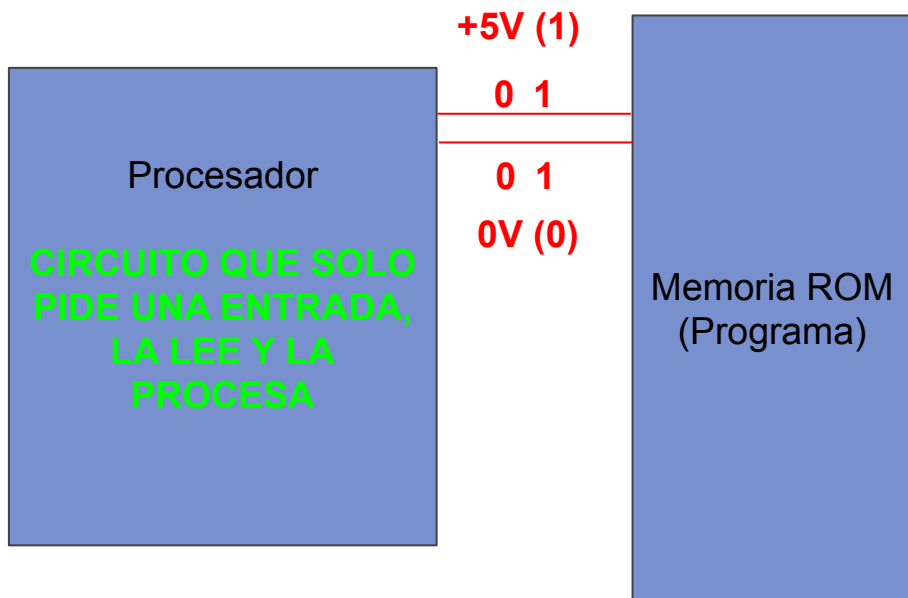


Introducción a la arquitectura Cortex M0

Informática II - R2004

Recordando... ¿cómo funcionaba un procesador?

¿Cómo funciona un circuito electrónico inteligente?

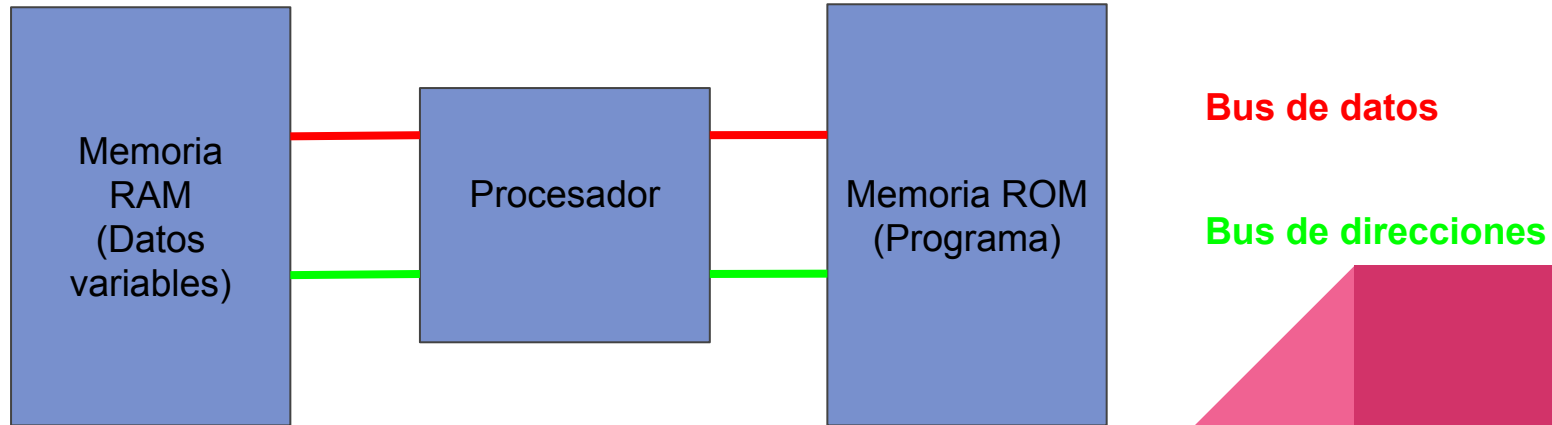


- | | | |
|-----------|--|-------------------|
| 1 cable: | 0 ó 1 | - 2 instrucciones |
| 2 cables: | 00, 01, 10 ó 11 | - 4 instrucciones |
| 3 cables: | 000, 001, 010, 011
100, 101, 110, 111 | - 8 instrucciones |
| N cables: | - 2^N instrucciones | |

