

# **Práctica 2: Sistema de E/S y dispositivos básicos**

---

Proyecto Hardware

# recordatorio práctica 1

- 14 y 15 de octubre corrección
- 22/10 entrega memoria
- Extensión memoria:
  - aprox. 10 hojas
  - 4000 palabras
    - Sin incluir tabla contenidos, portada y anexos necesarios

# Índice

- **Objetivos y descripción de la práctica 2**
- Sistema de E/S de la placa LPC2105
- Ayudas código

# Objetivo: Desarrollar código para un sistema empujado real

- En esta placa el procesador está acompañado de muchos dispositivos, principalmente de entrada/salida
- Vamos a aprender a interaccionar con ellos trabajando en C, pero empleando ensamblador cuando sea necesario
- Depurar un código con varias fuentes de interrupción activas
- Depurar un código en ejecución (no sólo paso a paso)

# ¿Qué tenemos que aprender?

- Entender la configuración de la placa:
  - Registros de configuración de los elementos utilizados
  - Acceso a los registros desde C y desde el entorno
- Gestión del hardware del sistema utilizando C:
  - Utilización de las bibliotecas de la placa
  - Gestión de las interrupciones en C
  - Entender las estructuras que genera el compilador a partir del código fuente (especialmente la pila de programa)

# Descripción de la práctica

- Estudiar y aprender a usar:
  - controlador de interrupciones vectorizadas (VIC)
  - temporizadores (*timers*)
  - GPIO (General-Purpose I/O)
  - botones (emulados con Int. Externas)
  - depurar eventos asíncronos
  - dormir y a despertar al procesador (uso eficiente energía)
  - integrar el juego y MSF
  - estructura modular

# Medir Tiempos

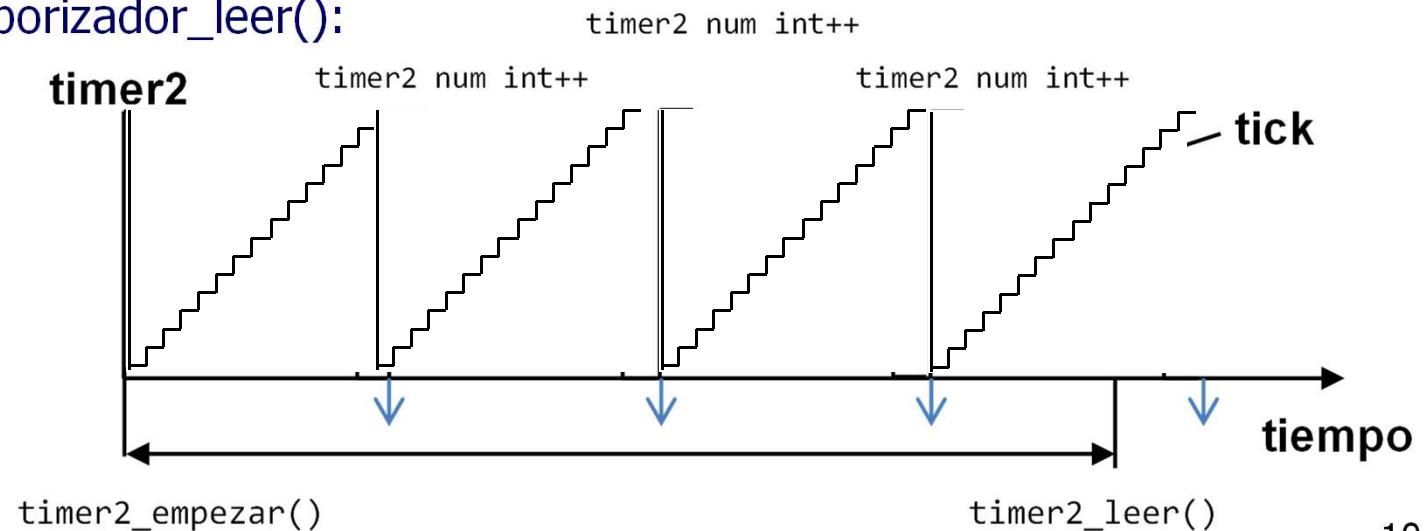
- Utilizar *timer* para medir tiempos

- Precisión y Rango

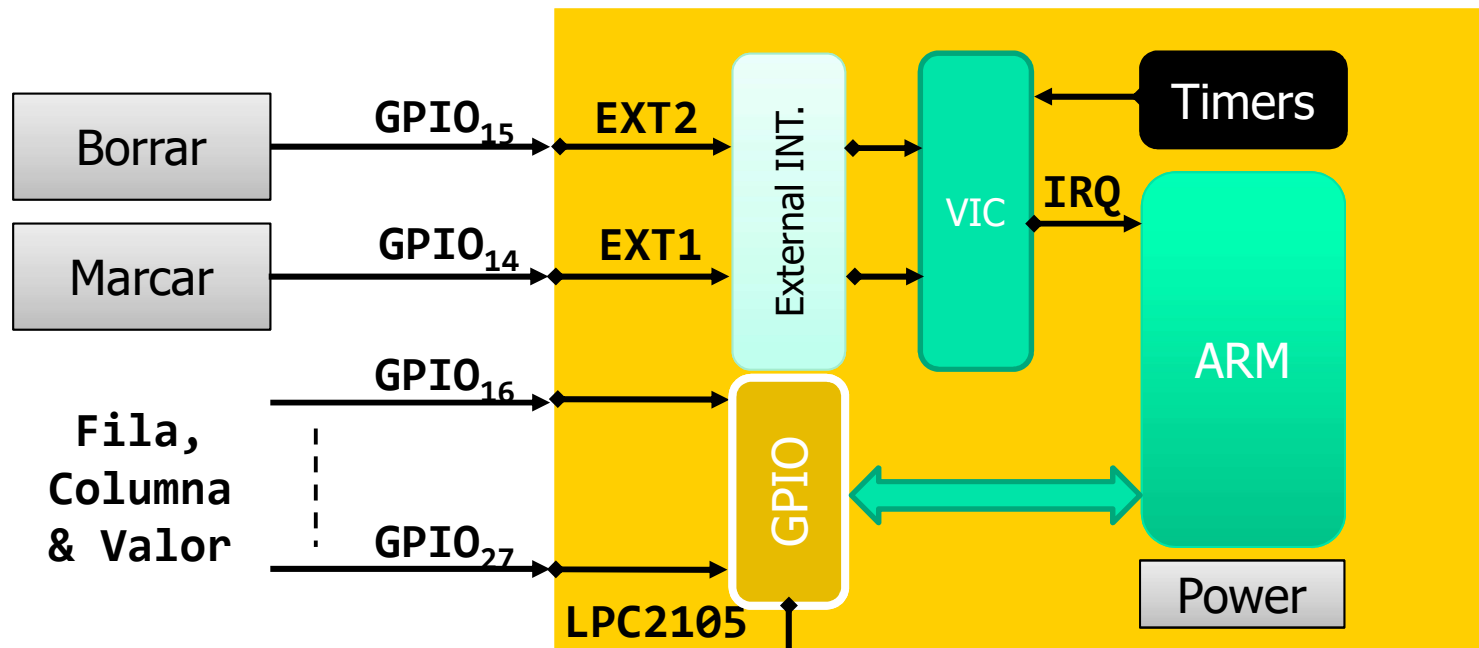
- T. monótono

- temporizador\_iniciar()
    - temporizador\_empezar()
    - temporizador\_leer():

ejemplo



# Sistema E/S



Visualización  
Valor, Candidatos, validación, overflow & latido  
(GPIOs en enunciado)





# Reducir el consumo de energía

- Cuando el procesador no tiene trabajo se queda esperando en un bucle while.
- ¿Qué sentido tiene consumir energía cuando el procesador no tiene nada que hacer?
- En el 2020 el sector TIC fue responsable del 15% de las emisiones de CO<sub>2</sub>!
- **Hay que mejorar la eficiencia energética de nuestros diseños**



# Reducir el consumo de energía

- En lugar de tirar energía ejecutando un bucle de vamos a utilizar los modos de **bajo consumo** del procesador
  - **Objetivo:**
    - reducir el consumo manteniendo el estado
    - Si no hay trabajo entramos en modo **idle** (el procesador para)
    - Si el juego no se usa durante mucho tiempo entramos en **power-down** (el procesador se duerme)

