# Resumen del Manejo de Interrupciones en el LPC1768

*Versión 5.1*

## Sistemas Electrónicos Digitales


## Universidad de Alcalá
## Departamento de Electrónica

# 1 Introducción al documento

Este documento presenta un resumen del sistema de gestión de interrupciones del Cortex M3 y de la configuración y manejo de las Interrupciones Externas en el LPC1768 que puede servir de base para la programación de interrupciones del LPC1768. También se presenta la biblioteca de funciones CMSIS definidas para homogeneizar el acceso a algunos recursos de procesadores basados en la arquirtectura Cortex y utilizado en la gestión de interrupciones.

Como anexos se incluyen la tabla de vectores del LPC1768 y los nombres de las funciones de interrupción utilizados por Keil en la versión V. 4. 21 del uVision 4.

La información está obtenida fundamentalmente del LPC17xx User Manual proporcionado por NXP y de la documentación incluida en la ayuda de Keil uVision 4.

# 2 Índice de contenidos

# 3 Modelo de Excepciones del Cortex M3

Al enfrentarse por primera vez a un microcontrolador es fundamental analizar cómo funciona su sistema de gestión de excepciones e interrupciones, conocer cómo se tratan sus prioridades y cómo y dónde se configuran, saber dónde está la tabla de vectores y saber realizar funciones de atención a las diferentes excepciones. También es importante saber las interrupciones externas de las que dispone y cómo se configuran.

Una excepción es un evento que se puede producir por diferentes causas y que, cuando se produce, si está habilitado, provoca la ejecución inmediata de una rutina de atención a la excepción interrumpiendo el programa que se estuviera ejecutando en ese momento. En la mayor parte de las excepciones, al finalizar la rutina de excepción, se produce un retorno al programa que se estaba ejecutando.

La dirección de salto asociada a cada una de las excepciones se encuentra en la llamada "Tabla de Vectores" donde el primer elemento es el "Vector de Reset". Tras el Reset, la tabla de vectores está situada en la posición de memoria 0x00000000 aunque se podría trasladar a otro lugar si fuera necesario. En la tabla de vectores hay una entrada por cada una de las posibles excepciones (112 excepciones en el LPC1768)[1].

**Las interrupciones son un caso particular de excepciones** donde el evento es producido por un periférico de la CPU (situados dentro del microcontrolador) o por interrupciones externas. El Cortex M3 distingue entre la atención a interrupciones (ISR) y la atención al resto de excepciones (Reset, Fault handlers y System handlers). A continuación se desarrolla más en profundidad la gestión de interrupciones dejando de lado la gestión del resto de excepciones.

Todas las excepciones tienen asociado un número de excepción, un número de interrupción y una prioridad [2]:

- El número de excepción va de 1 a 15 para las excepciones que no son interrupciones y del 16 en adelante para las interrupciones.
- El número de interrupción ( IRQ Number) toma valores de -11 a -1 para las excepciones que no son interrupciones y del 0 en adelante para las interrupciones.
- La prioridad es configurable salvo en el Reset y en la NMI (las dos primeras) que tienen las prioridades más altas posibles (-3 y -2 respectivamente). Se puede asignar a cada excepción un nivel de prioridad del 0 al 31, siendo el 0 el más prioritario. Tras el Reset, todas las excepciones de prioridad configurable tienen

[1] Ver apartados 6.4 y 34.4.3.5 del "LPC17xx User Manual" de NXP

[2] Ver apartados 34.3.3 del "LPC17xx User Manual" de NXP

como nivel de prioridad el 0. La prioridad de las excepciones (salvo las interrupciones) se configura con los registros SHPR1 – SHPR3[3] y las de las interrupciones en los registros IPR0 – IPR27 (en otro sitio pone hasta IPR59) [4].

Las interrupciones pueden estar en tres estados: inactiva, pendiente y ejecutándose (inactive, pending, run)[5] :

- Inactive: no se tiene registro de que se haya producido el evento asociado a la excepción.
- Pending: se ha detectado el evento asociado a una excepción y se ha registrado pero todavía no se está ejecutando la rutina de atención correspondiente. Se puede saber si una interrupción está pendiente o no con la lectura de los registros ISPRn o ICPRn.
- Run: se está ejecutando la rutina de atención correspondiente. Se puede saber si una interrupción está activa con la lectura del registro IABRn.