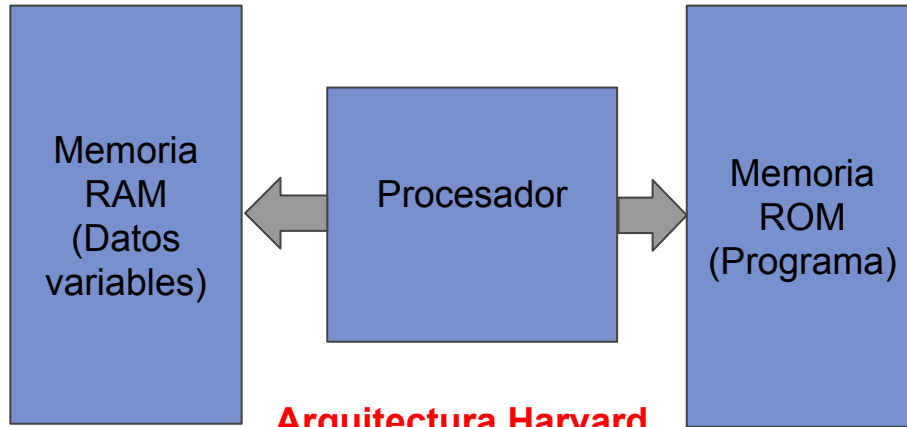
LA CANTIDAD DE CABLES (TAMAÑO DEL BUS)
ME INDICA CUÁNTAS INSTRUCCIONES PUEDO
PEDIR, O SEA EL TAMAÑO DE LA MEMORIA

Un poco de perspectiva

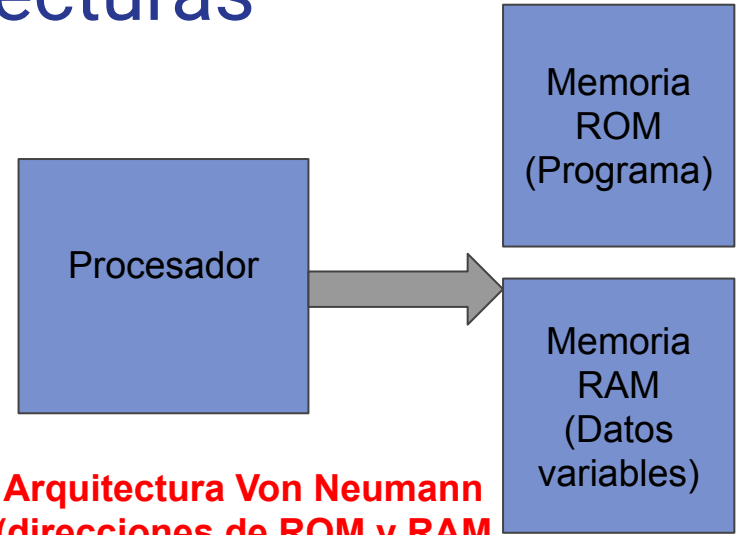
Las instrucciones una vez grabadas no cambian hasta que cargue un nuevo programa, en cambio las variables (datos) en el programa se van modificando a lo largo del mismo. Esto diferencia 2 tipos de memoria a las que puedo acceder: Memoria de programa (ROM) y memoria de datos (RAM). Sin embargo la forma de acceso es similar: Necesito un bus de direcciones, para indicar que posición de memoria quiero escribir o leer, y otro bus para poner el dato a escribir o leer



Tipos de arquitecturas



Arquitectura Harvard
(buses separados para memoria y datos)