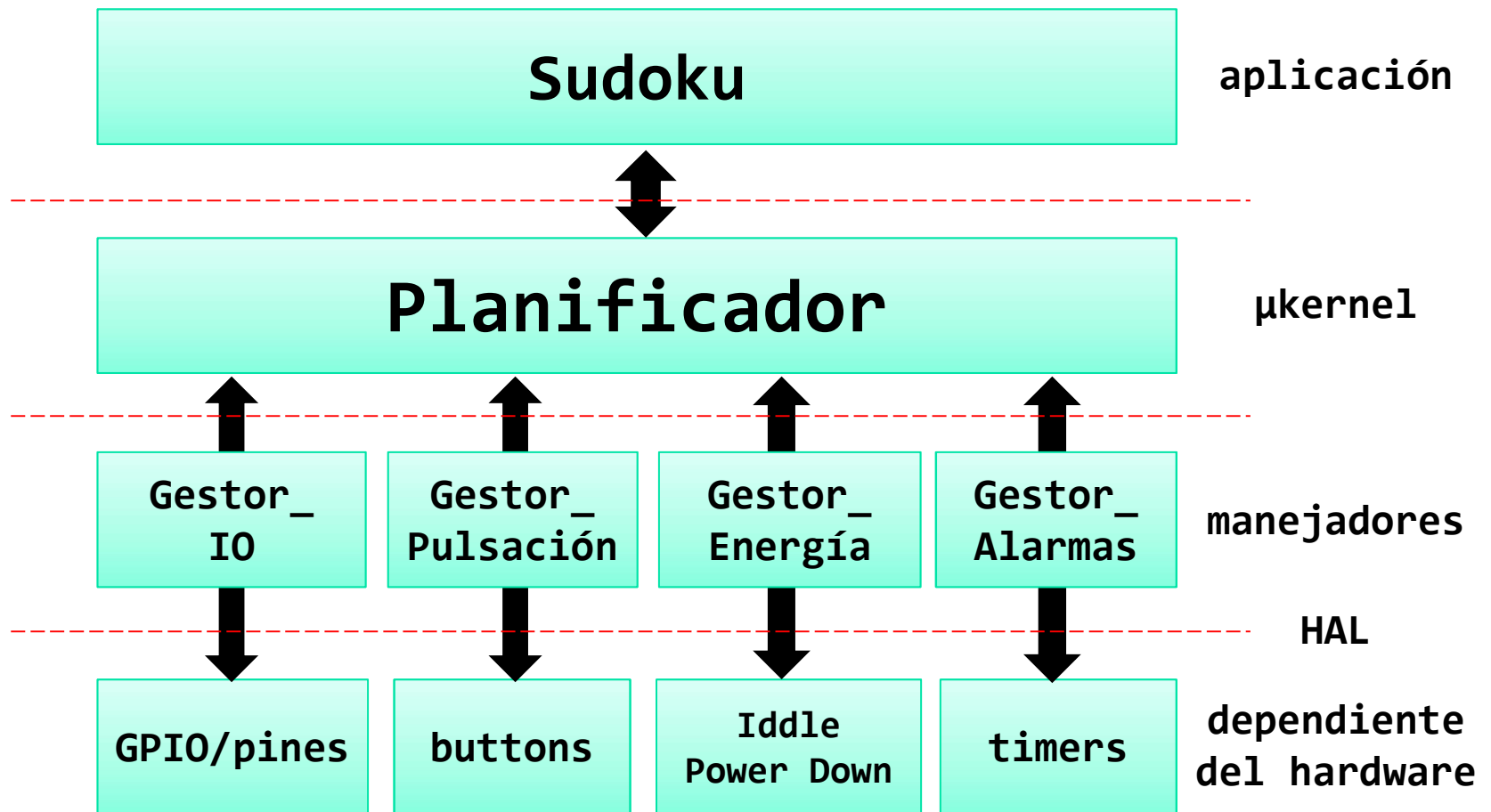
# Descripción de la práctica

- Incluir la iteración con el usuario y la E/S
  - botones
  - temporizadores
  - bajo consumo
  - Se añadirán más cosas en la P3. ¡Es importante que todo sea modular!

# Eventos asíncronos

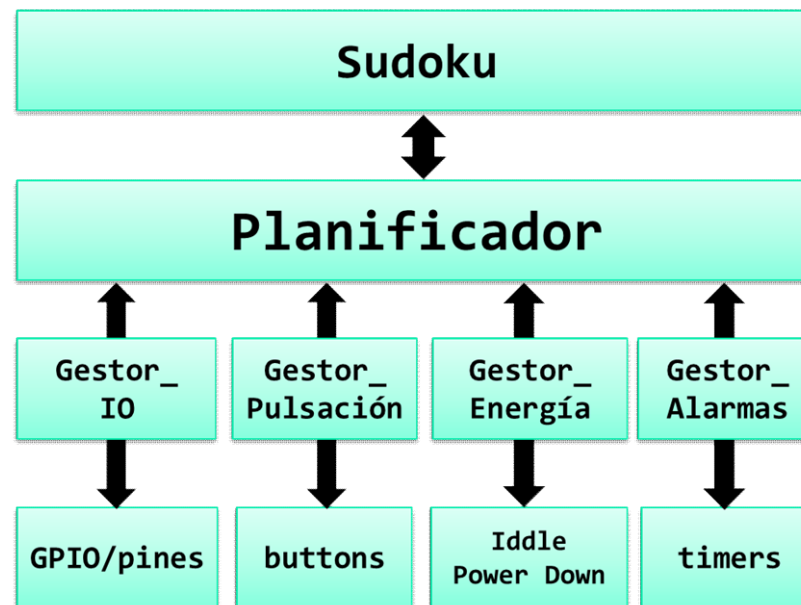
- Desarrollar una cola que almacena los eventos que se generan en el sistema
  - ***cola\_guardar\_evento(uint8\_t ID\_evento, uint32\_t auxData)*** : introduce en la pila dos enteros:
    - Registro con:
      - ID\_evento: se puede usar para identificar los eventos
      - auxData: datos auxiliares aclaratorios
    - time-stamp (del timer con máxima precisión)
  - Nos dice cuándo se han producido los eventos asíncronos que nos interesan  
(por ejemplo, las interrupciones de los botones o timers)
  - Lista circular: no puede salirse de sus límites
  - Definir eventos en eventos.h

# Arquitectura del sistema



# Arquitectura del sistema

- La entrada y salida y los distintos módulos se comunican entre sí generando eventos que se almacenan en la cola de eventos.
- El planificador procesa los eventos en orden. Invocando a las funciones que deben procesarlo.



# Fechas de entrega

- Primera parte: trabajar con el proyecto que os damos hasta paso 5
  - Inicio tercera semana (sesión 7)
  - Aproximadamente del 8 al 12 de Noviembre
- Segunda parte:
  - Aproximadamente el **sesión 9**
  - Los turnos de corrección aparecerán en Moodle

# Material disponible

- En Moodle disponéis de:
  - Un proyecto ejemplo:
  - Documentación original de la placa