Se puede forzar el paso de una interrupción a pendiente mediante la escritura de un 1 en el lugar correspondiente del registro ISPRn y se puede forzar el paso de pendiente a inactiva mediante la escritura de un 1 en el lugar correspondiente del registro ICPRn.

Existen tres registros relacionados con el enmascaramiento de interrupciones para evitar que sean atendidas cuando están pendientes[6]:

- Escribir un 1 en el registro PRIMASK enmascara la activación de todas las interrupciones de prioridad configurable, es decir todas excepto la NMI y el HardFault.
- Escribir un 1 en el registro FAULTMASK enmascara la activación de todas las interrupciones excepto la NMI. El procesador borra el FAULTMASK al finalizar la atención de cualquier excepción, salvo de la NMI.
- Mediante la escritura del registro BASEPRI se define el nivel de prioridad máxima que es atendido. Las interrupciones de nivel de prioridad igual o inferior al de BASEPRI no se atienden. Si BASEPRI vale cero no tiene ningún efecto.

Tras la ejecución de cada instrucción de ensamblador, se evalúan las interrupciones que están en estado PENDING y que no están enmascaradas. Si hay alguna en esta situación, se produce un salto a su rutina de interrupción asociada tras obtener la dirección de salto de la “Tabla de Vectores”.

[3] Ver apartados 34.4.3.9 del LPC17xx User Manual

[4] Ver apartados 34.4.2.7 y 6.5.11-19 del LPC17xx User Manual

[5] Ver apartados 34.4.2.9.1 del LPC17xx User Manual

[6] Ver apartados 34.3.1.3.6 del LPC17xx User Manual

Si en un determinado momento, hay más de una interrupciónn en estado PENDING, se ejecutará la rutina de interrupción asociada a la interrupción más prioritaria (con mayor número de prioridad). En caso de estar pendientes varias interrupciones del mismo nivel de prioridad, se atenderá a la que tenga menor número de excepción.

Si cuando está ejecutándose una rutina de atención a una interrupción, pasa a estado PENDING una interrupción de mayor nivel de prioridad no enmascarada, interrumpirá a la interrupción menos prioritaria anidando interrupciones. En cambio, si la interrupción es del mismo nivel de prioridad que la que se está ejecutando, no será interrumpida, esperando a que finalice la rutina de interrupción en ejecución para ser atendida.

Existe la posibilidad de dividir el nivel de prioridad en dos campos (grupo y subgrupo) que se puede configurar con el campo PRIGROUP del registro AIRCR [7] . En este caso, el anidamiento de interrupciones sólo se produce cuando está pendiente una interrupción de un grupo superior y está en ejecución una de un grupo inferior.

La NMI (Non Maskable Interrupt – Interrupción no enmascarable) es una excepción asociada a un pin del microcontrolador que siempre que se active, produce una interrupción. En el LPC1768 la NMI está en el pin 53 y comparte funcionalidad con el P2.10 y EINT0. NMI es la tercera función del pin y por tanto, para que esté activa, es necesario configurar un 10 en los bits 21:20 del registro PINSEL4 (Pin Function Select Register 4). El bit NMIPENDSET del registro ICSR indica si la NMI está pendiente o no. La escritura de un 1 fuerza el paso de la NMI a estado pendiente lo que provoca la entrada inmediata de su función de atención asociada ya que es la interrupción más prioritaria y no enmascarable. Al entrar en la rutina se borra el estado de pendiente.

# 4 Interrupciones externas del LPC1768

Las interrupciones externas permiten que un cambio del nivel de un pin de entrada del microcontrolador, provoque la ejecución de una rutina de atención a esa interrupción. Las interrupciones externas suelen utilizarse para que el procesador responda a eventos de elementos de hardware externos.

Las interrupciones externas se pueden configurar activas por nivel (alto o bajo) o activas por flanco (bajada o subida). Una interrupción activa por nivel implica que permanecerán activas mientras el nivel asociado se mantenga. Normalmente las interrupciones activas por nivel se utilizan para atender un evento de un dispositivo externo que desactiva la interrupción cuando es atendido.

[7] Ver apartados 34.4.3.6 y 34.3.3.6 del LPC17xx User Manual

El LPC1768 dispone de 4 entradas de interrupción externas específicas (/EINT0 .. /EINT3) situadas en los pines P2.10 … P2.13 como primera función alternativa por lo que para seleccionarlas es necesario escribir un 01 en los bits adecuados del registro PINSEL4. Además de estas cuatro líneas, se pueden configurar como interrupciones externas cualquiera de los pines de los puertos P0 y P2.