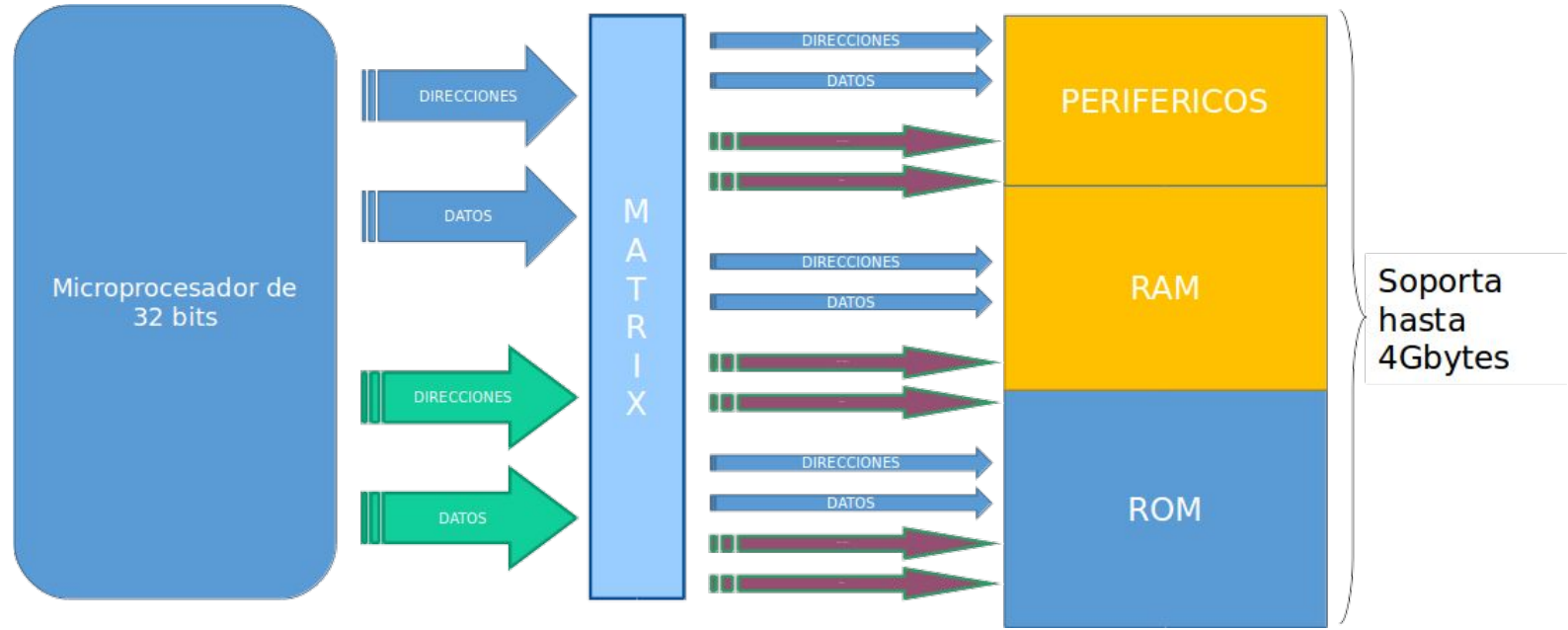


Arquitectura Von Neumann
(direcciones de ROM y RAM correlativas)

Basados en esta distinción, las memorias de datos y de programa se pueden organizar de distintas maneras. A estos distintos tipos de organización se los conoce como **arquitecturas**. La arquitectura

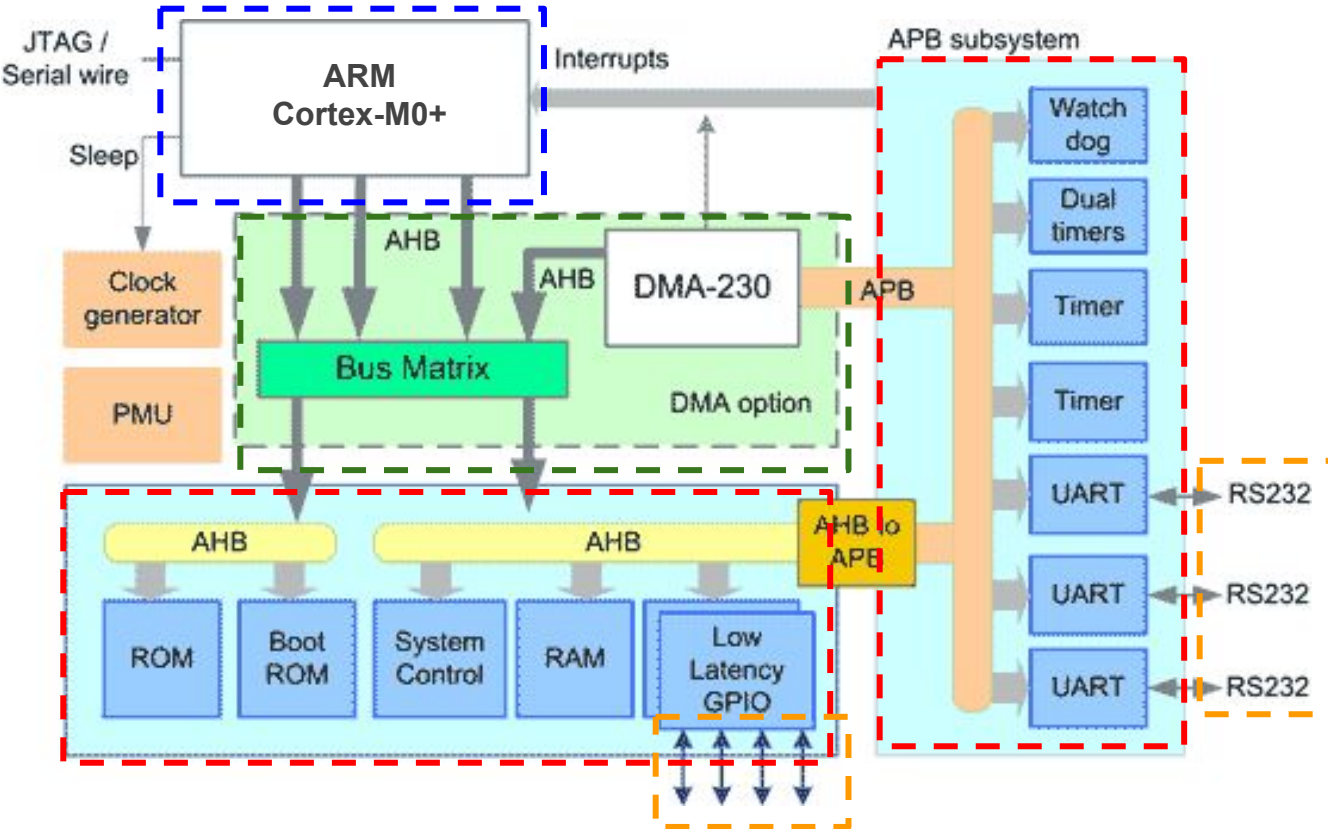
Harvard tiene 2 pares de buses, por lo que la dirección 0xC034, por ejemplo puede referirse a una instrucción de programa o a una variable. La arquitectura Von Neumann, aprovecha que el funcionamiento de los buses es similar y utiliza el mismo par de buses para acceder a datos y al programa. Esto significa que para que no haya colisiones, no pueden repetirse las direcciones (los datos deben estar “a continuación” de las variables, en direcciones distintas)

