesta de tiempo real determinista.

Discos duros, automoción, comunicaciones.



## μControladores de la familia LPC40xx

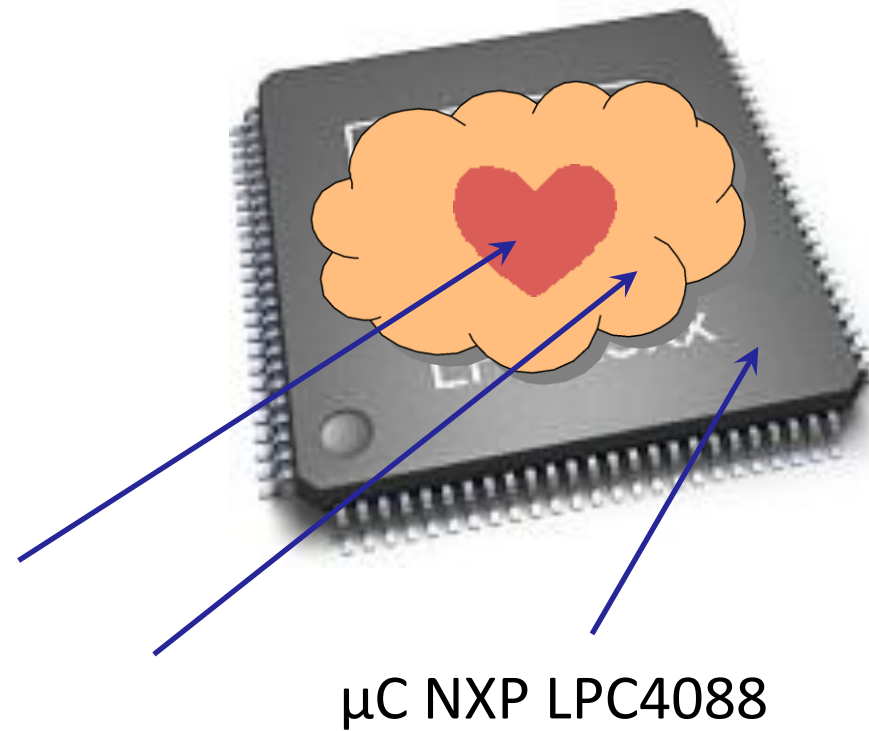


Type number	Core	Clock speed [max] (MHz)	Flash (kB)	RAM (kB)	EEPROM (kB)	Ethernet	USB	CAN	UART	I <sup>2</sup> C	SPI	ADC channels	ADC (bits)	DAC channels	DAC (bits)	Com ports	Timers	Timer (bits)	PWM	Package name	Temperature range (°C)
<a href="#">LPC4072FBD80</a>	Cortex-M4	120	64	24	2,048	0	1	2	4	3	3	8	12	1	10	0	4	32	6	LQFP80	-40 °C to +85 °C
<a href="#">LPC4072FET80</a>	Cortex-M4	120	64	24	2,048		1	2	4	3	3	8	12	1	10		4	32		TFBGA80	-40 °C to +85 °C
<a href="#">LPC4074FBD80</a>	Cortex-M4	120	128	40	2,048	0	1	2	4	3	3	8	12	1	10	0	4	32	6	LQFP80	-40 °C to +85 °C
<a href="#">LPC4074FBD144</a>	Cortex-M4	120	128	40	2,048	0	1	2	4	3	3	8	12	1	10	0	4	32	6	LQFP144	-40 °C to +85 °C
<a href="#">LPC4076FBD144</a>	Cortex-M4	120	256	80	2,048	1	1	2	5	3	3	8	12	1	10	2	4	32	6	LQFP144	-40 °C to +85 °C
<a href="#">LPC4076FET180</a>	Cortex-M4	120	256	80	2,048	1	1	2	5	3	3	8	12	1	10	2	4	32	6	TFBGA180	-40 °C to +85 °C
<a href="#">LPC4078FBD100</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	LQFP100	-40 °C to +85 °C
<a href="#">LPC4078FBD144</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	LQFP144	-40 °C to +85 °C
<a href="#">LPC4078FBD208</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	LQFP208	-40 °C to +85 °C
<a href="#">LPC4078FET180</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	TFBGA180	-40 °C to +85 °C
<a href="#">LPC4078FET208</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	TFBGA208	-40 °C to +85 °C
<a href="#">LPC4088FBD144</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	LQFP144	-40 °C to +85 °C
<a href="#">LPC4088FBD208</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	LQFP208	-40 °C to +85 °C
<a href="#">LPC4088FET180</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	TFBGA180	-40 °C to +85 °C
<a href="#">LPC4088FET208</a>	Cortex-M4	120	512	96	4,032	1	1	2	5	3	3	8	12	1	10	2	4	32	6	TFBGA208	-40 °C to +85 °C