## 4.1 Interrupciones externas dedicadas

Las interrupciones externas /EINT0 - /EINT3 tienen asociados los números de interrupción IRQ18 – IRQ21 [8] y, como el resto de las interrupciones, se configuran con el acceso a los registros generales del NVIC [9] , habilitándolas y deshabilitándolas mediante los registros ISER0 e ICER0 (bits 18 al 21), configurando su nivel de prioridad con los registros IPR4 e IPR5 y consultando o modificando el estado de interrupción con los registros IABR0, ISPR0 e ICPR0.

El tipo de interrupción externa (activo por nivel o por flanco) y el nivel activo (activa por nivel alto o bajo, o flanco de bajada o subida) se configura con los registros EXTMODE y EXTPOLAR respectivamente [10] . Cuando la entrada está habilitada y se recibe un flanco o nivel activo, se pone a 1 el flag de interrupción asociado que se puede consultar leyendo el registro EXTINT y que, inicia el proceso de interrupción pasando su estado a PENDING y siendo ejecutado o no dependiendo de si están o no habilitadas y tienen la prioridad adecuada.

Dentro de la rutina de atención a las interrupciones externas, se deben borrar los flags de interrupción que las provocaron. Esto se realiza escribiendo un 1 en el bit correspondiente del registro EXTINT. Esto es muy importante porque si no se hace, no se detectarán eventos futuros. Si la interrupción es activa por nivel, el flag de interrupción no se podrá borrar mientras permanezca activo el nivel, relanzando la interrupción si termina la rutina antes de que cambie el nivel de activación.

En el manual de usuario se recomienda mantener siempre la interrupción desactivada cuando se modifique EXTMODE y borrar el flag asociado antes de habilitarla.

## 4.2 Interrupciones asociadas a los pines de los puertos GPIO0 y GPIO2

Cada una de las entradas de los puertos GPIO0 y GPIO2 puede ser configurada como interrupción externa activa por flanco de subida, de bajada o por los dos flancos. Esto se

[8] Ver Tabla 50 del LPC17xx User Manual

[9] Ver apartado 6.5 del LPC17xx User Manual

[10] Ver apartado 3.6 del LPC17xx User Manual

realiza con los registros IO0IntEnR e IO2IntEnR para la activación por flanco de subida, y los registros IO0IntEnF e IO2IntEnF para la activación por flanco de bajada [11]

Las interrupciones asociadas al GPIO0 y GPIO2 comparten en el NVIC los recursos con la Interrupción Externa 3 (/EINT3) con el número de interrupción IRQ21.

Los flags de interrupción asociados a cada una de las líneas de los puertos GPIO0 y GPIO2 se almacenan en los registros IO0IntStatR e IO2IntStatR cuando se produce un flanco de subida y en IO0IntStatF e IO2IntStatF cuando se produce un flanco de bajada. En el registro IOIntStatus se puede comprobar en un bit si hay alguna interrupción pendiente del GPIO0 y en otro bit si hay alguna del GPIO2. Para borrar los flags de interrupción asociados a cada pin de los puertos es necesario escribir un 1 en los registros IO0IntClr e IO2IntClr

## 5 Ejemplo de programa que utiliza interrupciones externas

A conticuación se presenta un ejemplo de programa que conmuta un LED conectado al puerto P1.29 con cada activación por flanco de bajada de la interrupción externa EINT3.

Algunos de los registros necesarios para configurar el NVIC como es el registro IPR para configurar la prioridad y al registro ISER0 para habilitar la interrupción, no están definidos en el archivo LPC17xx.H sino en el archivo CORE_CM3.H que es incluído en el propio LPC17xx.H junto con otras de finiciones de registros propios del CortexM3.

```
// *****************************************************
//
// File Name: ConmutaLedPrincipal.c
//
// Purpose: Programa ejemplo que muestra el control del parpadeo de un LED
//          con una entrada de interrupción
//              LED      --> P1.29
//              PULSADOR --> EINT3 (P2.13)
//
// *****************************************************

#include <LPC17xx.h>
#include "delay.h"

#define RETARDO 1000000   // Retardo

uint8_t activo = 1;      // Cuando es uno parpadea el LED


//Funcion de interrupcion EINT3
void EINT3_IRQHandler(void)
{
  // Borrar el falg de la EINT3 --> EXTINT.3
  LPC_SC->EXTINT = 1 << 3;

  // Actualiza la variable que activa o desactiva el parpadeo
```