Arquitectura Cortex M0



El uP Cortex M0 utiliza conceptos de las 2 arquitecturas: Tiene 2 pares de buses (como la arq. Harvard), pero a su vez una matriz le permite usar estos buses para acceder tanto a memoria RAM, como a ROM, como incluso a **PERIFÉRICOS**. Esto le permite ejecutar instrucciones desde un periférico, o desde la RAM, o tratar a los periféricos como si fueran una variable más que se puede modificar.

Arquitectura Cortex M0 (II) - ¿Qué dice el manual?



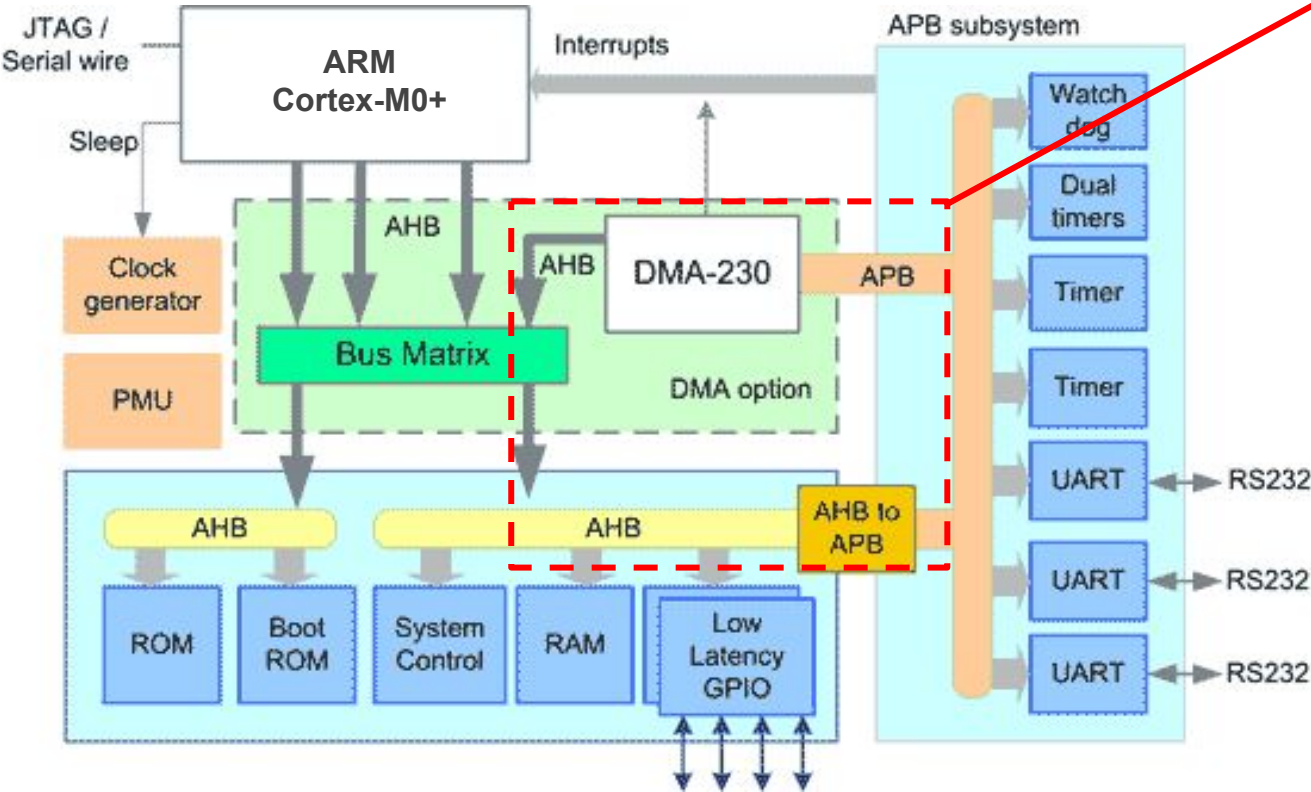
Periféricos: “hardware”, o dispositivos integrados que puedo controlar desde el procesador

Procesador: Circuito que pide instrucciones y las ejecuta.