## Características del LPC4088

- 32 bits de datos
- procesador genérico
- pipe-line (3)
- RISC (no puro)
- Estr. Mem. Harvard
- Bus externo abierto
- Formato Instr. Ortogonal
- Nucleo instr. ARMv7-M
- Procesador ARM Cortex-M4



## Características RISC

- Basado en el concepto RISC pero no RISC puro
  - Diseño más equilibrado apropiado para aplicaciones embebidas.
- Características RISC
  - Arquitectura carga/almacenamiento.
  - Juego de registros uniforme (16 registros de 32 bits) que pueden usarse como fuente o destino de la gran mayoría de las operaciones.
  - Juego de Instrucciones Thumb2 ISA de longitud de 16 y 32 bits.
  - Segmentación.
- Características no RISC
  - Ciclos de ejecución variable para algunas instrucciones.
  - Un desplazador en la entrada a la ALU (datapath) permite instrucciones más complejas.
  - Modos de direccionamiento más complejos.
  - Ejecución condicional de todas las instrucciones.



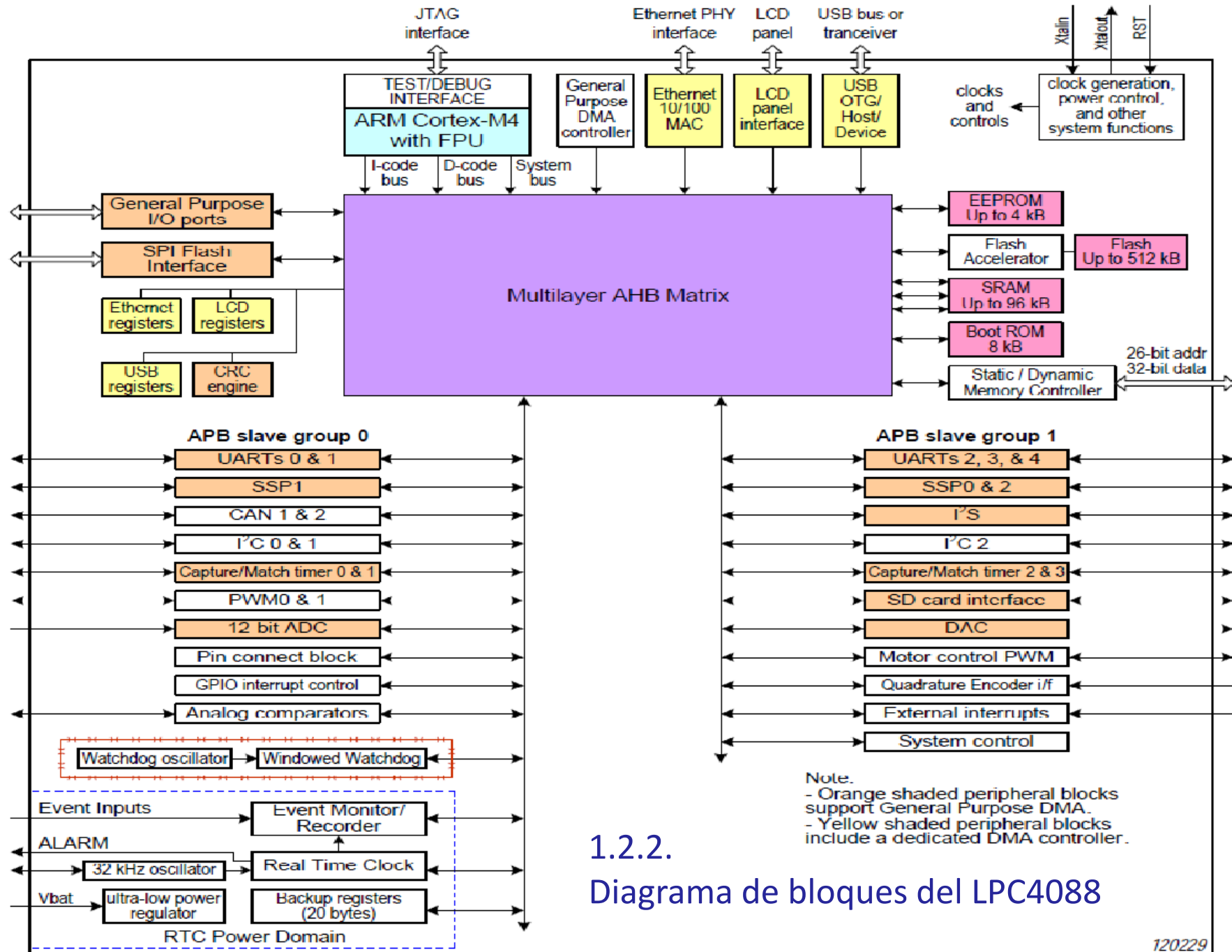
## Características del LPC4088. Periféricos