[11] Ver apartado 9.5.6 del LPC17xx User Manual

```
  activo ++;
}


// Función de inicialización
void Config(void)
{
  // Configuracion del P2.13 como EINT3   --> PINSEL4.26 y PINSEL4.27
  LPC_PINCON->PINSEL4 |= 1 << (13*2);

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

  // Interrupción activa por flanco de bajada        --> EXTMODE.3
  LPC_SC->EXTMODE |= 1<< 3;

  // Configuramos prioridad 1 a la interrupción EINT3 (IRQ21) --> IPR5
  // La definición de IP es uint8_t. Los bits válidos son los 5 de mayor peso
  NVIC->IP[21] = 0x01 << 3;

  // Habilitar la interrupción EINT3  --> ISER0.21
  NVIC->ISER[0] = 1 << 21;
}


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

  // Programa principal
  while (1) {
    if ((activo & 1) == 1) {
            LPC_GPIO1->FIOPIN ^= 1<<29;
    }

    delay(RETARDO);
  }
}
```

# 6 Anexos

## 6.1 Anexo I: Tabla de nombres y número de interrupción

Esta tabla se encuentra en el fichero LPC17xx.H

```
/******  Cortex-M3 Processor Exceptions Numbers ***********************************************/
  NonMaskableInt_IRQn           = -14,      /*!< 2 Non Maskable Interrupt                        */
  MemoryManagement_IRQn         = -12,      /*!< 4 Cortex-M3 Memory Management Interrupt         */
  BusFault_IRQn                 = -11,      /*!< 5 Cortex-M3 Bus Fault Interrupt                 */
  UsageFault_IRQn               = -10,      /*!< 6 Cortex-M3 Usage Fault Interrupt               */
  SVCall_IRQn                   = -5,       /*!< 11 Cortex-M3 SV Call Interrupt                  */
  DebugMonitor_IRQn             = -4,       /*!< 12 Cortex-M3 Debug Monitor Interrupt            */
  PendSV_IRQn                   = -2,       /*!< 14 Cortex-M3 Pend SV Interrupt                  */
  SysTick_IRQn                  = -1,       /*!< 15 Cortex-M3 System Tick Interrupt              */

/******  LPC17xx Specific Interrupt Numbers ***************************************************/
  WDT_IRQn                      = 0,        /*!< Watchdog Timer Interrupt                        */
  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                       */
  EINT0_IRQn                    = 18,       /*!< External Interrupt 0 Interrupt                  */
  EINT1_IRQn                    = 19,       /*!< External Interrupt 1 Interrupt                  */
  EINT2_IRQn                    = 20,       /*!< External Interrupt 2 Interrupt                  */
  EINT3_IRQn                    = 21,       /*!< External Interrupt 3 Interrupt                  */
  ADC_IRQn                      = 22,       /*!< A/D Converter Interrupt                         */
  BOD_IRQn                      = 23,       /*!< Brown-Out Detect Interrupt                      */
  USB_IRQn                      = 24,       /*!< USB Interrupt                                   */
  CAN_IRQn                      = 25,       /*!< CAN Interrupt                                   */
  DMA_IRQn                      = 26,       /*!< General Purpose DMA Interrupt                   */
  I2S_IRQn                      = 27,       /*!< I2S Interrupt                                   */
  ENET_IRQn                     = 28,       /*!< Ethernet Interrupt                              */
  RIT_IRQn                      = 29,       /*!< Repetitive Interrupt Timer Interrupt            */
  MCPWM_IRQn                    = 30,       /*!< Motor Control PWM Interrupt                     */
  QEI_IRQn                      = 31,       /*!< Quadrature Encoder Interface Interrupt          */
  PLL1_IRQn                     = 32,       /*!< PLL1 Lock (USB PLL) Interrupt                   */
  USBActivity_IRQn              = 33,       /*!< USB Activity Interrupt
                */
  CANActivity_IRQn              = 34,       /*!< CAN Activity Interrupt
```