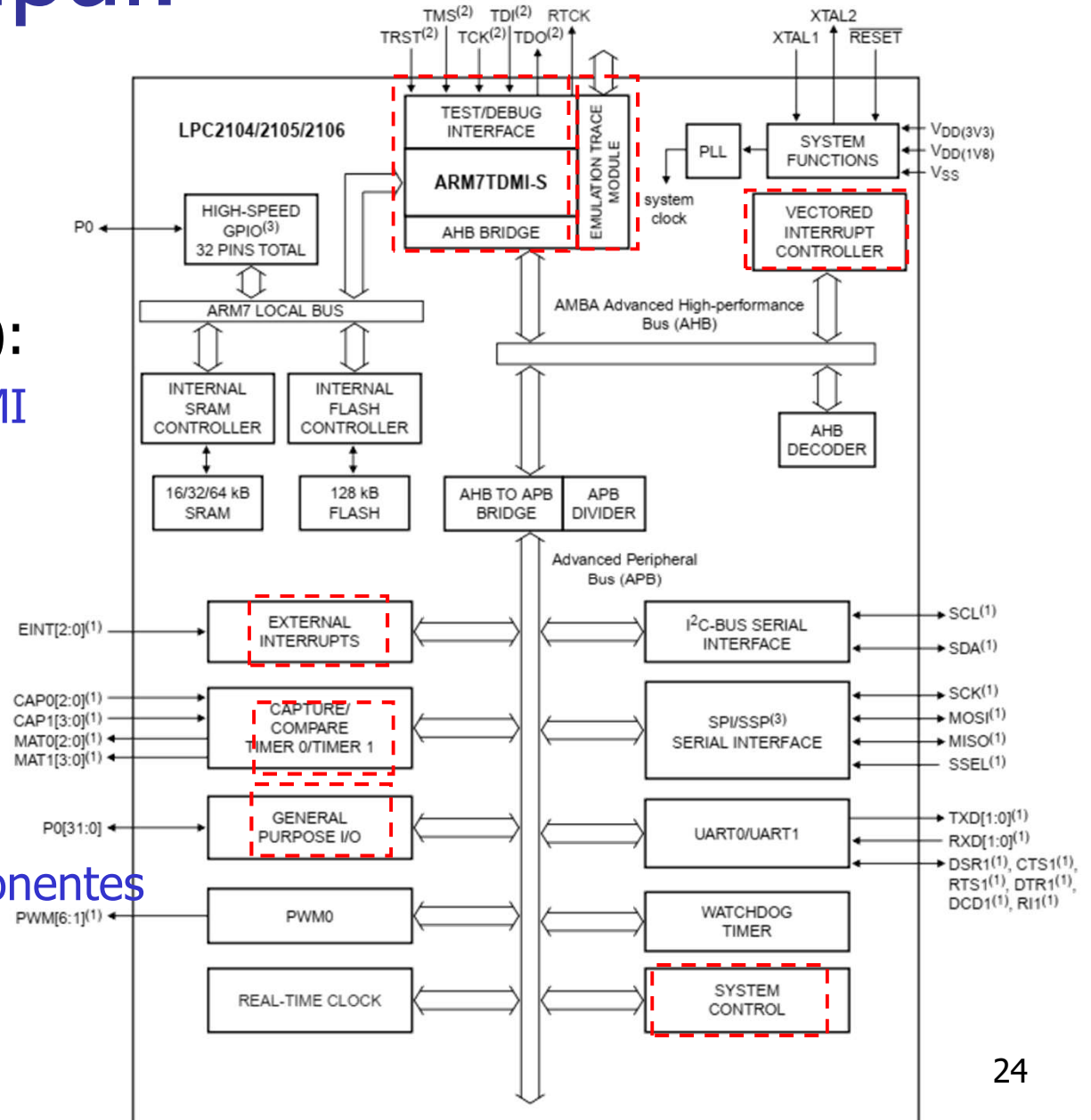


# Índice

- Objetivos y descripción de la práctica
- **Sistema de E/S**
- Ayudas código

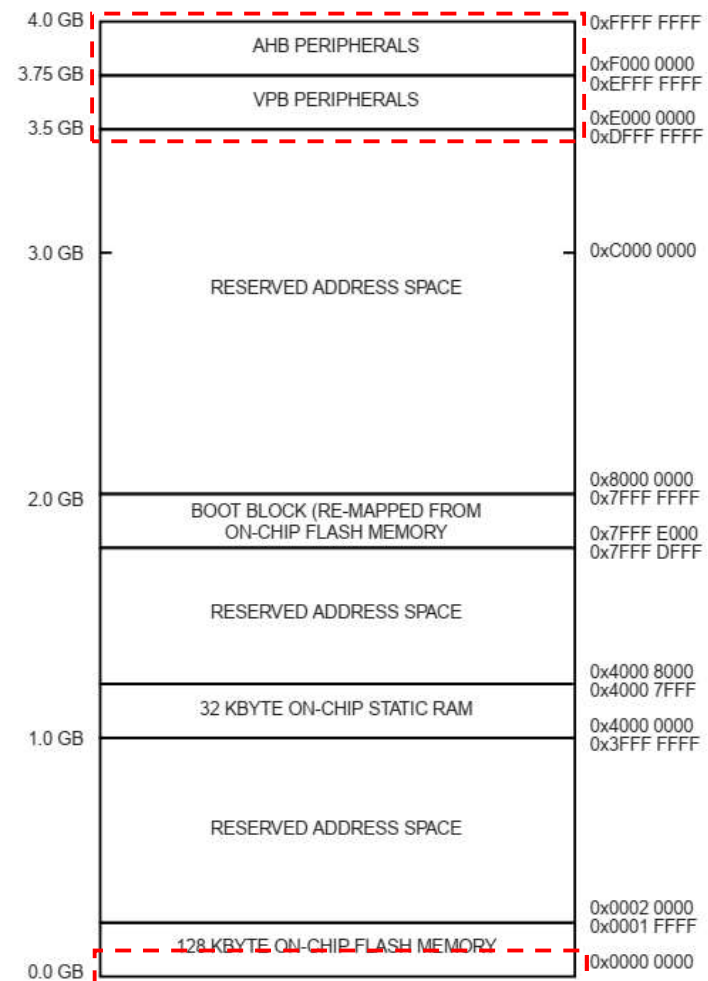
# Chip principal: LPC2105

- System-on-Chip (Soc):
  - Procesador ARM7TDMI
  - Controladores y E/S
    - GPIO
    - UART
    - Timers
    - Power control
  - Controlador INT. Vectorizadas
  - Y muchos más componentes



# Espacio de direcciones del LPC2105

- Direcciones de 32 bits (4GB)
- Espacio dividido zonas
  - 128 kB Flash memory system
    - Código y datos
  - 32 kB RAM estática
    - Accesible como 8, 16 o 32 bits
  - **Vector Interrupciones**
  - Registros E/S



# Sistema de E/S del S3C44B0X

- Puertos de E/S
  - E/S mapeada en memoria (ARM7TDMI):
    - Los registros que controlan a los distintos elementos de la placa tienen asignados una dirección de memoria
    - La entrada/salida se gestiona escribiendo/leyendo en esas direcciones
  - Ocupa el último tramo [0xE000\_0000-0xFFFF\_FFFF]
    - AHB tienen reservado el último tramo [0xFFE0\_0000 - 0xFFFF\_FFFF]
    - El resto de dispositivos periféricos suelen estar ubicados en el APB [0xE000\_0000 - 0xE01F\_FFFF]

@E/S

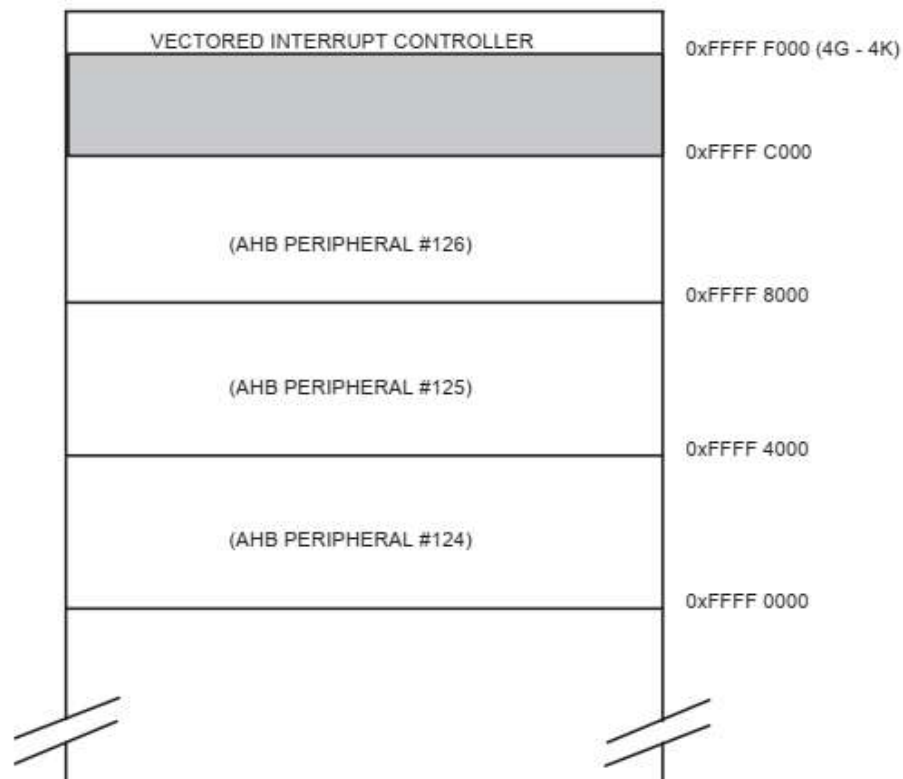


Table 3. APB peripherals and base addresses

APB peripheral	Base address	Peripheral name
0	0xE000 0000	Watchdog timer
1	0xE000 4000	Timer 0
2	0xE000 8000	Timer 1
3	0xE000 C000	UART0
4	0xE001 0000	UART1
5	0xE001 4000	PWM
6	0xE001 8000	Not used
7	0xE001 C000	I <sup>2</sup> C
8	0xE002 0000	SPI
9	0xE002 4000	RTC
10	0xE002 8000	GPIO
11	0xE002 C000	Pin connect block
12	0xE003 0000	Not used
13	0xE003 4000	Not used
14 - 22	0xE003 8000 - 0xE005 8000	Not used
23	0xE005 C000	SSP
24	0xE006 0000	Not used
25	0xE006 4000	Not used
26	0xE006 8000	Not used
27	0xE006 C000	Not used
28	0xE007 0000	Not used
29	0xE007 4000	Not used
30 - 126	0xE007 8000 - 0xE01F 8000	Not used
127	0xE01F C000	System Control Block