### Periféricos internos:

- 6 puertos de E/S. Total de 165 pines de E/S
- 4 Timers
- Convertidor A/D de 12 bits y 8 canales
- Convertidor D/A de 10 bits
- Dos comparadores analógicos
- Controlador de pantallas LCD
- Controlador tarjetas de memoria SD
- Controlador DMA de 8 canales
- Dos generadores PWM de 6 canales cada uno
- Controlador de motores de hasta tres fases
- Interfaz para encoder en cuadratura
- Reloj RTC
- Watchdog
- Calculador CRC
- Monitor/capturador de eventos

### Interfaces de comunicación integradas:

- Cinco UARTS
- Tres interfaces SSP (Modos SPI, TI SSI y Microwire)
- Tres interfaces I2C
- Interfaz I2S
- Interfaz CAN de dos canales
- Interfaz SPIFI para memorias Flash SPI
- Interfaz USB Full-speed device/host/OTG con DMA propio
- MAC Ethernet 10/100 Mb/s con DMA propio



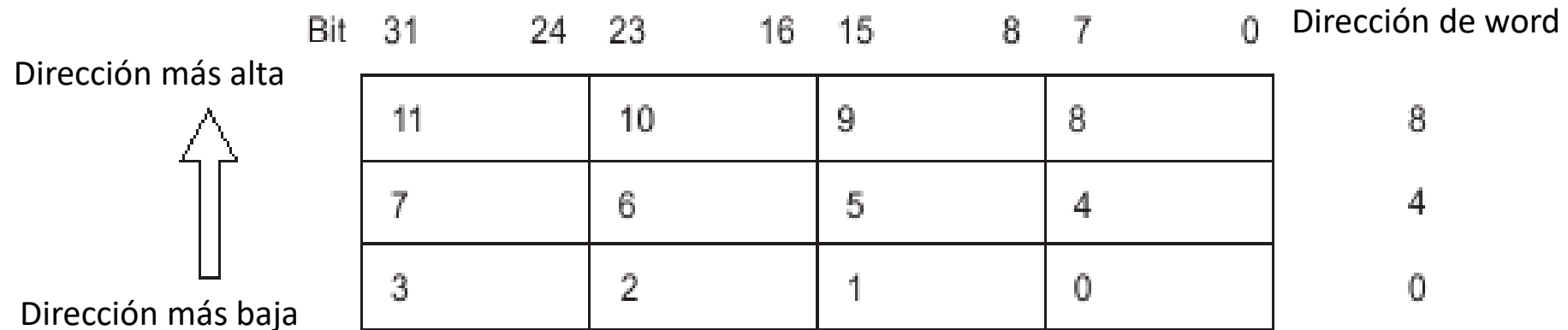
1.2.2.  
Diagrama de bloques del LPC4088

## Organización de datos en la memoria.

- La arquitectura ARM es de 32 bits de datos.
- En el contexto de la arquitectura ARM:
  - **Byte** es un dato de 8 bits.
  - **Halfword** (media palabra) es un dato de 16 bits.
  - **Word** es un dato de 32 bits.
  - **Doble** es un dato de 64 bits
- Existen implementaciones ARM que usan ordenación **“little endian”** y otras **“big endian”**.
- Los datos deben estar alineados:
  - Los halfwords deben estar en posiciones divisibles entre 2.
  - Los words deben estar en posiciones divisibles entre 4.

# Formato de almacenamiento de datos

## Formato Little Endian



Byte menos significativo dirección más baja

Word es direccionado por la dirección del byte menos significativo

# Sistemas de numeración y codificación de la información.

## Sistemas de numeración (base numérica):

**Binario:** dígitos (0..1)


**Decimal:** dígitos (0..9)

**Hexadecimal:** (0..15), 0..9 y A..F

## Codificación de números (n=nº de bits):

**Binario natural:** n (0..1)

**Binario entero:** n-1 (0..1)

**Binario real:**  $(-1)^S * 2^{(E - 127)} * (1 + F)$  SE..EF..F  


**BCD desempaquetado:** 1 BCD (0..9) en un byte

**BCD empaquetado:** 2 BCD en un byte

Caracteres de control ASCII			
DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de línea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII imprimibles								
DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	\$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_	elCodigoASCII.com.ar		

[elCodigoASCII.com.ar](http://elCodigoASCII.com.ar)

