

Interrupciones

Informática II - R2004
2018

Recordando...

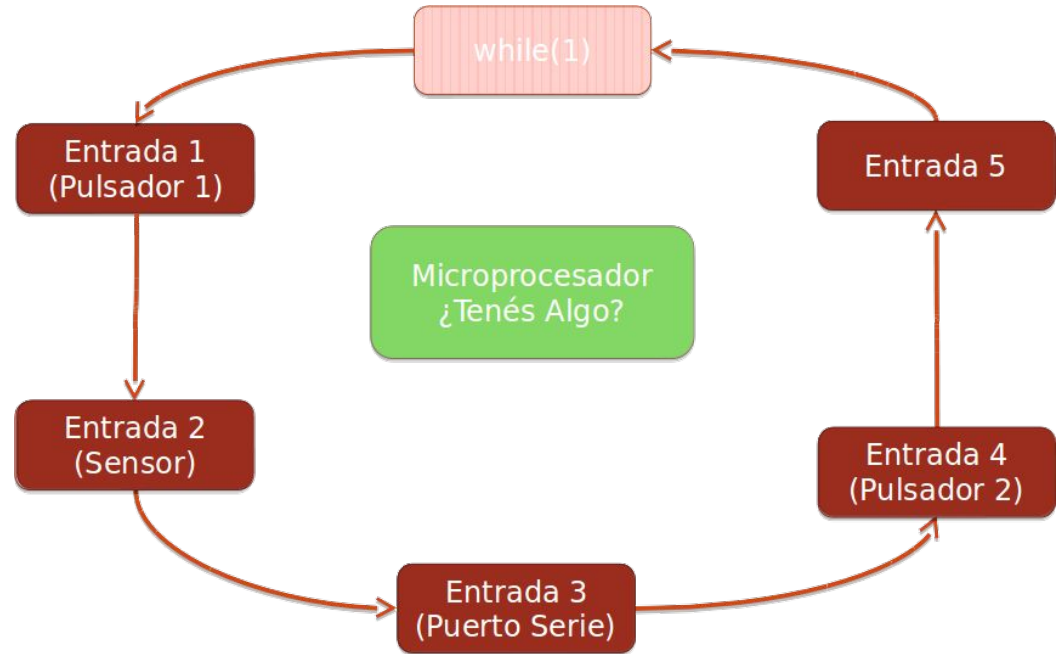
¿Cómo hacemos para que el microcontrolador esté siempre pendiente de lo que está pasando?

```
while ( 1 )  
{  
    if (LeerBoton1() == 1)  
        EncenderLampara1();  
    else  
        ApagarLampara1();  
  
    if (LeerBoton2() == 1)  
        EncenderLampara2();  
    else  
        ApagarLampara2();  
}
```

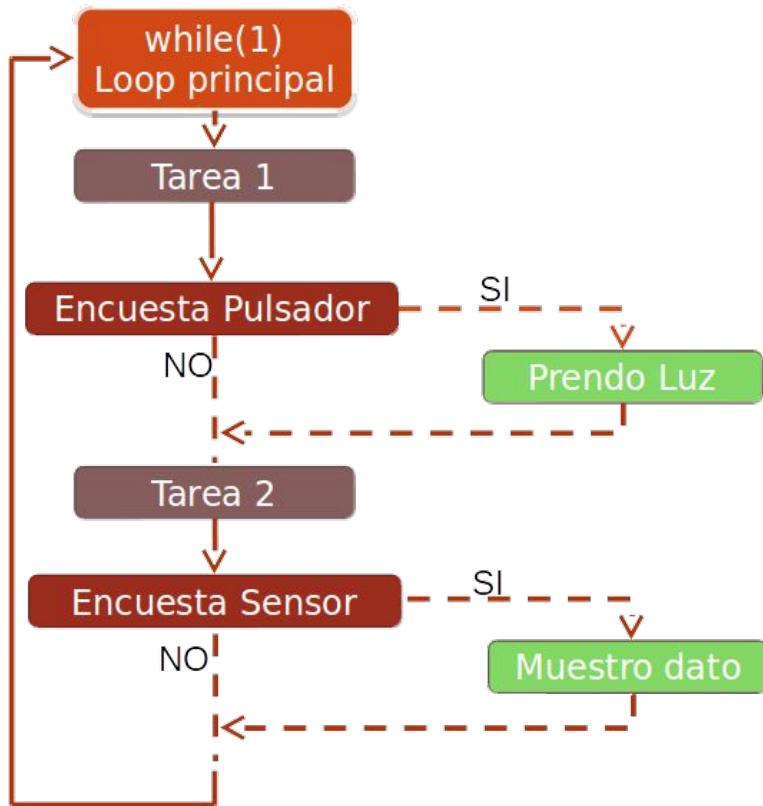


Entonces... ¿Cómo venimos programando?

```
void main ( void )  
{  
    Inicializar();  
    while ( 1 ) {  
        /* ... */  
        if ( Pulsador() == TRUE )  
        /* ... */  
        if ( Sensor() == FALSE )  
        /* ... */  
        if ( Temp() == TEMP_MAX )  
        /* ... */  
        if ( DatoSerie() == ERROR )  
        /* ... */  
    }  
}
```