# LPC210x.H

- Fichero cabecera del fabricante Macros para usar nombres simbólicos
  - Controlador de Interrupciones vectorizadas
  - GPIO
  - Perifericos

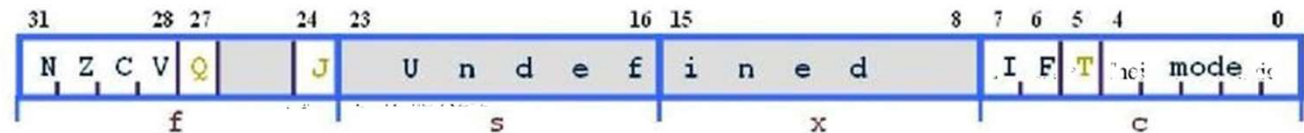
```
#ifndef __LPC210x_H
#define __LPC210x_H

/* Vectored Interrupt Controller (VIC) */
#define VICIRQStatus    (*((volatile unsigned long *) 0xFFFFF000))
#define VICFIQStatus    (*((volatile unsigned long *) 0xFFFFF004))
...
/* General Purpose Input/Output (GPIO) */
#define IOPIN            (*((volatile unsigned long *) 0xE0028000))
#define IOSET            (*((volatile unsigned long *) 0xE0028004))
```

# Interrupciones

- FIQ
- IRQ
- Modos del procesador

■ CPRS



I = 1 disables IRQ

F = 1 disables FIQ

Table A2-2 The mode bits

M[4:0]	Mode	Accessible registers
0b10000	User	PC, R14 to R0, CPSR
0b10001	FIQ	PC, R14_fiq to R8_fiq, R7 to R0, CPSR, SPSR_fiq
0b10010	IRQ	PC, R14_irq, R13_irq, R12 to R0, CPSR, SPSR_irq
0b10011	Supervisor	PC, R14_svc, R13_svc, R12 to R0, CPSR, SPSR_svc
0b10111	Abort	PC, R14_abt, R13_abt, R12 to R0, CPSR, SPSR_abt
0b11011	Undefined	PC, R14_und, R13_und, R12 to R0, CPSR, SPSR_und
0b11111	System	PC, R14 to R0, CPSR (ARMv4 and above)

- Reset

# Registros y modos

User	FIQ	IRQ	SVC	Undef	Abort
r0	User mode r0-r7, r15, and cpsr	User mode r0-r12, r15, and cpsr	User mode r0-r12, r15, and cpsr	User mode r0-r12, r15, and cpsr	User mode r0-r12, r15, and cpsr
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8	r8				
r9	r9				
r10	r10				
r11	r11				
r12	r12				
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
r15 (pc)					
cpsr					
	spsr	spsr	spsr	spsr	spsr



# VIC – Vectored Int. Controller

- Cap. 5
- 32 Int request
  - FIQ
  - 16 Vect. IRQ
  - Non Vect. IRQ
- Prioridades

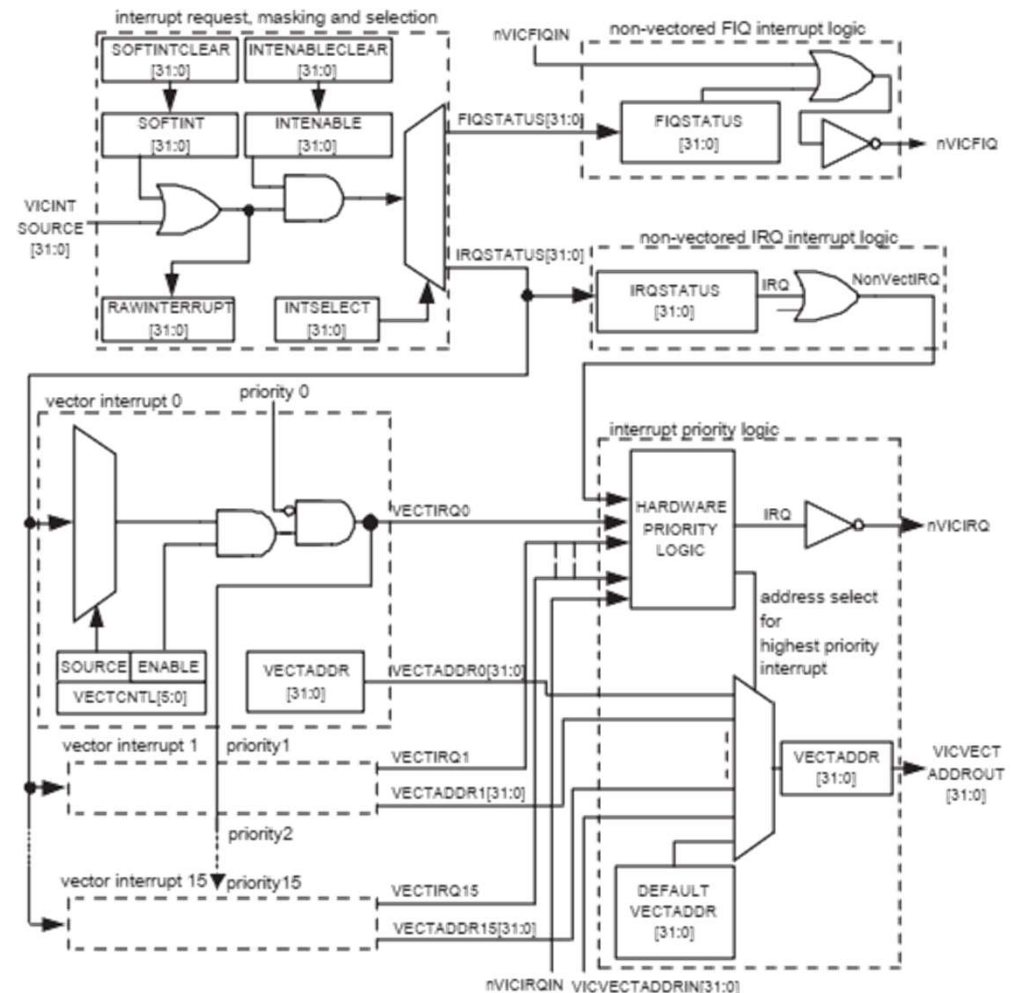


Fig 13. Block diagram of the Vectored Interrupt Controller

# VIC

## ■ Registros

- VICIntSelec
- VICIntEnable
- VICIntEnCl
- VICVectAddr
- VICDefVectAddr
- VICVectAddr<sub>x</sub>
- VICVectCntl<sub>x</sub>
- ...

Vectored Interrupt Controller (VIC)

Channel	Source	Name	Type	Vector	IntEnable	RawInt
0	Watchdog	WDINT	IRQ	00000000H	0	0
1	SW Interrupt		IRQ	00000000H	0	0
2	DbgCommRx		IRQ	00000000H	0	0
3	DbgCommTx		IRQ	00000000H	0	1
4	Timer 0		IRQ	00000000H	0	0
5	Timer 1		IRQ	00000000H	0	0
6	UART0		IRQ	00000000H	0	0
7	UART1		IRQ	00000000H	0	0
8	PWM		IRQ	00000000H	0	0
9	I2C		IRQ	00000000H	0	0
10	SPI		IRQ	00000000H	0	0
12	PLL Lock	PLOCK	IRQ	00000000H	0	1

Selected Interrupt: Watchdog

☐ IntEnable ☐ IntSelect ☐ SoftInt ☐ RawInt

IRQ Slot: --

VICDefVectAddr: 0x00000000

VICVectAddr: 0x00000000 VICIntSelect: 0x00000000 VICRawIntr: 0x00001008

VICSoftInt: 0x00000000 VICIntEnable: 0x00000000 VICIRQStatus: 0x00000000

VICSoftIntClear: 0x00000000 VICIntEnClr: 0x00000000 VICFIQStatus: 0x00000000

VICProtection: 0x00000000

# Tabla Vectores

; Exception Vectors ; Mapped to Address 0.  
 ; Absolute addressing mode must be used.  
 ; Dummy Handlers are implemented as infinite loops which can be modified.