ASCII (128)  
ASCII ext. (256)  
Unicode

$'a' \equiv 97 \equiv 0x61$

# Sistemas de numeración y codificación de la información.

## Codificación de caracteres:

**ASCII-dec:** 0..31 códigos de control,

32..127 ASCII estándar,

128..255 ASCII extendido

**ASCII-hex:** 0x00..0x1F códigos de control,

0x20..0x7F ASCII estándar,

0x80..0xFF ASCII extendido

## Codificación de cadenas de caracteres:

"abc0123fgh"\0

0x61, 0x62, 0x63, 0x30, 0x31, 0x32, 0x66, 0x67, 0x68, 0x00

**!!!!Conversión de la codificación de la información: Pasar de un formato a otro!!!!**

- Conversión de una base numérica a otra
- Conversión de una codificación a otra
- Conversión de números a caracteres numéricos y viceversa



Binario natural	Byte	Halfword	word
Rango valores	0..255	0..65535	0..4294967295
Nº de bits	8	16	32
Nº valores	$2^8 = 256$	$2^{16} = 65536$	$2^{32} = 4294967296$

Ejemplos: 38 (d: 0..9) / 00100110 (b: 0..1) / 26 (h: 0..9 y A..F)

Binario entero – $ca2 = ca1 + 1$	Byte	Halfword	word
Rango valores	-128..127	-32768..32767	-2147483648.. 2147483647
Nº de bits	8	16	32
Nº valores	$2^8 = 256$	$2^{16} = 65536$	$2^{32} = 4294967296$

Ejemplos: -38 (d) / 11011010 (b) / DA (h)

Extensión de signos:  
(pasar de b a h y w)

nº + se añaden 0 a la izquierda  
nº - se añaden 1 a la izquierda

## Programación del LPC4088. Tipos de datos (1)

### Uno de los inconvenientes del lenguaje C

- Los tamaños de los tipos de datos no están estandarizados (Muchos compiladores de C diferentes antes del estándar. Hay que consultar cada compilador)

### Sólo se garantiza que

- $(\text{tamaño char}) \leq (\text{tamaño short}) \leq (\text{tamaño int}) \leq (\text{tamaño long}) \leq (\text{tamaño long long})$
- $(\text{tamaño char}) \leq (\text{tamaño float}) \leq (\text{tamaño double}) \leq (\text{tamaño long double})$
- Tamaño char > 8 bits
- Tamaño int > 16 bits
- Tamaño long > 32 bits
- Tamaño long long > 64 bits

## Programación del LPC4088. Tipos de datos (2)

### Tamaños típicos de tipos básicos en compiladores de 16 bits y 32 bits

- Un mismo tipo puede tener distinto tamaño según el compilador
- Además, hay tipos enteros del mismo tamaño en cada compilador

### Todo esto puede provocar

- Errores al elegir el tipo correcto para las variables
- Errores al analizar el funcionamiento de un programa
- Problemas al portar un programa a otra arquitectura

Tipo	Compilador 16 bits	Compilador 32 bits
char	8 ¿con signo?	8 ¿con signo?
short	16	16
int	16	32
long	32	32
long long	64	64
float	32	32
double	64	64

## Programación del LPC4088. Tipos de datos (3)

Para evitar estos problemas es mejor no usar los tipos de datos básicos de C. En su lugar, pueden usarse los tipos enteros definidos en `stdint.h`: El `stdint.h` que acompaña a cada compilador define `int8_t`, `uint8_t`, etc. De forma que, independientemente de los tamaños concretos de los tipos base en ese compilador y de si `char` es allí un tipo con o sin signo, éstos tengan siempre los tamaños y capacidad de representar signo correctos

Tipo	Bits	Signo	Rango
<code>int8_t</code>	8	Sí	−128 a 127
<code>uint8_t</code>	8	No	0 a 255
<code>int16_t</code>	16	Sí	−32768 a 32767
<code>uint16_t</code>	16	No	0 a 65535
<code>int32_t</code>	32	Sí	−2147483648 a 2147483647
<code>uint32_t</code>	32	No	0 a 4294967295
<code>int64_t</code>	64	Sí	−232 a 232 − 1
<code>uint64_t</code>	64	No	0 a 264 − 1