Buses: Comunicación entre el procesador y la memoria y periféricos

E/S: Comunicación con el exterior

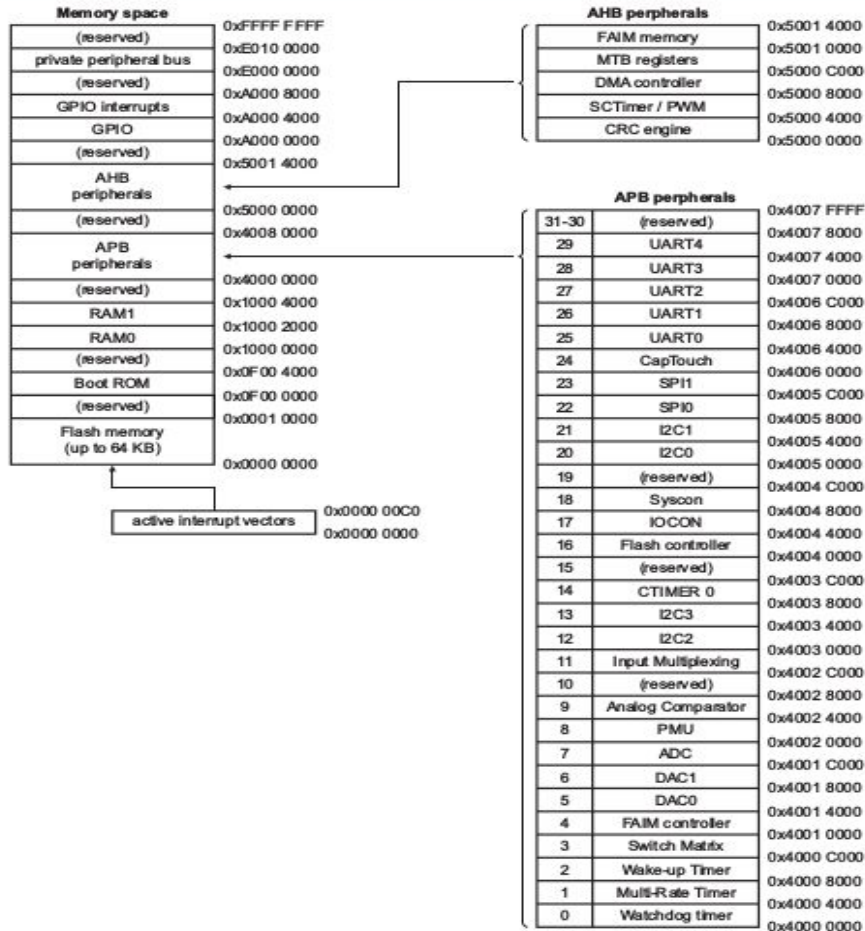
Arquitectura Cortex M0 (II)



¿Qué significa esta parte del esquema?

Implica que los periféricos (UART, Timers, etc.), que son **PERIFÉRICOS**, pueden comunicarse con otros periféricos (memoria RAM, o entradas/salidas digitales), sin hacer uso del procesador, por medio de un bloque que se llama **DMA** (Direct Memory Access)

Arquitectura Cortex M0 (III) - Mapa de memoria



(Tomado del manual del microcontrolador)

El manual nos muestra como los 4GB de memoria direccionable está dividida. Empezando de la dirección 0, comienza la tabla de interrupciones, la memoria ROM, la RAM, y luego los periféricos (a partir de la dirección 0x40000000 en adelante). Lo indicado como “reservado” está pensado para futuros desarrollos.

Entonces... ¿Cuál es la diferencia entre memoria y periférico?

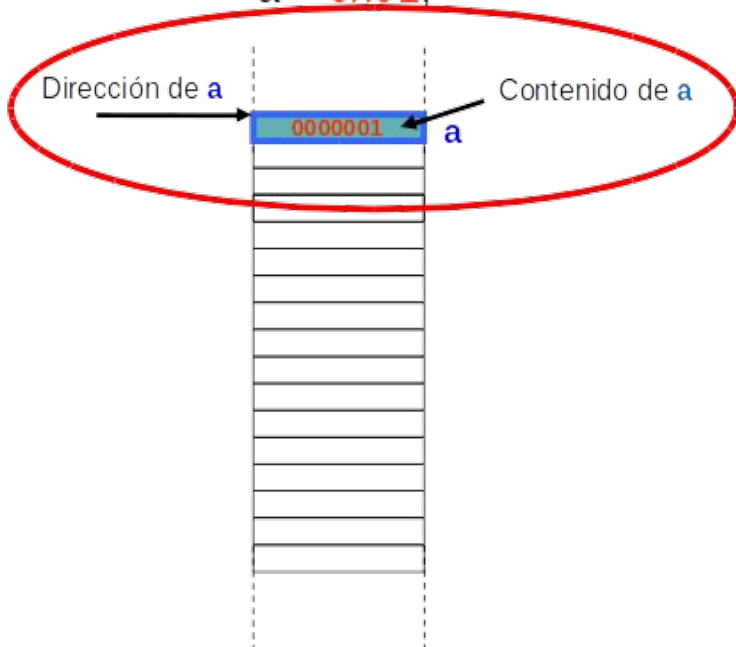


Microprocesador

MEMORIA

unsigned char **a**;

a = 0x01;