Vectors	LDR	PC, Reset_Addr	
	LDR	PC, Undef_Addr	
	LDR	PC, SWI_Addr	
	LDR	PC, PAbt_Addr	
	LDR	PC, DAbt_Addr	
	NOP		; Reserved Vector
;	LDR	PC, IRQ_Addr	
	LDR	PC, [PC, #-0x0FF0]	; Vector from VicVectAddr
	LDR	PC, FIQ_Addr	
Reset_Addr	DCD	Reset_Handler	
Undef_Addr	DCD	Undef_Handler	
SWI_Addr	DCD	SWI_Handler	
PAbt_Addr	DCD	PAbt_Handler	
DAbt_Addr	DCD	DAbt_Handler	
	DCD	0	; Reserved Address
IRQ_Addr	DCD	IRQ_Handler	
FIQ_Addr	DCD	FIQ_Handler	

# Controlador de interrupciones

- Funcionamiento simplificado int. vectorizadas:
  - VIC provoca la señal IRQ al micro
  - VIC provee la @ de la RSI (de la más prioritaria o default)
  - Se ejecuta RSI con I inhabilitadas.
  - Se avisa al VIC que se ha acabado la RSI de esa prioridad
  - End Of I
- Cambio de modo, cspr .... PC....
- ¿Anidadas?

## Diapositiva 34

---

J4

opcional

Javi; 06/10/2020

# Controlador de interrupciones

## ■ Código de ejemplo para int. vectorizadas:

### Inicialización

```
VICVectAddr0 = (unsigned long)F_RSI;//set int. vector in 0
...
VICIntSelect = ...; //tipo interrupción, IRQ (0) o FIQ (1)
VICVectCntl0 = 0x20 | 4; //ejemplo... IRQ slot enable + #
VICIntEnable = VICIntEnable | 0x00000010;//Enable Int
```

### F\_RSI

```
Atiende al periférico
Limpia flag de solicitud de Int. periférico
VICVectAddr = 0; //Reconoce Interrupción (prioridades)
Retorna EOI
```

# Declaración de funciones para gestionar interrupciones en C

- Keil dispone de un atributo de función para especificar su uso como gestor de interrupciones
  - `void f(void) __irq;`
- ¿Cuál sera la diferencia del bloque de activación en estas funciones?

# Recordatorio: Marco de pila

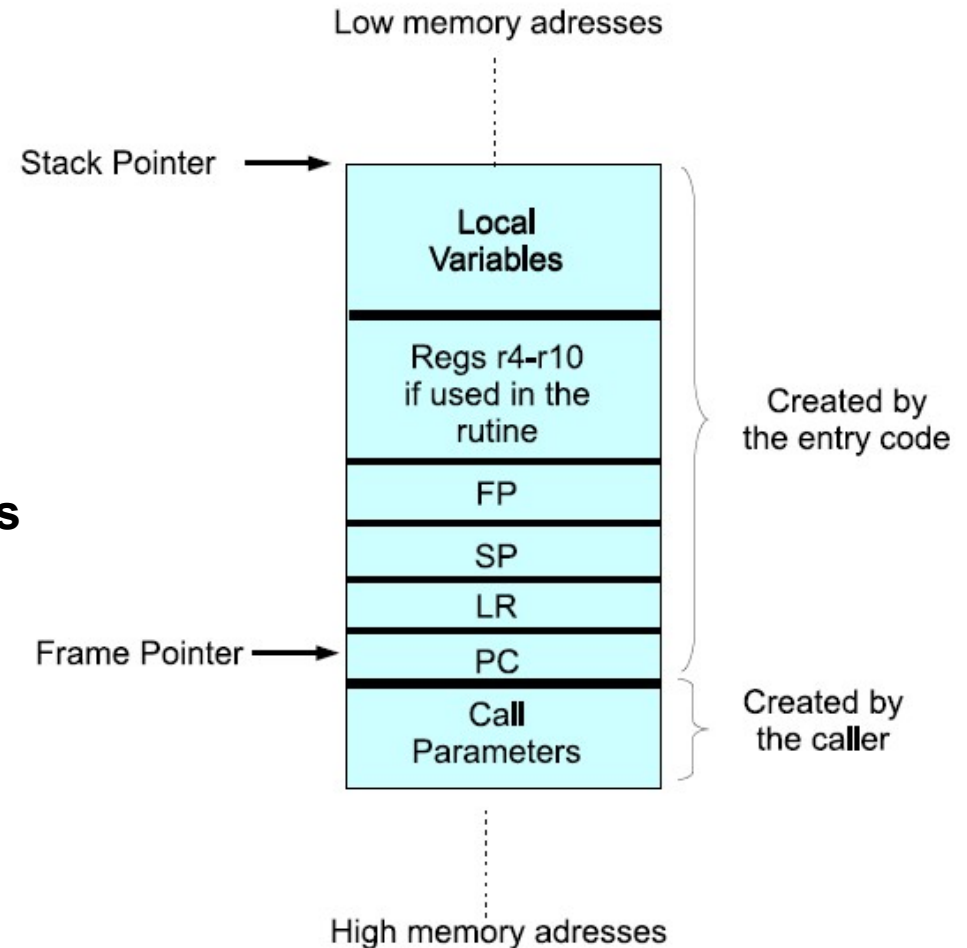
- Estructura de una rutina:

- **Prólogo**

```
MOV    IP, SP
STMDB  SP!, {r4-r10,FP,IP,LR,PC}
SUB     FP, IP, #4
SUB     SP, #SpaceForLocalVariables
```