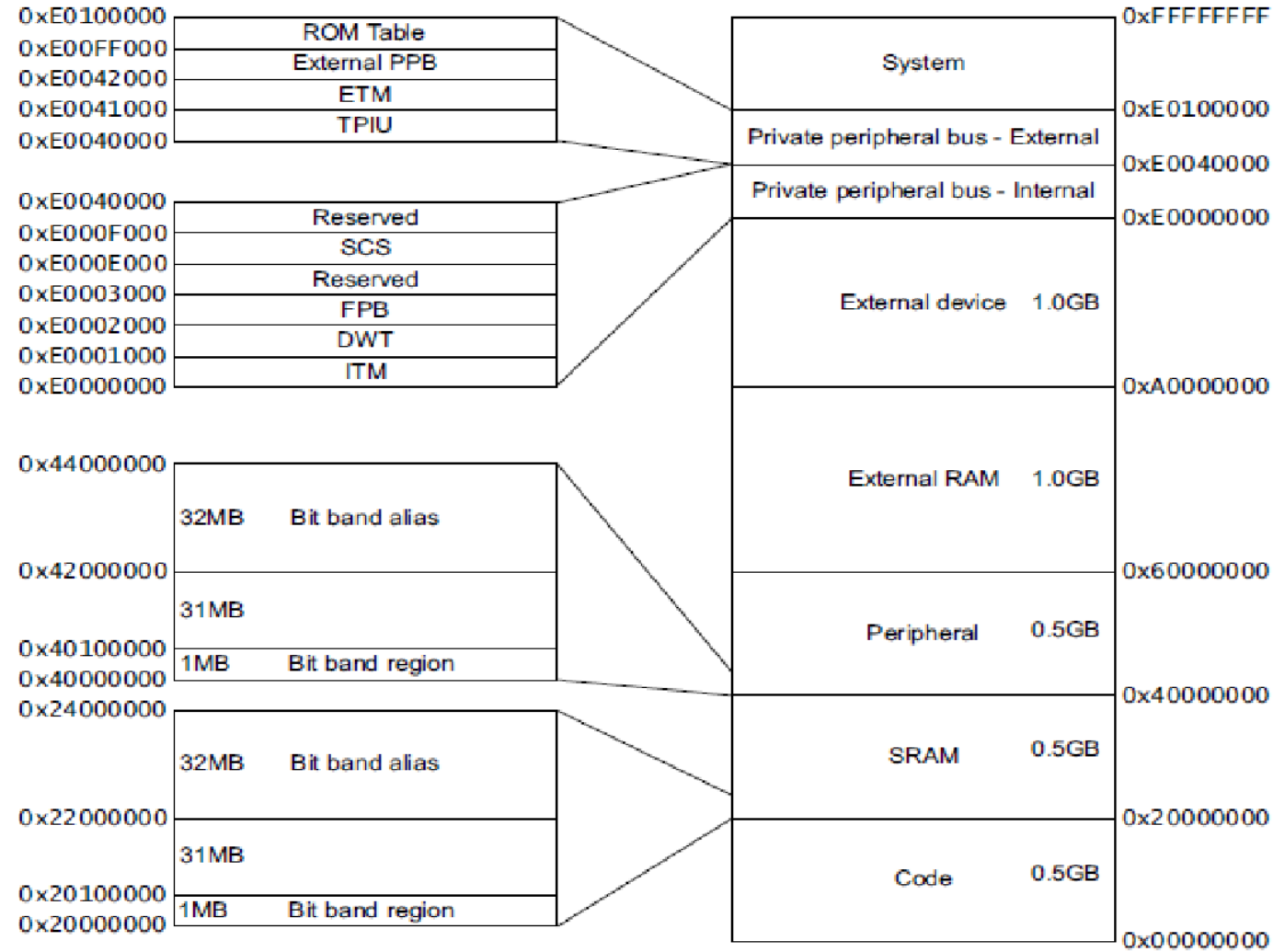
Compilador típico 16 bits	Compilador típico 32 bits
<code>typedef signed char int8_t;</code>	<code>typedef signed char int8_t;</code>
<code>typedef signed int int16_t;</code>	<code>typedef signed short int int16_t;</code>
<code>typedef signed long int int32_t;</code>	<code>typedef signed int int32_t;</code>
<code>typedef signed long long int int64_t;</code>	<code>typedef signed long long int int64_t;</code>
<code>typedef unsigned char uint8_t;</code>	<code>typedef unsigned char uint8_t;</code>
<code>typedef unsigned int uint16_t;</code>	<code>typedef unsigned short int uint16_t;</code>
<code>typedef unsigned long int uint32_t;</code>	<code>typedef unsigned int uint32_t;</code>
<code>typedef unsigned long long int uint64_t;</code>	<code>typedef unsigned long long int uint64_t;</code>

El tipo `char` seguirá usándose, pero sólo para almacenar caracteres ASCII. Necesario por compatibilidad con código ajeno (ej.: la biblioteca estándar usa `char`). Para datos de 8 bits que no sean caracteres ASCII se usará `int8_t` o `uint8_t`.