Pooling - Atención SINCRÓNICA de las E/S

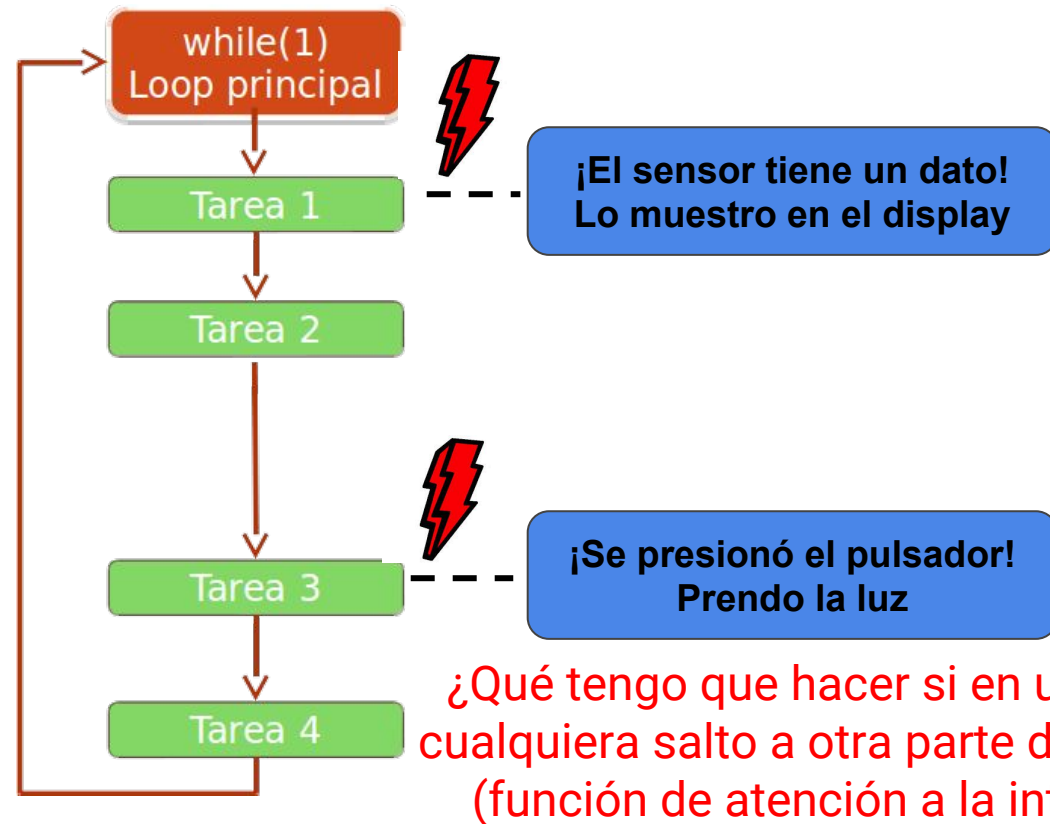


En este esquema el microcontrolador sabe exactamente en qué momento va a ver la entrada (el sensor, pulsador, etc.).

Esto hace que la atención a esa entrada sea **SINCRÓNICA** con el programa (el programa le destina un determinado momento para atender a ese evento).

¿Qué problemas puede tener esta estrategia?

Interrupciones - Atención ASINCRÓNICA



En este esquema la entrada **INTERRUMPE** la ejecución del programa para indicar que hay algún dispositivo que necesita atención.

Esto hace que la atención a esa entrada sea **ASINCRÓNICA** con el programa (no sabemos en qué momento puede llegar una INTERRUPTIÓN).

¿Qué tengo que hacer si en un momento cualquiera salto a otra parte del programa?
(función de atención a la interrupción)

¿Qué tengo que hacer antes de atender la interrupción? (I)

```
switch ( estado ){  
    /*...*/  
    case MAQUINA_ON:  
        if ( Pulsador() == TRUE ){  
            if ( luz )  
                SetPIN( LED1 , OFF );  
            else  
                SetPIN( LED1 , ON );  
        }  
        break;  
    /*...*/  
}
```



Al momento que llega una interrupción, debería:

- Terminar lo que estaba haciendo.
- Guardar el estado del micro al momento de la interrupción.
- Guardar el lugar a donde tengo que volver luego de atender a la interrupción (punto del programa)
- **Atender la interrupción**
- Recuperar el estado del micro, volver al punto del programa desde donde partí y continuar la ejecución.

¿Qué tengo que hacer antes de atender la interrupción? (II)

- Terminar lo que estaba haciendo.

```
if (luz)
0x00000c9c <EINT3_IRQHandler+28>: movw r3, #516 ; 0x204
0x00000ca0 <EINT3_IRQHandler+32>: movt r3, #4096 ; 0x1000
0x00000ca4 <EINT3_IRQHandler+36>: ldr r3, [r3, #0]
0x00000ca6 <EINT3_IRQHandler+38>: cmp r3, #0
0x00000ca8 <EINT3_IRQHandler+40>: beq.n 0xcc0 <EINT3_IRQHandler+64>
    SetPIN(LED1, ON);
0x00000caa <EINT3_IRQHandler+42>: movw r0, #49216 ; 0xc040
0x00000cae <EINT3_IRQHandler+46>: movt r0, #8201 ; 0x2009
0x00000cb2 <EINT3_IRQHandler+50>: mov.w r1, #0
0x00000cb6 <EINT3_IRQHandler+54>: mov.w r2, #1
0x00000cba <EINT3_IRQHandler+58>: bl 0xb80 <SetPIN>
0x00000cbe <EINT3_IRQHandler+62>: b.n 0xcd4 <EINT3_IRQHandler+84>
    SetPIN(LED1, OFF);
0x00000cc0 <EINT3_IRQHandler+64>: movw r0, #49216 ; 0xc040
0x00000cc4 <EINT3_IRQHandler+68>: movt r0, #8201 ; 0x2009
0x00000cc8 <EINT3_IRQHandler+72>: mov.w r1, #0
0x00000ccc <EINT3_IRQHandler+76>: mov.w r2, #0
```