## 6.2 Anexo II: Definición de los nombres de rutinas de atención a las interrupciones

Tabla de nombres de las diferentes rutinas de interrupción obtenidas de la Tabla de Vectores de Interrupción definida en el archivo startup_LPC17xx.h

```
                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     SPI_IRQHandler             ; 29: SPI
                DCD     SSP0_IRQHandler            ; 30: SSP0
                DCD     SSP1_IRQHandler            ; 31: SSP1
                DCD     PLL0_IRQHandler            ; 32: PLL0 Lock (Main PLL)
                DCD     RTC_IRQHandler             ; 33: Real Time Clock
                DCD     EINT0_IRQHandler           ; 34: External Interrupt 0
                DCD     EINT1_IRQHandler           ; 35: External Interrupt 1
                DCD     EINT2_IRQHandler           ; 36: External Interrupt 2
                DCD     EINT3_IRQHandler           ; 37: External Interrupt 3
                DCD     ADC_IRQHandler             ; 38: A/D Converter
                DCD     BOD_IRQHandler             ; 39: Brown-Out Detect
                DCD     USB_IRQHandler             ; 40: USB
                DCD     CAN_IRQHandler             ; 41: CAN
                DCD     DMA_IRQHandler             ; 42: General Purpose DMA
                DCD     I2S_IRQHandler             ; 43: I2S
                DCD     ENET_IRQHandler            ; 44: Ethernet
                DCD     RIT_IRQHandler             ; 45: Repetitive Interrupt Timer
                DCD     MCPWM_IRQHandler           ; 46: Motor Control PWM
                DCD     QEI_IRQHandler             ; 47: Quadrature Encoder Interface
                DCD     PLL1_IRQHandler            ; 48: PLL1 Lock (USB PLL)
                DCD     USBActivity_IRQHandler     ; 49: USB Activity interrupt to wakeup
                DCD     CANActivity_IRQHandler     ; 50: CAN Activity interrupt to wakeup
```

Por ejemplo, las funciones de interrupción de las interrupciones externas de definen así

```
void EINT0_IRQHandler (void) {
       …….
}
void EINT1_IRQHandler (void) {
       …….
}
void EINT2_IRQHandler (void) {
       …….
}
void EINT3_IRQHandler (void) {
       …….
}
```

## 6.3 Anexo III: Resumen de registros relacionados con las interrupciones

### 6.3.1 *Configuración de las interrupciones en general*

- Configuración de la prioridad de excepciones: SHPRn
- Configuración de la prioridad de interrupciones: IPRn
- Leer estado de las interrupciones pendientes: lectura de ISPRn ó ICPRn
- Leer estado de interrupciones activas: lectura de IABRn
- Forzar el paso de Inactive a Pending escribiendo 1 en ISPRn
- Forzar el paso de Pending a Inactive escribiendo 1 en ICPRn
- Para enmascarar la activación de todas las interrupciones de prioridad configurable escribir un 1 en el registro PRIMASK.
- Para enmascarar la activación de todas las interrupciones excepto la NMI escribir un 1 en el registro FAULTMASK. El procesador borra el FAULTMASK al finalizar la atención de cualquier excepción, salvo de la NMI.
- Mediante la escritura del registro BASEPRI se define el nivel de prioridad máxima que es atendido.
- Para configurar el nivel de prioridad de interrupción en grupo y subgrupo se configura el campo PRIGROUP del registro AIRCR .

### 6.3.2 *Configuración de las interrupciones externas*

- Habilitación mediante registro ISER0
- Deshabilitación mediante registro ICER0
- El nivel de prioridad se configura en los registros IPR4 e IPR5
- Se consulta o modifica el estado de la interrupción en los registros IABR0, ISPR0 e ICPR0
- El tipo de interrupción externa (activo por nivel o por flanco) se configura con el registro EXTMODE
- El nivel activo (activa por nivel alto o bajo, o flanco de bajada o subida) se configura con el registro EXTPOLAR.
- La activación del evento asociado a la interrupción externa se consulta en el registro EXTINT.

### 6.3.3 *Configuración de las interrupciones asociadas a las entradas GPIO0 y GPIO2*

- Configurada como interrupción externa activa por flanco de subida en el registro IO0IntEnR e IO2IntEnR
- Configurada como interrupción externa activa por flanco de bajada en el registro IO0IntEnF e IO2IntEnF.
- Las interrupciones asociadas al GPIO0 y GPIO2 comparten en el NVIC los recursos con la Interrupción Externa 3 (/EINT3) con el número de interrupción IRQ21.
- El registro de un evento de flanco de subida se consulta en el registro IO0IntStatR e IO2IntStatR.
- El registro de un evento de flanco de bajada se consulta en el registro IO0IntStatF e IO2IntStatF.
- Para comprobar si hay alguna interrupción pendiente en GPIO0 o GPIO2 se puede consultar el registro IOIntStatus
- Para borrar los flags de interrupción asociados a cada pin de los puertos es necesario escribir un 1 en los registros IO0IntClr e IO2IntClr