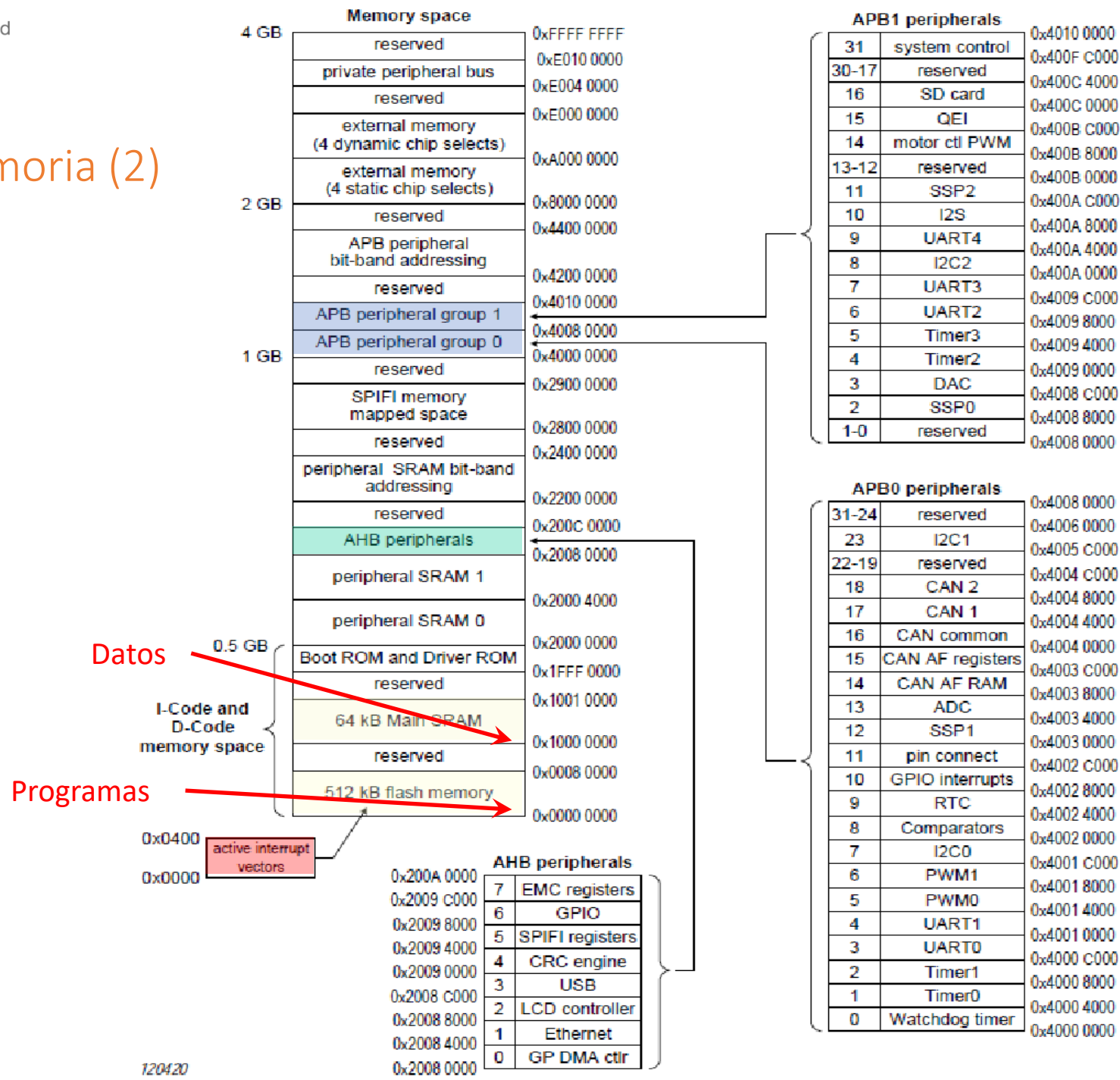
Los tipos `float` y `double` raramente presentan ambigüedades: Coinciden con los tipos IEEE-754 de precisión simple (32 bits) y doble (64 bits). Aun así, es común definir los tipos `float32_t` y `float64_t` para hacer patentes sus tamaños.

```
typedef float float32_t; typedef double float64_t;
```