Siempre que llegue una interrupción el micro va a estar haciendo ALGO (ejecutando alguna instrucción).

Las instrucciones de ASSEMBLER no pueden ser detenidas por la mitad, por lo que en primer lugar se debe completar la instrucción en curso.

¿Qué tengo que hacer antes de atender la interrupción? (III)

- Guardar el estado del micro al momento de la interrupción.

Estado del microcontrolador: Program Status Registers (xPSW)



Entre los 3 registros PSR (Application, Interrupt y Execution PSR) se puede saber el estado del microcontrolador en un determinado momento.

¿Qué tengo que hacer antes de atender la interrupción? (IV)

- Guardar el lugar a donde tengo que volver luego de atender a la interrupción

PC →

```
movw r3,  
movt r3,  
ldr r3,  
cmp r3,  
beq.n 0xcc  
  
movw r0,  
movt r0,  
mov.w r1,  
mov.w r2,  
bl 0xb8  
b.n 0xcd  
  
movw r0,  
movt r0,  
mov.w r1,  
mov.w r2,
```

El Program Counter (PC) es un registro de 32 bits que guarda la dirección de la instrucción que se está ejecutando en cada momento.

Salvar el punto de retorno del programa implica, entonces, guardar el valor del registro PC

Y... ¿Cómo hago todo esto?

!!!Lo hace automáticamente el procesador!!!



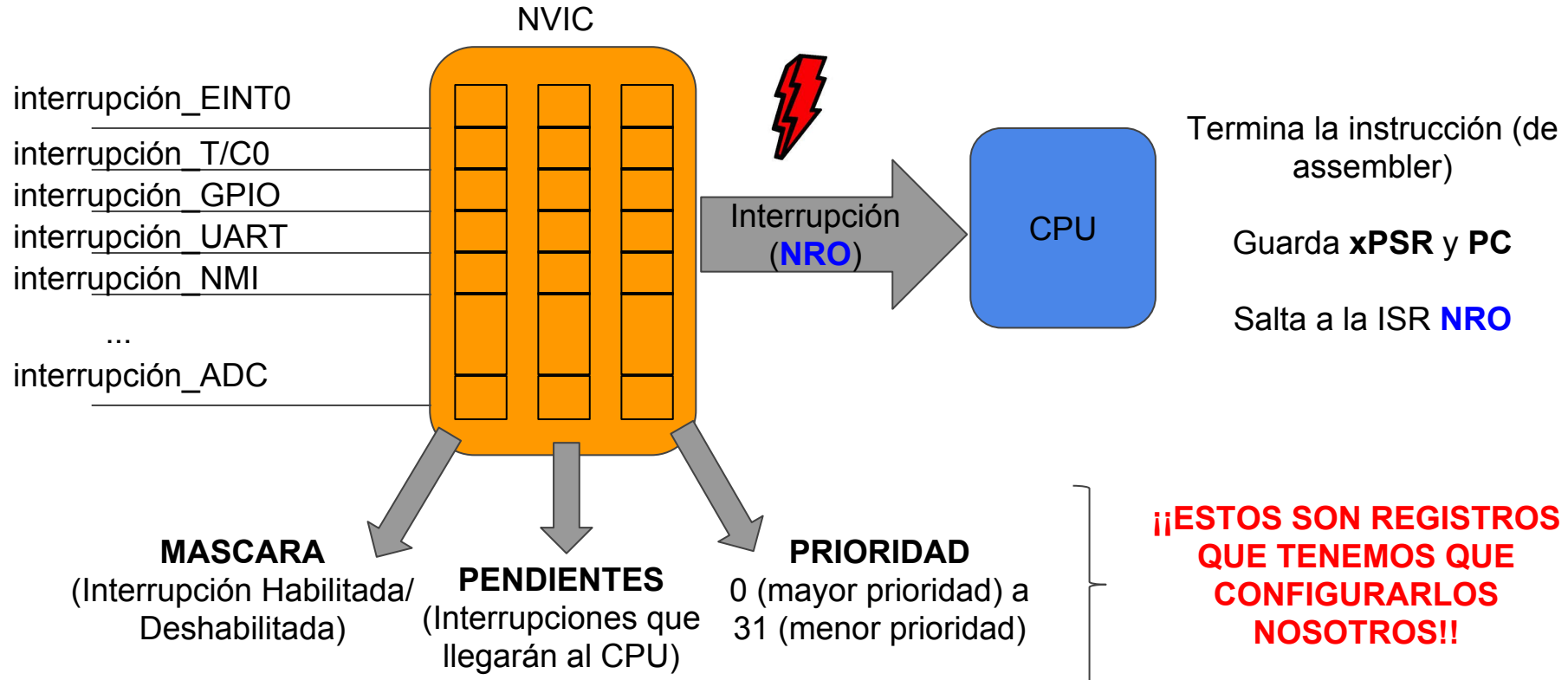
Sin embargo... ¿Cómo configuro el comportamiento de las interrupciones?