## 6.4 Anexo IV: Introducción al CMSIS (Cortex Microcontroller Software Interface Standard)

Con el fin de facilitar y homogeneizar el diseño de aplicaciones para diferentes procesadores basados en ARM y favorecer el poder reusar código realizado para unos procesadores en otros, el propio ARM definió en 2008, una biblioteca de funciones que estandariza:

- La definición de los registros del Cortex-M relacionados con el NVIC, System Control Block, SYSTICK, MPU así como un conjunto de funciones de acceso al NVIC y otras funciones básicas.
- Estandariza los nombres de las excepciones.
- Introduce una función denominada SystemInit() que se encarga de inicializar todo el sistema. Especialmente incluye la inicialización de los osciladores.
- Proporciona un conjunto de funciones que facilita el acceso a funciones intrínsecas cuya implementación están fuera del C estándar y depende de cada compilador.
- Funciones básicas de comunicación serie, Ethernet y SPI.
- Forma estándar de configurar los osciladores para definir la frecuencia de reloj del procesador.
- Funciones estándar para configurar y acceder al SYSTIC, muy utilizado por los sistemas operativos.

Si los programadores utilizan las funciones CMSIS, su código puede ser reutilizable en otros procesadores de la familia ARM y con otro entorno de compilación diferente.

Para utilizar las funciones CMSIS es necesario incluir en los ficheros .C que las utilicen los archivos **core_cm3.h** y **system_LPC17xx.h** (ambos a su vez incluidos en LPC17xx.H) y en el proyecto, para algunas de las funciones, el fichero **core_cm3.c**. Los ficheros .H se encuentran en el directorio C:/Keil/CMSIS/INCLUDE y el fichero .C en el directorio C:/Keil/ARM/Startup/.

### *6.4.1 Listado de las funciones CMSIS*

#### *6.4.1.1 Funciones relacionadas con el acceso a los registros del Corex-M3:*

```
void __enable_irq (void)
Global Interrupt enable (PRIMASK = 0) using the instruction CPSIE i

void __disable_irq (void)
Global Interrupt disable (PRIMASK = 1 ) using the instruction CPSID i

uint32_t __get_CONTROL (void)
Return Control Register Value using the instruction MRS

void __set_CONTROL (uint32_t value)
Set CONTROL register value using the instruction MSR
```

```
uint32_t __get_IPSR (void)
Return IPSR Register Value using the instruction MRS

uint32_t __get_APSR (void)
Return APSR Register Value using the instruction MRS

uint32_t __get_xPSR (void)
Return xPSR Register Value using the instruction MRS

uint32_t __get_PSP (void)
Return Process Stack Pointer (PSP) using the instruction MRS

void __set_PSP (uint32_t TopOfProcStack)
Set Process Stack Pointer (PSP) value using the instruction MSR

uint32_t __get_MSP (void)
Return Main Stack Pointer (MSP) using the instruction MRS

void __set_MSP (uint32_t TopOfMainStack)
Set Main Stack Pointer (MSP) using the instruction MSR

uint32_t __get_PRIMASK (void)
Return Priority Mask Register (PRIMASK) using the instruction MRS

void __set_PRIMASK (uint32_t value)
Assign value to Priority Mask Register (PRIMASK) using the instruction MSR

void __enable_fault_irq (void)
Global Fault exception and Interrupt enable (FAULTMASK = 0) using the instruction CPSIE f

void __disable_fault_irq (void)
Global Fault exception and Interrupt disable (FAULTMASK = 1) using the instruction CPSID f

uint32_t __get_BASEPRI (void)
 Return Base Priority (BASEPRI) using the instruction MRS

void __set_BASEPRI (uint32_t value)
Set Base Priority (BASEPRI) using the instruction MSR

uint32_t __get_FAULTMASK (void)
Return Fault Mask Register (FAULTMASK) using the instruction MRS

void __set_FAULTMASK (uint32_t value)
Assign value to Fault Mask Register (FAULTMASK) using the instruction MSR
```