- **Epílogo:**

```
LDMDB  FP, {r4-r10,FP,SP,PC}
```





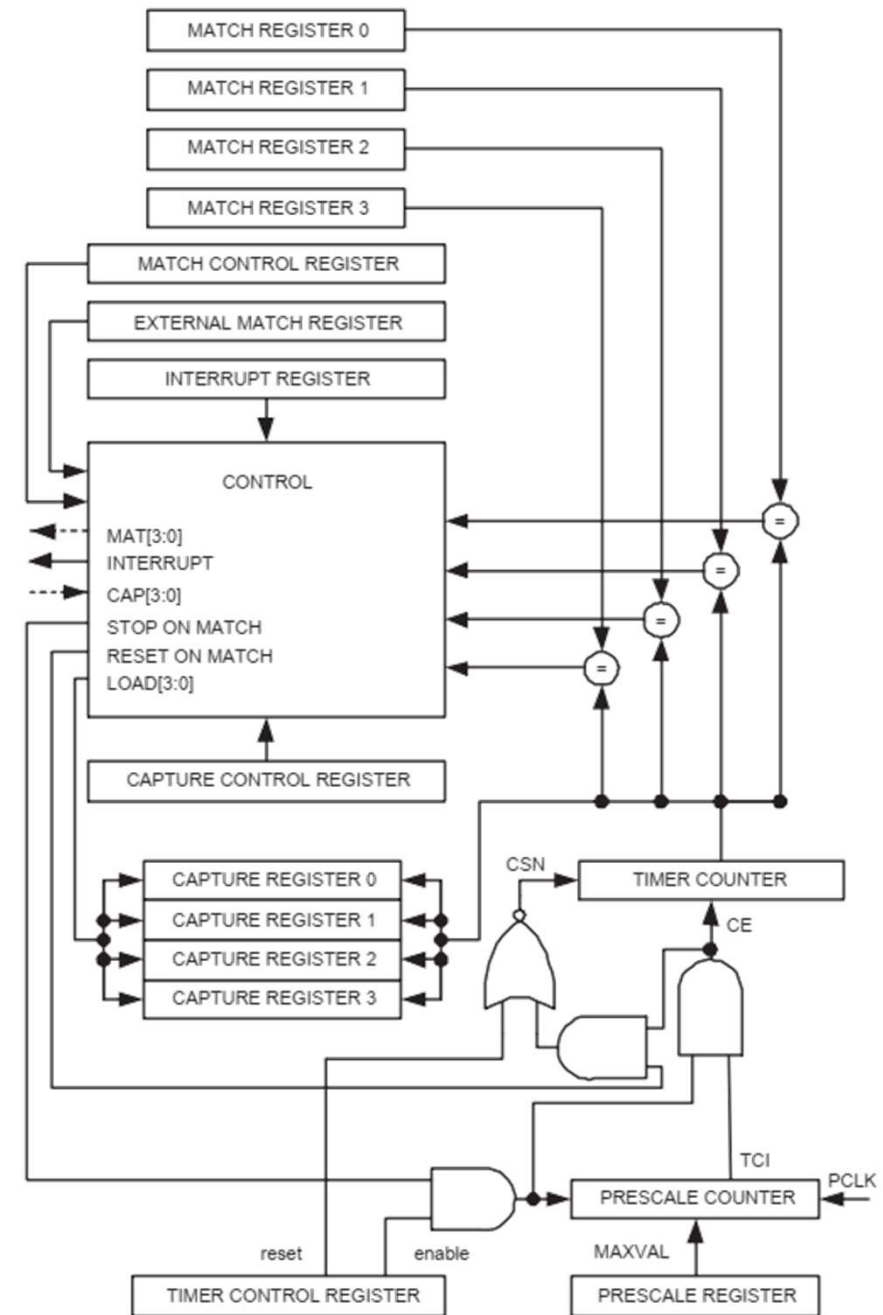
# Marco de pila para excepciones

- La gestión es muy parecida:
- Prólogos y epílogos casi idénticos al anterior
- Diferencias:
  - Antes de guardar LR hay que restarle 4 (8 para las excepciones abort) **¿Por qué?**
  - Hay que guardar todos los registros: **también IP y r0-r3!**
  - Al entrar y al salir el procesador cambia de modo:
    - En la entrada el procesador cambia de forma automática en función del tipo de excepción
    - Al terminar para volver al modo anterior hay que añadir “^” a la instrucción de retorno

# Notificación

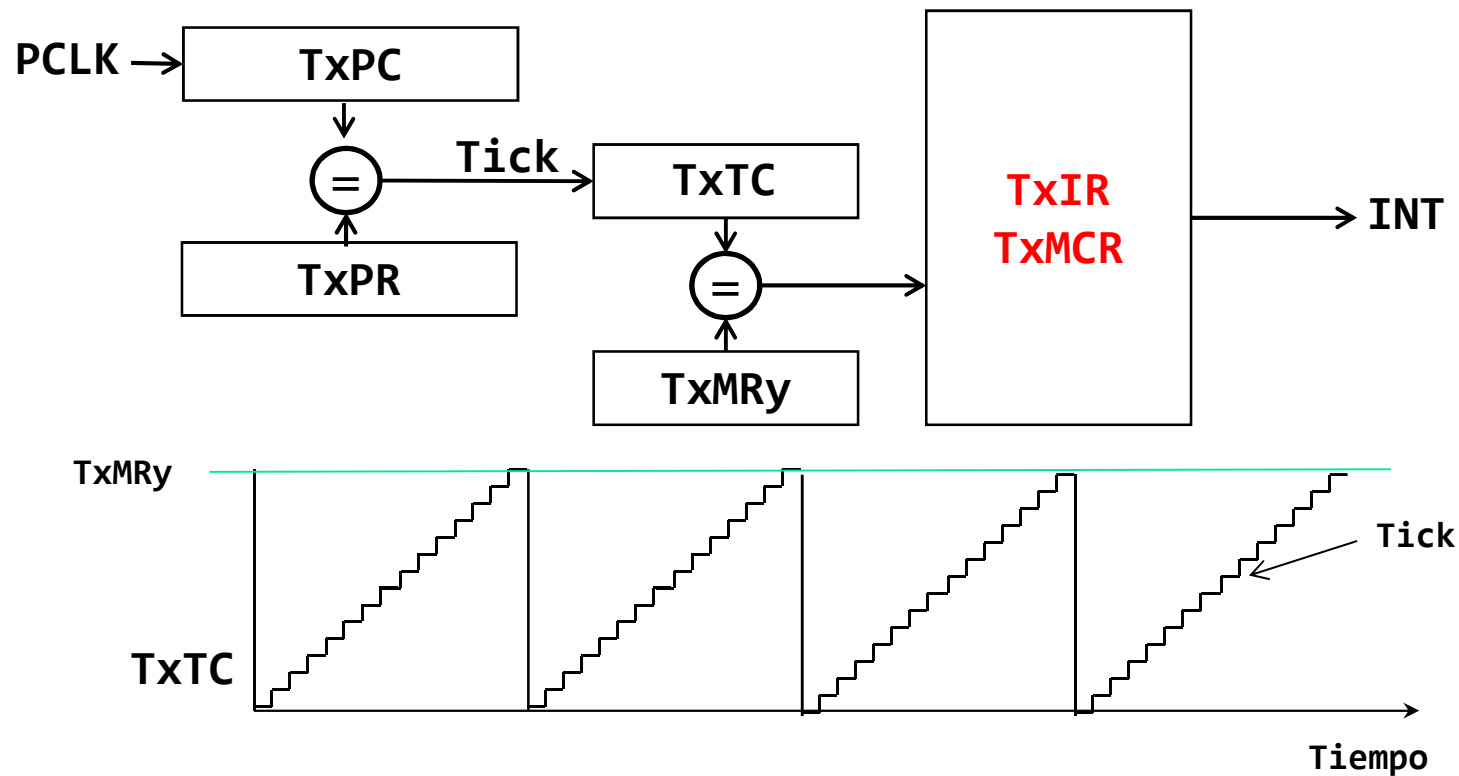
- RSI y programa principal necesitan comunicarse
    - Variable compartida + función de acceso
      - visible desde las dos funciones del mismo modulo
      - cambia fuera flujo normal de ejecución
- ```
static volatile uint32_t var_compartida;
```
- Apilando evento en planificador
  - Callback

- 2 timers/counter
- 32 bits
- preescalado 32 bits
- 4 match
  - contínua + IRQ
  - stop + IRQ
  - reset + IRQ
- Interval Timer
  - PCLK (¿3Mhz?)



# Temporizadores

- Comportamiento básico



# Timer

- TxIR: Interrupt Register
  - (rw: clear, rd:id de las 8 posibles causas [MR0,MR1,..])
- TxTCR: Time Control Register
  - bit 0: enable (1) / disable, bit 1: reset counters (1)
- TxTC: Timer Counter
- TxPR: Prescale Register
- TxPC: Prescale Counter
- TxMCR: Match Control Register
- TxMR<sub>y</sub>: Match Register 0..3
- TxTCR: Count Control Register (00 - Timer Mode)
- TxCCR ... TxCR<sub>y</sub> ...

# GPIO

- 32 GPIO (1 puerto)
  - P0.0-P0.31
- Selección
- Dirección
- Bit-level set & clear

Table 54. Pin description ...continued

| Symbol             | Pin | Type | Description                                          |
|--------------------|-----|------|------------------------------------------------------|
| P0.16/EINT0/MAT0.2 | 46  | I/O  | <b>P0.16</b> — Port 0 bit 16.                        |
|                    |     | I    | <b>EINT0</b> — External interrupt 0 input.           |
|                    |     | O    | <b>MAT0.2</b> — Match output for Timer 0, channel 2. |