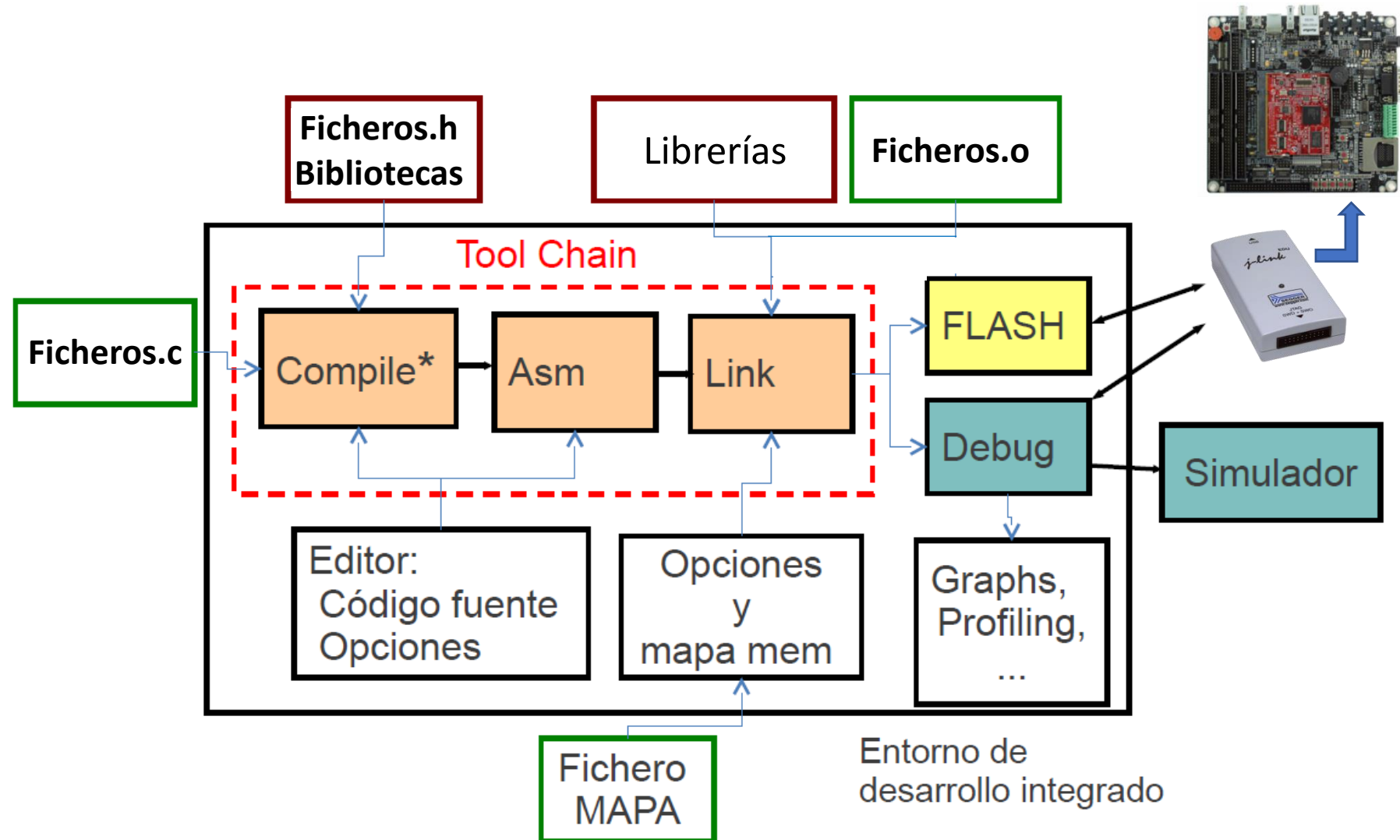
## Mapa de Memoria (1)