El **microprocesador** tiene un periférico encargado de manejar las interrupciones, llamado NVIC (Nested Vectored Interrupt Controller).

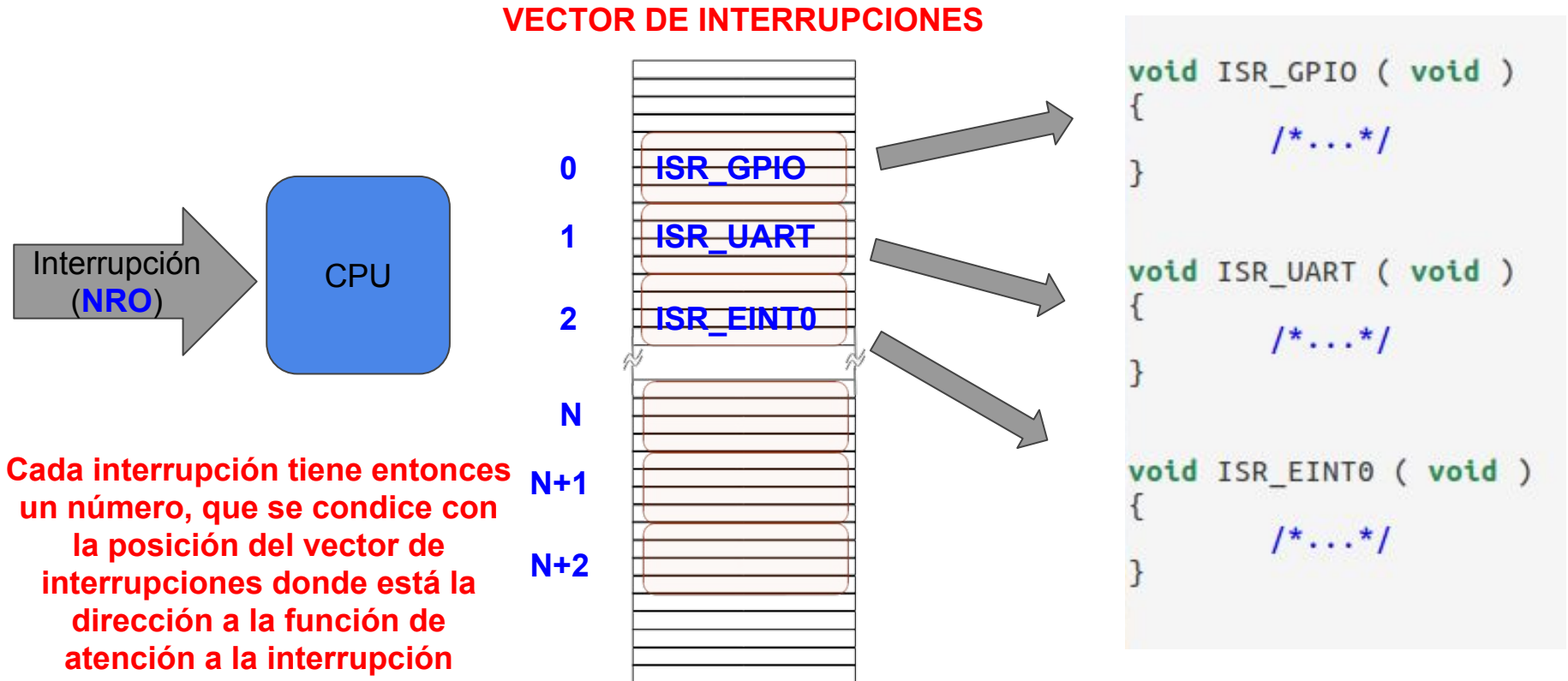
Este periférico es el encargado de recibir los pedidos de interrupción, manejarlos (habilitarlos o deshabilitarlos), darles prioridad y hacer llegar la señal a la CPU para que se genere la interrupción.

El NVIC puede manejar hasta 240 **fuentes de interrupciones** (El LPC1769 tiene implementadas hasta 35), puede darles diferentes **niveles de prioridad** (hasta 256 en el NVIC, 32 implementados en el LPC1769), y **enmascarar** cada una individualmente.

Comunicación NVIC - CPU



Y... ¿Cómo sabe la CPU qué tiene que hacer?




Fuentes de interrupción - Hoja de datos

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
16	32	0x80	PLL0 (Main PLL)	PLL0 Lock (PLOCK0)
17	33	0x84	RTC	Counter Increment (RTCCIF) Alarm (RTCALF)
18	34	0x88	External Interrupt	External Interrupt 0 (EINT0)
19	35	0x8C	External Interrupt	External Interrupt 1 (EINT1)
20	36	0x90	External Interrupt	External Interrupt 2 (EINT2)
21	37	0x94	External Interrupt	External Interrupt 3 (EINT3). Note: EINT3 channel is shared with GPIO interrupts
22	38	0x98	ADC	A/D Converter end of conversion

Las INTERRUPTCIONES y las EXCEPCIONES tienen un tratamiento idéntico, y muchas veces se usa un nombre en lugar del otro. La diferencia entre ambas es que las EXCEPCIONES suelen generarse por software, mientras que las INTERRUPTCIONES las genera el hardware.