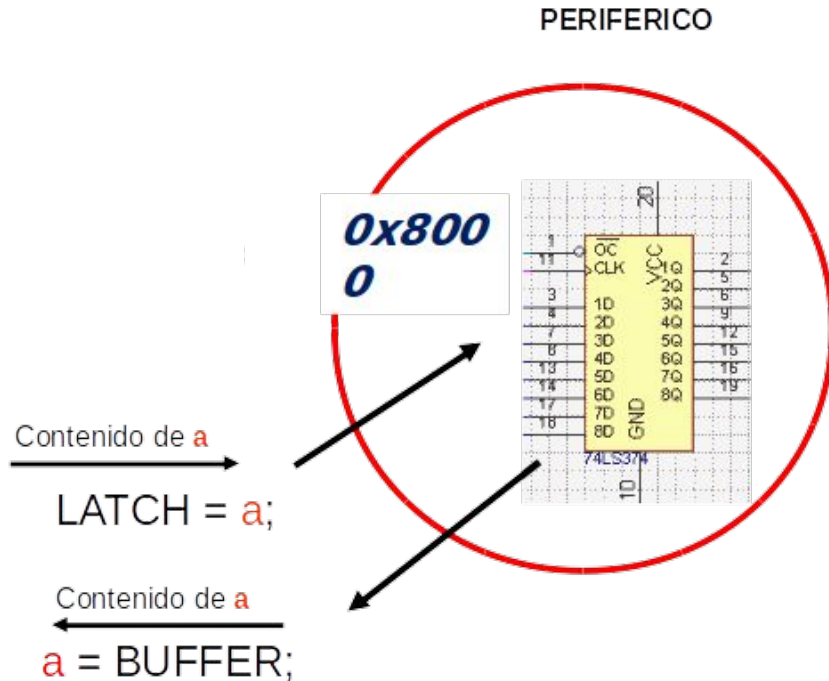


Memoria

Cuando escribo una dirección asociada con una memoria (RAM o ROM), estoy tomando un espacio de almacenamiento específico y cambiando su valor. El mismo será retenido hasta la próxima escritura.

Una lectura de esta misma dirección de memoria nos devolverá exactamente el mismo valor que se escribió anteriormente.

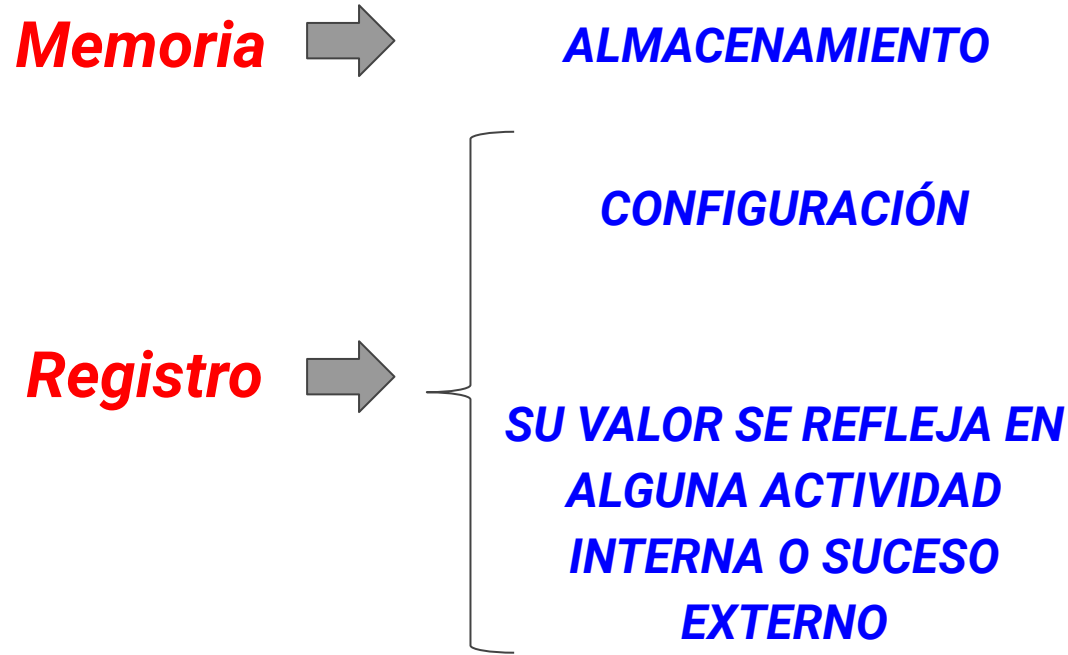
Entonces... ¿Cuál es la diferencia entre memoria y periférico?



Cuando **escribo** una dirección asociada con un periférico (**REGISTRO**), estoy indicando a dicho periférico que estoy ejecutando una instrucción de **ESCRITURA**, y poniendo sus entradas en un determinado valor, que tiene que ver con la conexión del bus de datos al periférico en cuestión. El resultado de cambiar la entrada de un integrado tiene que ver con el funcionamiento del mismo (puede prender una salida, activar un timer, generar una interrupción, etc.)