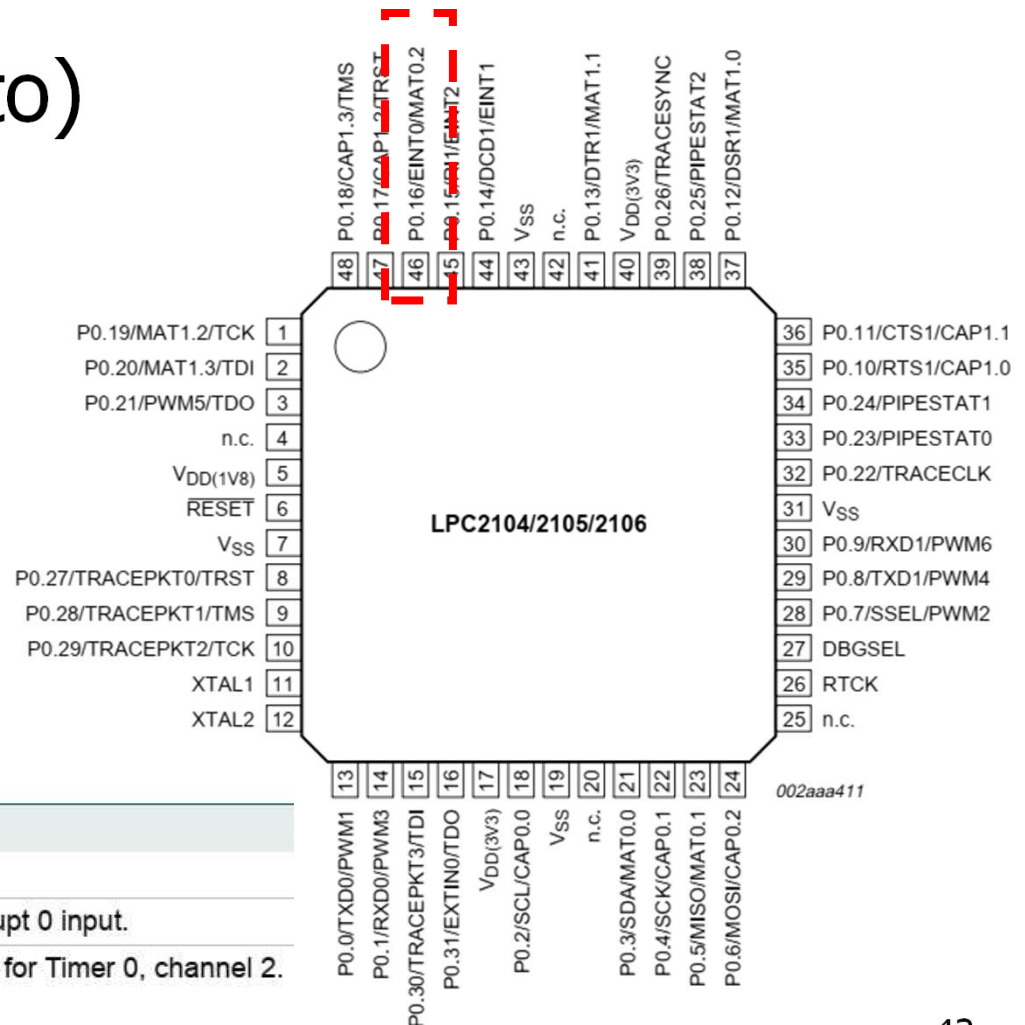
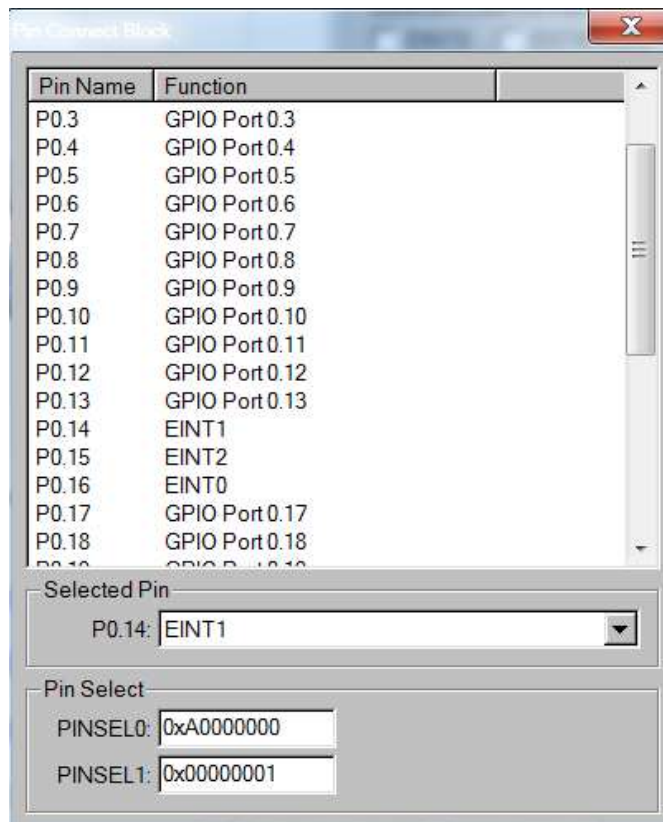


Fig 14. LPC2104/05/06 LQFP48 pin package

# GPIO

- Asignar PINS a un dispositivo
  - cap 6 y 7.



**Table 55. Pin connect block register map**

| Name    | Description                     | Access     | Reset value <sup>[1]</sup> | Address     |
|---------|---------------------------------|------------|----------------------------|-------------|
| PINSEL0 | Pin function select register 0. | Read/Write | 0x0000 0000                | 0xE002 C000 |
| PINSEL1 | Pin function select register 1. | Read/Write | 0x0000 0000                | 0xE002 C004 |

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

**Table 57. Pin function select register 1 (PINSEL1 - 0xE002 C004)**

| PINSEL1 | Pin Name | Value | Function            | Value after reset |
|---------|----------|-------|---------------------|-------------------|
| 1:0     | P0.16    | 0     | GPIO Port 0.16      | 0                 |
|         |          | 0     | EINT0               |                   |
|         |          | 1     | Match 0.2 (Timer 0) |                   |

- P0xDIR
- P0xVAL
- P0xSET
- P0xCLR

**Table 60. GPIO register map (legacy APB accessible registers)**

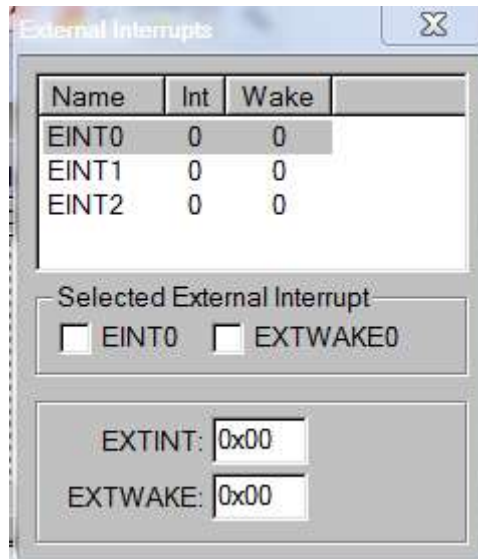
[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

**Table 62. GPIO port 0 Direction register (IO0DIR - address 0xE002 8008) bit description**

| Bit  | Symbol | Value | Description                                                                      | Reset value |
|------|--------|-------|----------------------------------------------------------------------------------|-------------|
| 31:0 | P0xDIR |       | Slow GPIO Direction control bits. Bit 0 controls P0.0 ... bit 30 controls P0.30. | 0x0000 0000 |
|      |        | 0     | Controlled pin is input.                                                         |             |
|      |        | 1     | Controlled pin is output.                                                        |             |



# EXT INT



External Interrupts

| Name  | Int | Wake |
|-------|-----|------|
| EINT0 | 0   | 0    |
| EINT1 | 0   | 0    |
| EINT2 | 0   | 0    |

Selected External Interrupt

☐ EINT0 ☐ EXTWAKE0

EXTINT: 0x00

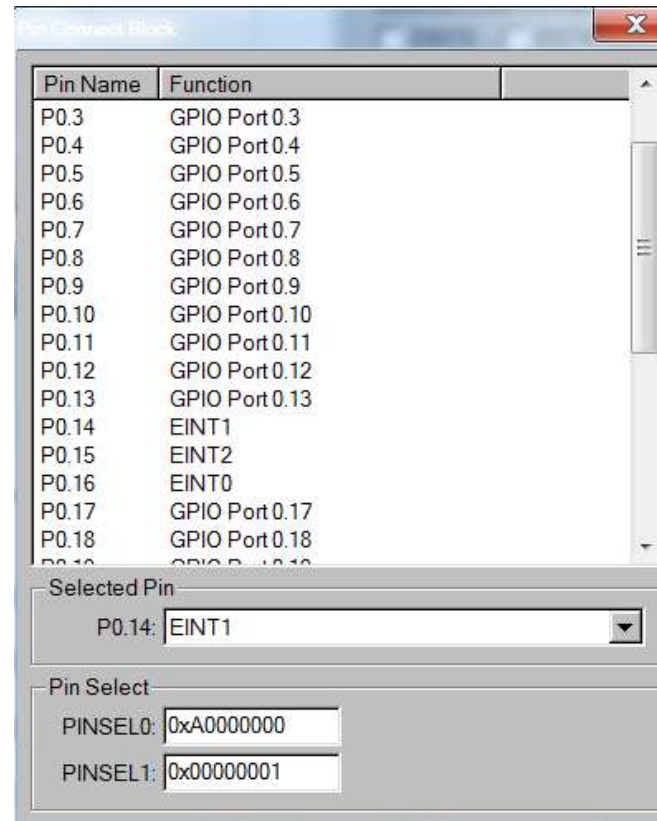
EXTWAKE: 0x00



Toolbox

Update Windows

|   |      |
|---|------|
| 1 | Play |
| 2 | Pass |
| 3 | F0   |
| 4 | F1   |
| 5 | ...  |
| 6 | C7   |