¿Cómo configurar las interrupciones?

1. Inicializar la tabla de vectores.
 2. Configurar el NVIC.
 3. Configurar el periférico correspondiente y los pines asociados.
 4. Escribir la rutina de servicio de interrupción (Isr).
- 

Inicializar la Tabla de vectores (I)

El fabricante lo resuelve en el archivo *cr_startup_lpc175x_6x.c*

```
/**
 *
 * // The vector table.
 * // This relies on the linker script to place at correct location in memory.
 *
 */
extern void (* const g_pfnVectors[])(void);
__attribute__((used, section(".isr_vector")))
void (* const g_pfnVectors[])(void) = {
    // Core Level - CM3
    &_vStackTop, // The initial stack pointer
    ResetISR,    // The reset handler
    NMI_Handler, // The NMI handler
    HardFault_Handler, // The hard fault handler
    MemManage_Handler, // The MPU fault handler
    BusFault_Handler, // The bus fault handler
    UsageFault_Handler, // The usage fault handler
    __valid_user_code_checksum, // LPC MCU Checksum
    0, // Reserved
    0, // Reserved
    0, // Reserved
    SVC_Handler, // SVC call handler
    DebugMon_Handler, // Debug monitor handler
}
```

Vector de punteros a función

... Y muchas más!
Abramos el archivo para verlo.

Inicializar la Tabla de vectores (II)

```
// The vector table.  
// This relies on the linker script to place at correct location in memory.  
//  
//*****  
extern void (* const g_pfnVectors[])(void);  
__attribute__ ((used,section(".isr_vector")))  
void (* const g_pfnVectors[])(void) = {  
    // Core Level - CM3  
    &vStackTop, // The initial stack pointer  
    ResetISR,   // The reset handler  
    NMI_Handler, // The NMI handler  
    //...  
    SysTick_Handler, // The SysTick handler  
  
    // Chip Level - LPC17  
    WDT_IRQHandler, // 16, 0x40 - WDT  
    TIMER0_IRQHandler, // 17, 0x44 - TIMER0  
    //...  
    UART0_IRQHandler, // 21, 0x54 - UART0  
    //...  
    EINT0_IRQHandler, // 34, 0x88 - EINT0  
    EINT1_IRQHandler, // 35, 0x8c - EINT1  
    //...  
};
```

El fabricante ya les pone nombre a las funciones. NO PUEDO ELEGIR QUE NOMBRE PONERLE

Prototipos de las funciones de interrupción

Para poder asignarlas al vector de funciones de interrupción se deben definir y declarar las funciones.

Prototipos de funciones de interrupción para excepciones Core

```
void ResetISR(void);  
WEAK void NMI_Handler(void);  
WEAK void HardFault_Handler(void);  
WEAK void MemManage_Handler(void);  
WEAK void BusFault_Handler(void);  
WEAK void UsageFault_Handler(void);  
WEAK void SVCall_Handler(void);  
WEAK void DebugMon_Handler(void);  
WEAK void PendSV_Handler(void);  
WEAK void SysTick_Handler(void);  
WEAK void IntDefaultHandler(void);
```

Prototipos de funciones de interrupción para interrupciones Cortex M3

```
...  
void PWM1_IRQHandler(void) ALIAS(IntDefaultHandler);  
void I2C0_IRQHandler(void) ALIAS(IntDefaultHandler);  
void I2C1_IRQHandler(void) ALIAS(IntDefaultHandler);  
void I2C2_IRQHandler(void) ALIAS(IntDefaultHandler);  
void SPI_IRQHandler(void) ALIAS(IntDefaultHandler);  
void SSP0_IRQHandler(void) ALIAS(IntDefaultHandler);  
void SSP1_IRQHandler(void) ALIAS(IntDefaultHandler);  
void PLL0_IRQHandler(void) ALIAS(IntDefaultHandler);  
void RTC_IRQHandler(void) ALIAS(IntDefaultHandler);  
void EINT0_IRQHandler(void) ALIAS(IntDefaultHandler);  
...
```


Prototipos de las funciones de interrupción (II)

void EINT0_IRQHandler(void) ALIAS(IntDefaultHandler);

ALIAS significa que el nombre de la función definida refiere a otra función. Se usa para hacer que muchas funciones se refieran a una misma función. Es una función default que debe estar definida si o si por seguridad. El modificador **ALIAS** está definido para que incluya también al atributo weak, por lo tanto las funciones ISR (Rutinas de servicio de interrupciones) están declaradas por default como **ALIAS** lo que implica que todas se refieren a otra común y que son válidas siempre y cuando no se defina otra con el mismo nombre que no sea **WEAK**.