Cuando **leo** una dirección asociada con un periférico (**REGISTRO**), estoy indicándole a este periférico que quiero hacer una operación de **LECTURA**. El periférico, entonces, pondrá en el bus de datos algo que tiene que ver con su funcionamiento, que en mi programa almacenaré en alguna variable. Este dato tiene que ver con el funcionamiento del periférico (puede ser el valor de una entrada, el valor de configuración de un timer, el estado de funcionamiento del periférico, etc.)

Resumiendo...



No podemos suponer que si escribimos un valor en un periférico, luego leeremos el mismo valor, esto depende del funcionamiento del periférico en cuestión, y para cada periférico se puede ver en el manual del microcontrolador


¿Cómo leemos una hoja de datos?

UM11029.pdf — LPC84x User manual

1 de 535

Índice

- Chapter 1: LPC84x Introductory information 4
- Chapter 2: LPC84x memory mapping 10
- Chapter 3: LPC84x Boot Process 12
- Chapter 4: LPC84x FAIM 17
- Chapter 5: LPC84x ISP and IAP 22
- Chapter 6: LPC84x Flash signature generator 62
- Chapter 7: LPC84x Nested Vectored Interrupt ... 67
- Chapter 8: LPC84x System configuration (SYS... 79
- Chapter 9: LPC84x FRO API ROM routine 125
- Chapter 10: LPC84x Switch matrix (SWM) 127
- Chapter 11: LPC84x I/O Configuration (IOCON) 147
- Chapter 12: LPC84x General Purpose I/O (GPIO) 208
- Chapter 13: LPC84x Pin interrupts/pattern ma... 216
- Chapter 14: LPC84x Input multiplexing and D... 241
- Chapter 15: LPC84x Reduced power modes an... 247
- Chapter 16: LPC84x DMA controller 262
- Chapter 17: LPC84x USART0/1/2/3/4 285
- Chapter 18: LPC84x SPI0/1 308
- Chapter 19: LPC84x I2C0/1/2/3 331
- Chapter 20: LPC84x Standard counter/timer (C... 359
- Chapter 21: LPC84x SCTimer/PWM 375
- Chapter 22: LPC84x Windowed Watchdog Tim... 416
- Chapter 23: LPC84x Self-wake-up timer (WKT) 425
- Chapter 24: LPC84x Multi-Rate Timer (MRT) 429
- Chapter 25: LPC84x System tick timer (SysTick) 436
- Chapter 26: LPC84x Capacitive Touch 440
- Chapter 27: LPC84x 12-bit Analog-to-Digital C... 453
- Chapter 28: LPC84x Digital-to-Analog Convert... 485
- Chapter 29: LPC84x Analog comparator 490
- Chapter 30: LPC84x CRC engine 496
- Chapter 31: LPC84x ROM API integer divide ro... 501
- Chapter 32: LPC84x Serial Wire Debug (SWD) 505

 **UM11029**
LPC84x User manual
Rev. 1.7 — 14 April 2021

User manual

Document information

Info	Content
Keywords	LPC84x, LPC84x UIM, LPC84x user manual
Abstract	LPC84x User manual

El manual que utilizaremos para el LPC845 puede descargarse de:
<https://www.nxp.com/webapp/Download?colCode=UM11029>

Por ejemplo: ¿Cómo accedemos al registro para modificar el valor de un pin del micro? (pag 211)

Table 260. GPIO port pin register (PIN[0:1], addresses 0xA000 2100 (PIN0) to 0xA0000 2104 (PIN1)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	PORT	Reads pin states or loads output bits (bit 0 = PIOm_0, bit 1 = PIOm_1, ..., bit 31 = PIOm_31). m = port 0 to 1; n = pin 0 to 31 for port 0 and pin 0 to 21 for port 1. 0 = Read: pin is low; write: clear output bit. 1 = Read: pin is high; write: set output bit.	ext	R/W

Dirección del registro:

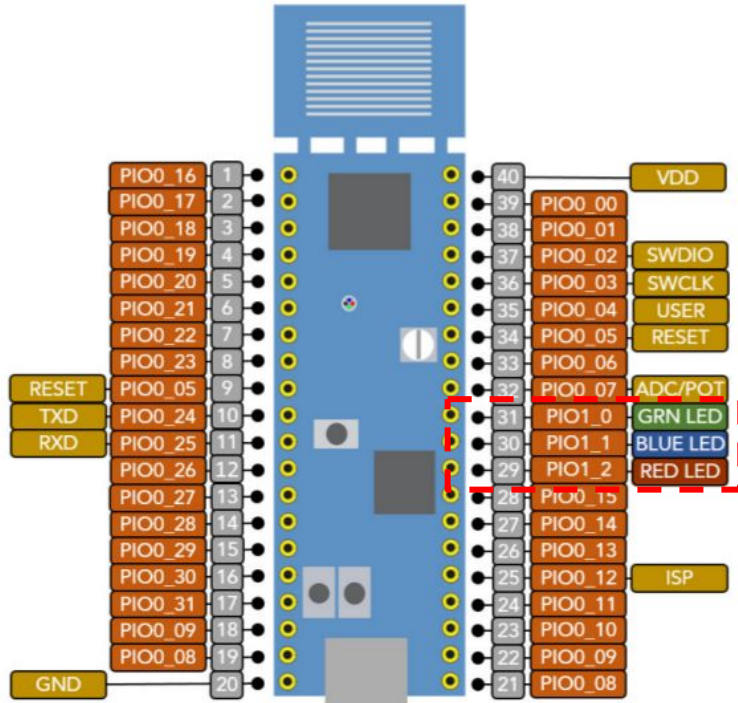
PORT0: 0x A000 2100
PORT1: 0x A000 2104

(4 bytes de diferencia porque son registros de 32 bits)