#### 6.4.1.2 Funciones relacionadas más concretamente con el NVIC

```
void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)
Set the Priority Grouping (Groups . Subgroups)

uint32_t NVIC_GetPriorityGrouping (void)
Get the Priority Grouping (Groups . Subgroups)

void NVIC_EnableIRQ (IRQn_Type IRQn)
Enable IRQn

void NVIC_DisableIRQ (IRQn_Type IRQn)
Disable IRQn

uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)
Return 1 if IRQn is pending else 0

void NVIC_SetPendingIRQ (IRQn_Type IRQn)
Set IRQn Pending

void NVIC_ClearPendingIRQ (IRQn_Type IRQn)
Clear IRQn Pending Status

uint32_t NVIC_GetActive (IRQn_Type IRQn)
Return 1 if IRQn is active else 0

void NVIC_SetPriority (  IRQn_Type IRQn,   uint32_t priority)
Set Priority for IRQn

uint32_t NVIC_GetPriority (IRQn_Type IRQn)
Get Priority for IRQn

uint32_t NVIC_EncodePriority (uint32_t PriorityGroup,  uint32_t PreemptPriority,
                              uint32_t SubPriority)
Encode priority for given group, preemptive and sub priority

void NVIC_DecodePriority (uint32_t Priority,  uint32_t PriorityGroup,  uint32_t* pPreemptPriority,
                          uint32_t* pSubPriority)
Decode given priority to group, preemptive and sub priority

void NVIC_SystemReset (void)
Resets the System
```

### 6.4.2 Ejemplo de programa que utiliza interrupciones externas usando CMSIS

A conticuación se presenta un ejemplo de programa que conmuta un LED conectado al puerto P1.29 con cada activación por flanco de bajada de la interrupción externa EINT3.

En este ejemplo se utilizan las funciones CMSIS para configurar el NVIC no siendo necesario definir los registros correspondientes y simplificando el programa. En el proyecto del programa se incluiría, los ficheros ***startup_LPC17xx**.s* y ***system_LPC17xx.c*** (C:\Keil\ARM\Startup\NXP\LPC17xx\) y, si se desea tener disponibles todas las funciones CMSIS (aunque en este caso no sería necesario) el fichero ***core_3m.c*** (C:\Keil\ARM\Startup\), aunque en este caso particular no sería necesario ya que las funciones que allí se implementan no se utilizan.

En negrita se muestran las funciones CMSIS.

```
// ******************************************************
//
// File Name: ConmutaLedPrincipal.c
//
// Purpose: Programa ejemplo que muestra el control del parpadeo de un LED
//          con una entrada de interrupciÛn
//              LED      --> P1.29
//              PULSADOR --> EINT3 (P2.13)
//
// ******************************************************

#include <LPC17xx.h>
#include "delay.h"

#define RETARDO 1000000   // Retardo

uint8_t activo = 1;      // Se utiliza el bit menos significativo
                         // para activar o no el parpadeo del LED


//Funcion de interrupcion EINT3
void EINT3_IRQHandler(void)
{
  // Borrar el falg de la EINT3 --> EXTINT.3
  LPC_SC->EXTINT = 1 << 3;

  // Actualiza la variable que activa o desactiva el parpadeo
  activo ++;
}


// Función de inicialización
void Config(void)
{
  // Configuracion del P2.13 como EINT3   --> PINSEL4.26 y PINSEL4.27
  LPC_PINCON->PINSEL4 |= 1 << (13*2);

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

  // Interrupción activa por flanco de bajada        --> EXTMODE.3
  LPC_SC->EXTMODE |= 1<< 3;

  // Configuramos prioridad 1 a la interrupcion EINT3 (IRQ21) --> IPR5
  NVIC_SetPriority(EINT3_IRQn, 0x01);

  // Habilitar la interrupcion EINT3  --> ISER0.21
  NVIC_EnableIRQ(EINT3_IRQn);
}


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

  // Programa principal
  while (1) {
    if ((activo & 1) == 1) {
           LPC_GPIO1->FIOPIN ^= 1<<29;
    }

    delay(RETARDO);
  }
}
```

```
// *******************************************************
// Fichero: Delay.C
//
//    Funciones de retardo
// *******************************************************

#include <LPC17xx.H>

void delay(uint32_t n)
{
  int32_t i;

  for(i=0;i<n;i++);
}
```

```
// *******************************************************
// Fichero: Delay.H
//
//    Funciones de retardo
// *******************************************************

extern void delay(int n);
```