WEAK **void SysTick_Handler(void)**;

WEAK significa que esa función será pisada por otra con el mismo nombre. Si no hay otra con el mismo nombre, entonces es válida. Se usa esto para definir funciones default.



funciones de interrupción default

```
__attribute__((section(".after_vectors")))
void SysTick_Handler(void)
{ while(1) {}
}
```

```
__attribute__((section(".after_vectors")))
void IntDefaultHandler(void)
{ while(1) {}
}
```

IMPORTANTE: Las funciones de interrupción por defecto realizar un while(1), si no redefino las que voy a usar la ejecución se va a quedar bloqueada en la función.

Para redefinirla, debido al atributo WEAK, simplemente escribimos otra función **con el mismo nombre** y ya tenemos NUESTRA función de interrupción

Configurar el NVIC (Registros ISER)

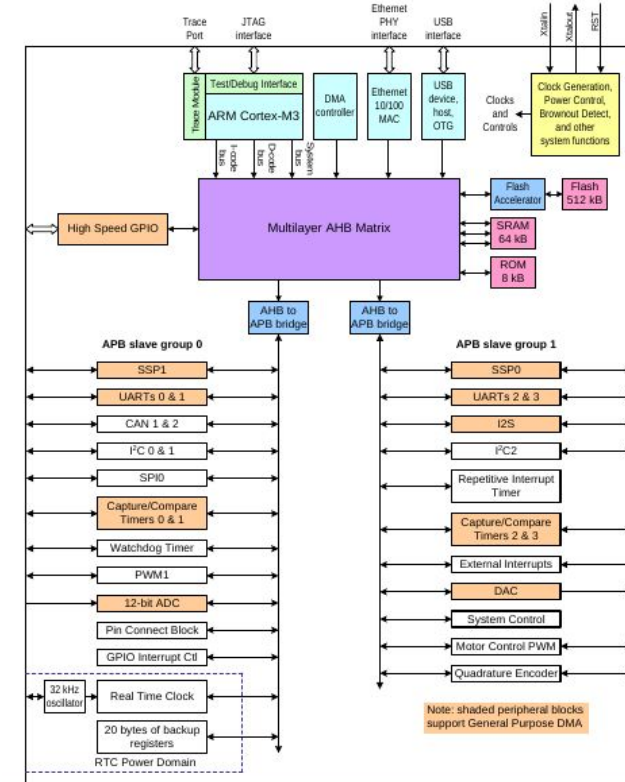
Table 52. Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100)

Bit	Name	Function
0	ISE_WDT	Watchdog Timer Interrupt Enable. Write: writing 0 has no effect, writing 1 enables the interrupt. Read: 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.
1	ISE_TIMER0	Timer 0 Interrupt Enable. See functional description for bit 0.
2	ISE_TIMER1	Timer 1. Interrupt Enable. See functional description for bit 0.
3	ISE_TIMER2	Timer 2 Interrupt Enable. See functional description for bit 0.
4	ISE_TIMER3	Timer 3 Interrupt Enable. See functional description for bit 0.
18	ISE_EINT0	External Interrupt 0 Interrupt Enable. See functional description for bit 0.
19	ISE_EINT1	External Interrupt 1 Interrupt Enable. See functional description for bit 0.
20	ISE_EINT2	External Interrupt 2 Interrupt Enable. See functional description for bit 0.
21	ISE_EINT3	External Interrupt 3 Interrupt Enable. See functional description for bit 0.
22	ISE_ADC	ADC Interrupt Enable. See functional description for bit 0.

Colocando un 1 en el bit correspondiente del registro ISER0 o ISER1 habilito al NVIC para controlar esa interrupción

Configurar el periférico y los pines asociados

- Cada periférico posee sus fuentes de interrupción.
- Se deben habilitar en los registros del periférico que deseo utilizar las interrupciones que queremos utilizar.
- Se debe implementar la función de interrupción asociada a la fuente elegida.



Escribir la rutina de servicio de interrupción (Isr).

Para la fuente de interrupción que deseamos utilizar buscamos en el archivo startup el nombre que debemos utilizar y redefinimos la función.

IMPORTANTE

- Nuestro código de una función de interrupción debe ser CORTO y SIN LOOPs BLOQUEANTES.
- Las funciones de interrupción son ASINCRONICAS y las invoca el CPU, NO podemos recibir NI devolver nada.
- Para comunicarnos con nuestro programa principal debe utilizar variables globales

```
void SysTick_Handler(void)
{
    //Nuestro código
}
```


Comunicación entre interrupción y programa principal

```
int main(void)
{
    Kit_Init();
    HW_Init();

    while(1)
    {
        if(g_flag)
        {
            if(GetPIN(0, 22))
                SetPIN(0, 22, 0);
            else
                SetPIN(0, 22, 1);

            g_flag = 0;
        }
    }

    return 0 ;
}
```

Se produce la
interrupción. No
sabemos cuando!

```
void EINT3_IRQHandler(void)
{
    EXTINT |= 1 << 3;
}
```

Para avisarle al programa
principal cambiamos un
flag GLOBAL.

```
uint8_t g_flag = 0;

void EINT3_IRQHandler(void)
{
    EXTINT |= 1 << 3;

    g_flag = 1;
}
```


Comunicación entre interrupción y programa principal

En caso de que la acción a realizar por la función de interrupción sea corto y conciso se puede realizar dentro de la función. **NUNCA** tengo que realizar tareas largas o loops bloqueantes

```
int main(void)
{
    Kit_Init();
    HW_Init();

    while(1)
    {

    }

    return 0 ;
}
```

```
void EINT3_IRQHandler(void)
{
    EXTINT |= 1 << 3;

    if(GetPIN(0, 22))
        SetPIN(0, 22, 0);
    else
        SetPIN(0, 22, 1);
}
```

Interrupciones Externas - Fuentes

El LPC1769 tiene 4 fuentes de interrupción externa. Estas fuentes se identifican de 0 a 3.