Pin Connect Block

| Pin Name | Function       |
|----------|----------------|
| P0.3     | GPIO Port 0.3  |
| P0.4     | GPIO Port 0.4  |
| P0.5     | GPIO Port 0.5  |
| P0.6     | GPIO Port 0.6  |
| P0.7     | GPIO Port 0.7  |
| P0.8     | GPIO Port 0.8  |
| P0.9     | GPIO Port 0.9  |
| P0.10    | GPIO Port 0.10 |
| P0.11    | GPIO Port 0.11 |
| P0.12    | GPIO Port 0.12 |
| P0.13    | GPIO Port 0.13 |
| P0.14    | EINT1          |
| P0.15    | EINT2          |
| P0.16    | EINT0          |
| P0.17    | GPIO Port 0.17 |
| P0.18    | GPIO Port 0.18 |

Selected Pin

P0.14: EINT1

Pin Select

PINSEL0: 0xA0000000

PINSEL1: 0x00000001

# EXT INT

- Cap3, 6.1
  - permiten despertar al procesador desde power-down

EXTMODE y EXTPOLAR no están en el sistema: las interrupciones se activan por nivel con un '0'

Table 9. External interrupt registers

| Name     | Description                                                                                                                                                                                              | Access | Reset value <sup>[1]</sup> | Address     |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|----------------------------|-------------|
| EXTINT   | The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, EINT2. See <a href="#">Table 3-10</a> .                                                                                  | R/W    | 0                          | 0xE01F C140 |
| EXTWAKE  | The External Interrupt Wakeup Register contains four enable bits that control whether each external interrupt will cause the processor to wake up from Power-down mode. See <a href="#">Table 3-11</a> . | R/W    | 0                          | 0xE01F C144 |
| EXTMODE  | The External Interrupt Mode Register controls whether each pin is edge- or level sensitive.                                                                                                              | R/W    | 0                          | 0xE01F C148 |
| EXTPOLAR | The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt.                                                                                               | R/W    | 0                          | 0xE01F C14C |

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

# Power

## ■ 10.1

**Table 25. Power control registers**

| Name  | Description                                                                                                                                                                                                        | Access | Reset value <sup>[1]</sup> | Address     |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|----------------------------|-------------|
| PCON  | Power Control Register. This register contains control bits that enable the two reduced power operating modes of the microcontroller. See <a href="#">Table 3–26</a> .                                             | R/W    | 0x00                       | 0xE01F C0C0 |
| PCONP | Power Control for Peripherals Register. This register contains control bits that enable and disable individual peripheral functions, allowing elimination of power consumption by peripherals that are not needed. | R/W    | 0x0000 1FBE                | 0xE01F C0C4 |

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

# Índice

- Objetivos y descripción de la práctica
- Sistema de E/S
- **Ayudas código**

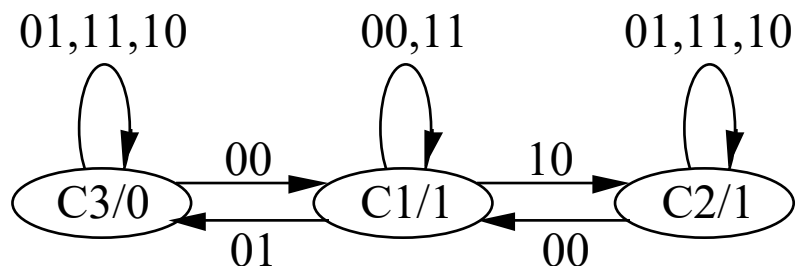
# Autómatas. Implementación

- Ej.: Detección sentido contrario

- **MOORE**

- Entradas nivel muestreadas (síncronas)

- Salidas asíncronas

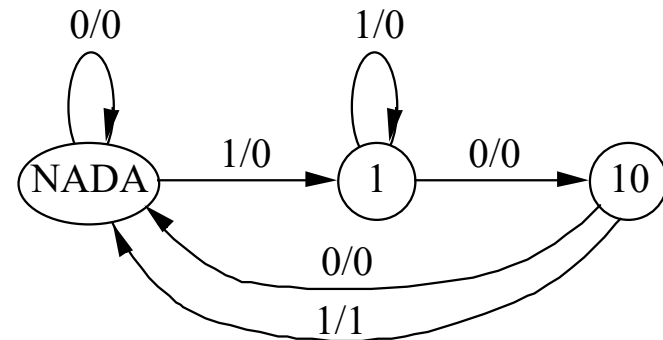


```
// Estado pertenece al conjunto de estados
Entrada = Leer_Entrada ();
switch (Estado)
{
    case C1 : Salida(NO_ALARMA) ;
        switch (Entrada) {
            case I01 : Estado = C3 ; break ;
            case I10 : Estado = C2 ; break ;
            default : }
        break ;
    case C2 : Salida (NO_ALARMA) ;
        if (Entrada == I00) Estado = C1 ;
        break ;
    case C3 : Salida (ALARMA) ;
        if (Entrada == I00) Estado = C1 ;
        break ;
}
```

# Autómatas. Implementación

```
Espera_Sincronismo () ;  
Entrada = Leer_Bit () ;  
switch (Estado)  
{  
  case NADA : if (Entrada==0) {Salida=0; Estado=NADA;}  
              else if (Entrada==1) {Salida=0; Estado=E1;}  
              break ;  
  case E1 :   if (Entrada==0) {Salida=0; Estado=E10;}  
              else if (Entrada==1) {Salida=0; Estado=E1;}  
              break ;  
  case E10 :  if (Entrada==0) {Salida=0; Estado=NADA;}  
              else if (Entrada==1) {Salida=1; Estado=E101;}  
              break ;  
  case E101 : if (Entrada==0) {Salida=0; Estado=NADA;}  
              else if (Entrada==1) {Salida=0; Estado=E1;}  
              break ;  
}  
Genera (Salida) ;
```

Ej: reconocedor de cadenas: **MEALY**



*Podéis encontrar una forma más eficiente de implementar MSF en:  
<http://johnsantic.com/comp/state.html>*

# Compilación condicional

- En C es posible utilizar directivas del preprocesador para compilar condicionalmente código
- Habréis visto que los ficheros de cabecera suelen incluir una guarda para no tener problemas de doble inclusión

```
#ifndef FICHERO1_H  
#define FICHERO1_H  
int var;  
#endif
```

# Compilación condicional

- Si fichero2.h incluye a fichero1.h y fichero\_fuente.c incluye a ambos, sin la guarda tendremos dos definiciones de la variable var
- Otro posible uso es escribir múltiples implementaciones de una misma función.



# Ejemplo compilación condicional

```
#define USAR_EMULACION

int escribir_pantalla(int n, int pos_x, pos_y) {
#ifdef USAR_EMULACION
    printf("x=%d, y=%d, n=%d\n", x, y, n);
#else
    // utilizar una zona de memoria como pantalla
    pantalla_mem[x * NUM_COL + y] = itoa(n);
#endif
}
```

Si antes de compilar comentamos la definición, el código ejecutara la rama del `#else` y viceversa

# Variables en C

- visibilidad, alcance y cualificadores
  - global, local
  - const (#define, enum)
  - volatile
  - extern
  - static
    - en globales
    - en locales
    - en funciones

# Volatile

- `volatile` es una palabra reservada que indica que una variable puede ser modificada **concurrentemente** por un operador externo aunque parezca que desde el código no se modifique
- Ejemplo de variables que necesiten ser definidas como volátiles son todos los registros de E/S de periféricos o las variables compartidas con *rsi*.

# Volatile: Ejemplo de uso

- Temporizador que espera 10 ticks de reloj para imprimir por pantalla

```
volatile uint32_t ticks = 0;  
while( ticks < 10);  
printf("10 ticks\n");
```

- Sin volatile un compilador podría asumir que es un bucle infinito y optimizarlo como while(1);

# Static

- `static` es una palabra reservada en C que indica que la duración de la variable es la duración del programa y que la variable o función sólo es visible en su unidad de compilación (fichero) (internal linkage).
- Restringir el alcance y visibilidad de variables y funciones reduce errores y toda función que sólo deba emplearse en el fichero actual deberá estar definida como `static`.

# Static: Ejemplo de uso

reloj.c

```
static unsigned long eventos = 0;  
void reloj_eventos() { eventos++; ... }
```

teclado.c

```
static unsigned long eventos = 0;  
void teclado_eventos() { eventos++; }
```

- El enlazador no tendrá problemas con ambos ficheros objetos y sabrá que tiene que definir 2 variables globales distintas en la sección de datos del ejecutable

# static

- Variables locales

- almacenar estado entre ejecuciones

- Visibilidad local (solo dentro de la función o módulo)

- Tiempo de vida “global”  
(durante toda la ejecución del programa

- (se guardan en zona de datos, no en pila.)

```
int foo(){  
    static estado = 100;  
    ...  
    estado--;  
    ...  
}
```