## Mapa de Memoria (2)

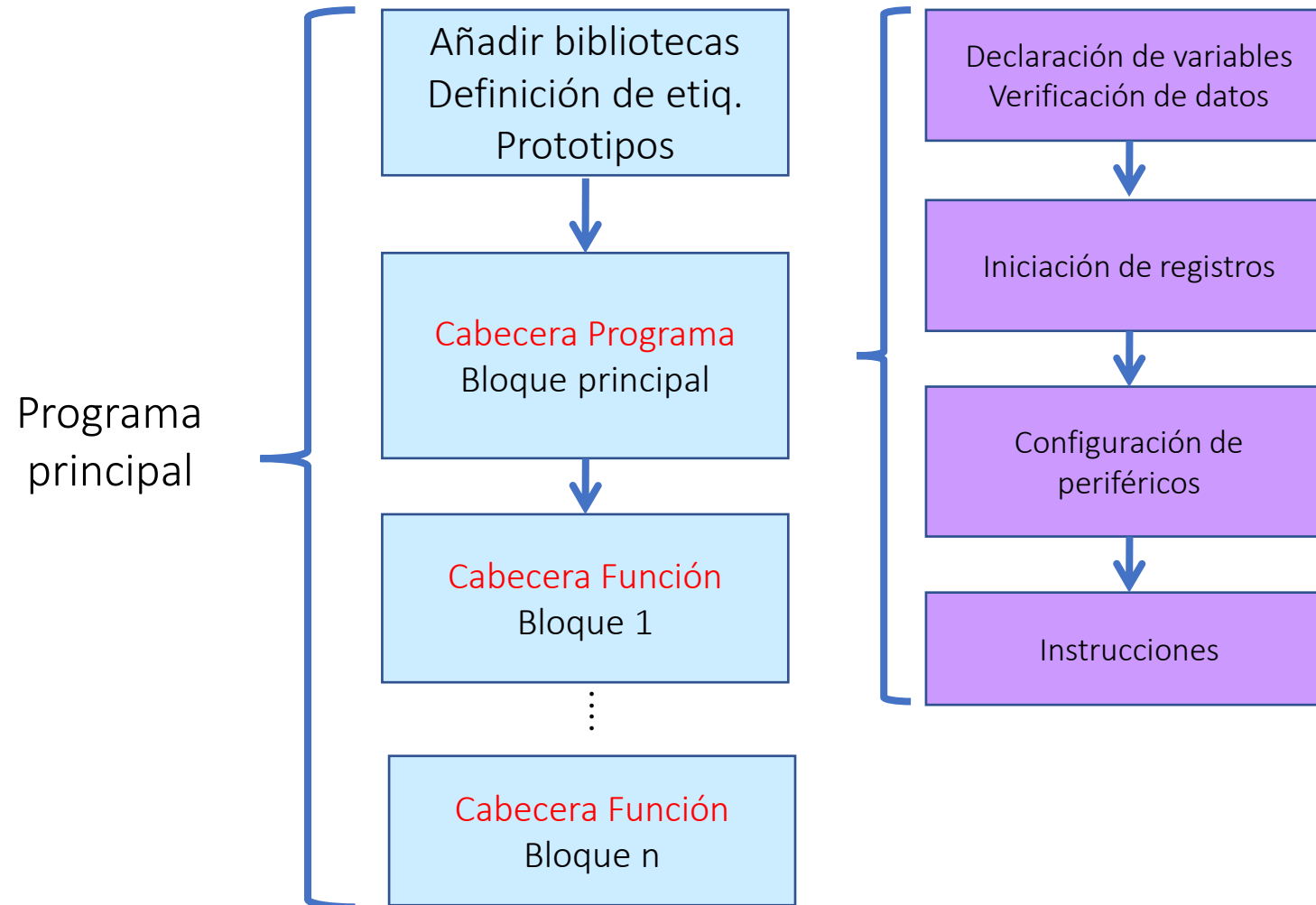


# Programación del LPC4088





## Estructura de los programas



## Programación del LPC4088. Macros

### Macros para control de error:

```
ERROR (mensaje);  
ASSERT (expresión, mensaje);
```

```
#ifndef ERROR_H  
#define ERROR_H  
#define HABILITAR_ASSERT 1  
  
/*===== Macros =====*/  
#define ERROR(mensaje) parar_con_error(__FILE__, __FUNCTION__, __LINE__, mensaje)  
  
#if HABILITAR_ASSERT != 0  
#define ASSERT(expr, mensaje)\  
if (!(expr)) {parar_con_error(__FILE__, __FUNCTION__, __LINE__, mensaje);}\  
#else  
#define ASSERT(expr, mensaje)  
#endif /* HABILITAR_ASSERT */  
  
/*===== Prototipos de funciones =====*/  
void parar_con_error(const char *fichero,  
                    const char *funcion,  
                    const uint32_t linea,  
                    const char *mensaje);  
  
#endif /* ERROR_H */
```

## Acceso a los registros de los periféricos

Definición de etiquetas para las direcciones de memoria de los registros:

```
/*dirección base de periféricos con bus APB0*/  
#define LPC_APB0_BASE (0x40000000UL)
```

```
/*dirección de registros del periférico*/  
#define LPC_periferico_BASE (LPC_APB0_BASE + 0x2C000)
```

```
/*asociación de una estructura a la dirección base de periféricos*/  
#define LPC_periferico ((LPC_periferico_TypeDef *) LPC_periferico_BASE)
```

```
typedef struct{  
    __IO uint32_t reg_0;           //0x000  
    __IO uint32_t reg_1;  
    .....  
    __IO uint32_t reg_n-1;  
    __IO uint32_t reg_n;           // 0x290  
} LPC_periferico_TypeDef;
```

Para leer de cada registro de cada periférico:

```
var = LPC_periferico -> reg_x  
/*reg_x alguno de los registros definidos en la  
estructura*/
```

Para escribir en cada registro de cada periférico:

```
LPC_periferico -> reg_x = valor  
/*reg_x alguno de los registros definidos en la estructura*/
```