PINSEL4	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P2.0	GPIO Port 2.0	PWM1.1	TXD1	Reserved	00
3:2	P2.1	GPIO Port 2.1	PWM1.2	RXD1	Reserved	00
5:4	P2.2	GPIO Port 2.2	PWM1.3	CTS1	Reserved [2]	00
7:6	P2.3	GPIO Port 2.3	PWM1.4	DCD1	Reserved [2]	00
9:8	P2.4	GPIO Port 2.4	PWM1.5	DSR1	Reserved [2]	00
11:10	P2.5	GPIO Port 2.5	PWM1.6	DTR1	Reserved [2]	00
13:12	P2.6	GPIO Port 2.6	PCAP1.0	RI1	Reserved [2]	00
15:14	P2.7	GPIO Port 2.7	RD2	RTS1	Reserved	00
17:16	P2.8	GPIO Port 2.8	TD2	TXD2	ENET_MDC	00
19:18	P2.9	GPIO Port 2.9	USB_CONNECT	RXD2	ENET_MDIO	00
21:20	P2.10	GPIO Port 2.10	EINT0	NMI	Reserved	00
23:22	P2.11 [1]	GPIO Port 2.11	EINT1	Reserved	I2STX_CLK	00
25:24	P2.12 [1]	GPIO Port 2.12	EINT2	Reserved	I2STX_WS	00
27:26	P2.13 [1]	GPIO Port 2.13	EINT3	Reserved	I2STX_SDA	00
31:28	-	Reserved	Reserved	Reserved	Reserved	0

Al configurar alguno de estos pines (**P2.10 - P2.11 - P2.12 - P2.13**) en **modo 01** tengo disponible, por cada uno de ellos, **una función de interrupción para asociarlas a los cambios del Pin.**

Configurar el NVIC - ISER0

18	ISE_EINT0	External Interrupt 0 Interrupt Enable. See functional description for bit 0.
19	ISE_EINT1	External Interrupt 1 Interrupt Enable. See functional description for bit 0.
20	ISE_EINT2	External Interrupt 2 Interrupt Enable. See functional description for bit 0.
21	ISE_EINT3	External Interrupt 3 Interrupt Enable. See functional description for bit 0.
22	ISE_ADC	ADC Interrupt Enable. See functional description for bit 0.

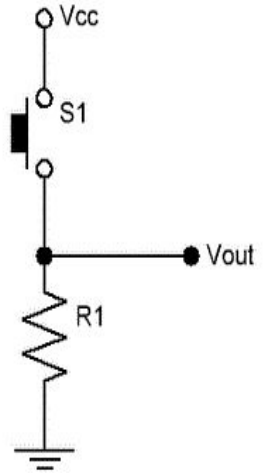
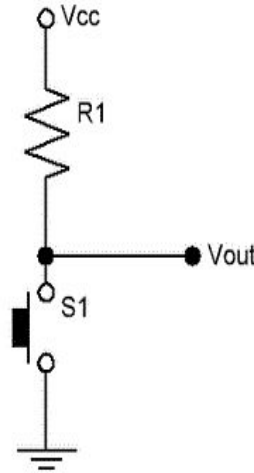
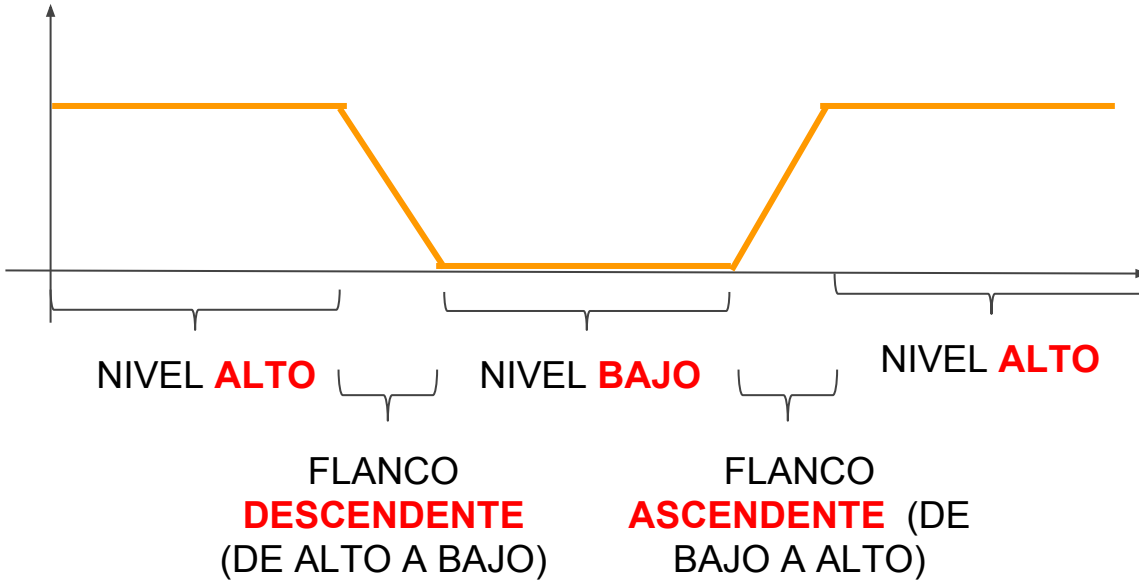
Para habilitar la fuente de interrupción en el NVIC debemos colocar un 1 en el bit 18, 19, 20 o 21 respectivamente para cada interrupción externa