Función del registro:

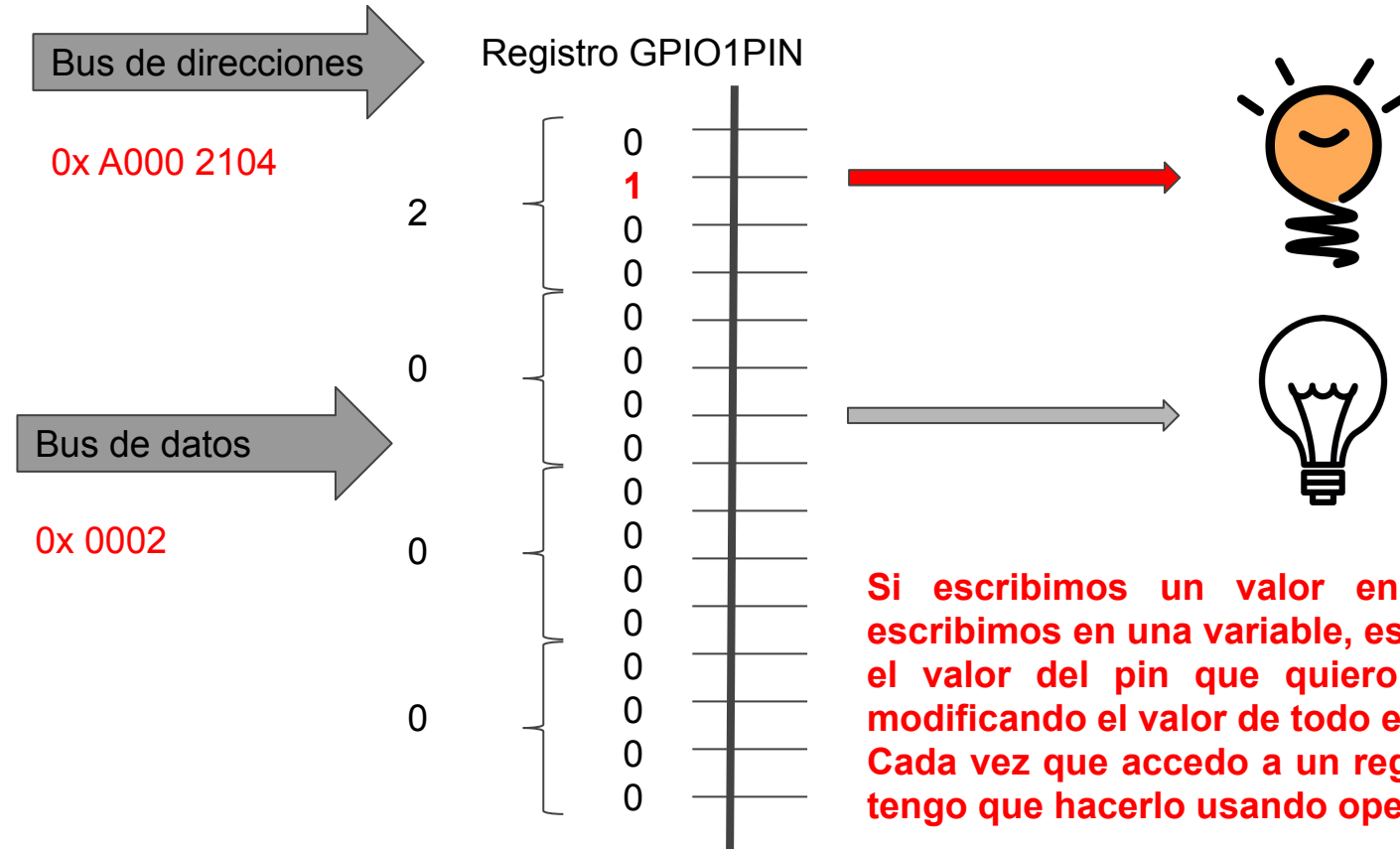
Con 0: "BAJO" el estado del pin (0V)
Con 1: "SUBO" el estado del pin (3,3V)

Del manual de usuario para nuestra placa (UM11181.pdf)



Si queremos, por ejemplo, empezar
prendiendo o apagando el led azul de la
placa, debemos modificar el pin 1 del
puerto 1

Vemos proyecto Registros



Si escribimos un valor en este registro, tal como escribimos en una variable, estamos modificando no solo el valor del pin que quiero modificar, también estoy modificando el valor de todo el resto de los pines!
Cada vez que accedo a un registro para modificar un bit, tengo que hacerlo usando operadores e nivel de bit!