Configurar el periférico

Pin name	Pin direction	Pin description
EINT0	Input	External Interrupt Input 0 - An active low/high level or falling/rising edge general purpose interrupt input. This pin may be used to wake up the processor from Sleep, Deep-sleep, or Power-down modes.
EINT1	Input	External Interrupt Input 1 - See the EINT0 description above.
EINT2	Input	External Interrupt Input 2 - See the EINT0 description above.
EINT3	Input	External Interrupt Input 3 - See the EINT0 description above.
RESET	Input	External Reset input - A LOW on this pin resets the chip, causing I/O ports and peripherals to take on their default states, and the processor to begin execution at address 0x0000 0000.

Las interrupciones externas se pueden configurar para que se activen por **NIVEL** alto o bajo o por **FLANCO** descendente o ascendente. También se puede utilizar para “despertar” el microprocesador si se encuentra en modo bajo consumo

“Flancos” y “niveles” en las señales



Configurar una interrupción por FLANCO o por NIVEL va a depender del circuito y de la aplicación que estemos desarrollando

Registros a configurar

Registro donde se activan los flags indicando que llegó una interrupción. Registro donde se indica si quiero que la interrupción se genere por NIVEL o por FLANCO. Registro donde se indica cual NIVEL o FLANCO deseo o cual NIVEL deseo.

Name	Description	Access	Reset value	Address
EXTINT	The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, EINT2 and EINT3. See Table 10 .	R/W	0x00	0x400F C140
EXTMODE	The External Interrupt Mode Register controls whether each pin is edge- or level-sensitive. See Table 11 .	R/W	0x00	0x400F C148
EXTPOLAR	The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt. See Table 12 .	R/W	0x00	0x400F C14C

Registro EXTINT

Bit	Symbol	Description	Reset value
0	EINT0	<p>In level-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the selected edge occurs on the pin.</p> <p>This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. [1]</p>	0
1	EINT1	<p>In level-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the selected edge occurs on the pin.</p> <p>This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. [1]</p>	0
2	EINT2	<p>In level-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the selected edge occurs on the pin.</p> <p>This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. [1]</p>	0
3	EINT3	<p>In level-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the selected edge occurs on the pin.</p> <p>This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. [1]</p>	0
31:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Registro EXTMODE

Bit	Symbol	Value	Description	Reset value
0	EXTMODE0	0	Level-sensitivity is selected for $\overline{\text{EINT0}}$.	0
		1	$\overline{\text{EINT0}}$ is edge sensitive.	
1	EXTMODE1	0	Level-sensitivity is selected for $\overline{\text{EINT1}}$.	0
		1	$\overline{\text{EINT1}}$ is edge sensitive.	
2	EXTMODE2	0	Level-sensitivity is selected for $\overline{\text{EINT2}}$.	0
		1	$\overline{\text{EINT2}}$ is edge sensitive.	
3	EXTMODE3	0	Level-sensitivity is selected for $\overline{\text{EINT3}}$.	0
		1	$\overline{\text{EINT3}}$ is edge sensitive.	
31:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Indica si la interrupción se activará por **nivel** o **flanco**.

Si el bit correspondiente **vale 0**, la interrupción será activa por nivel y si vale 1 por flanco.

Registro EXTPOLAR

Bit	Symbol	Value	Description	Reset value
0	EXTPOLAR0	0	$\overline{\text{EINT0}}$ is low-active or falling-edge sensitive (depending on EXTMODE0).	0
		1	$\overline{\text{EINT0}}$ is high-active or rising-edge sensitive (depending on EXTMODE0).	
1	EXTPOLAR1	0	$\overline{\text{EINT1}}$ is low-active or falling-edge sensitive (depending on EXTMODE1).	0
		1	$\overline{\text{EINT1}}$ is high-active or rising-edge sensitive (depending on EXTMODE1).	
2	EXTPOLAR2	0	$\overline{\text{EINT2}}$ is low-active or falling-edge sensitive (depending on EXTMODE2).	0
		1	$\overline{\text{EINT2}}$ is high-active or rising-edge sensitive (depending on EXTMODE2).	
3	EXTPOLAR3	0	$\overline{\text{EINT3}}$ is low-active or falling-edge sensitive (depending on EXTMODE3).	0
		1	$\overline{\text{EINT3}}$ is high-active or rising-edge sensitive (depending on EXTMODE3).	
31:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Indica si la interrupción correspondiente será **activa por nivel bajo/alto o por flanco descendente o ascendente**. Esto dependerá de la elección hecha en EXTMODE

Veamos un ejemplo!

