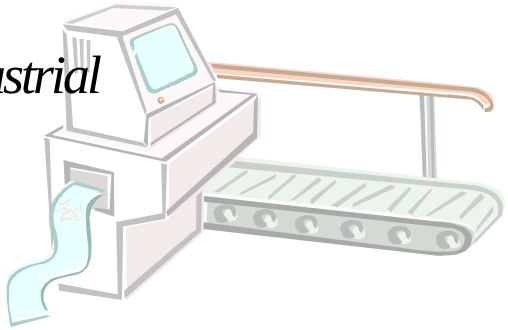
Página Web: https://web08.uah.es/
e-mail: eliseo.garcia@uah.es
Despacho: N-245
Departamento de Automática

Práctica 4

Interrupciones

Laboratorio de Informática Industrial



Interrupciones

Introducción

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 arquitectura Cortex y utilizado en la gestión de interrupciones.

Se incluyen en él 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.

Modelo de excepciones del Cortex M3

Al enfrentarse por primera vez a un microcontrolador es importante 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 en el LPC1768). (6.4 y 34.4.3.5 del LPC17xx User Manual).

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 (34.3.3 del LPC17xx User Manual):

- 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 como nivel de prioridad el 0. La prioridad de las excepciones (salvo las interrupciones) se configura con los registros SHPR1 – SHPR3 (34.4.3.9 del LPC17xx User Manual) y las de las interrupciones en los registros IPR0 – IPR27 (en otro sitio pone hasta IPR59) (34.4.2.7 y 6.5.11-19 del LPC17xx User Manual).

Las interrupciones pueden estar en tres estados: inactiva, pendiente y ejecutándose (inactive, pending, run) (34.4.2.9.1 del LPC17xx User Manual):

- 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 formar 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 (34.3.1.3.6 del LPC17xx User Manual):

- 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".

Si en un determinado momento, hay más de una interrupción en estado PENDING, se ejecutará la rutina de interrupción asociada a la interrupción más prioritaria (con menor 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 más 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 (34.4.3.6 y 34.3.3.6 del LPC17xx User

Manual). 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.

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.

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.

Interrupciones externas dedicadas

Las interrupciones externas /EINT0 - /EINT3 tienen asociados los números de interrupción IRQ18 – IRQ21 (Tabla 50 del LPC17xx User Manual) y, como el resto de las interrupciones, se configuran con el acceso a los registros generales del NVIC (6.5 del LPC17xx User Manual), 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 (3.6 del LPC17xx User Manual). 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.

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 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 (9.5.6 del LPC17xx User Manual).

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.

Introduccion 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 los archivos **core_cm3.h** y **system_LPC17xx.h** (ambos a su vez incluidos en LPC17xx.H) y en el proyecto el fichero **core_cm3.c**. Los ficheros .H se encuentran en el directorio C:/Keil/CMSIS/INCLUDE y el fichero .C en C:/Keil/ARM/Startup.

Listado de funciones CMSIS 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
```

Listado de funciones CMSIS 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
```

Ejemplo de programa que utiliza interrupciones externas usando CMSIS

A continuación 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*** y, para tener disponibles todas las funciones CMSIS (aunque en este caso no sería necesario) el fichero ***core_3m.c***, 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.

```
#include <LPC17xx.H>

// El programa conmuta un LED conectado al pin P1.29 con EINT3

// Funcion 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;

  // Configuracion 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
void main(void)
{
  //Funcion de inicialización
  Config();

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

  // Conmutar el LED  --> GPIO1 FIOPIN          P1.29
  LPC_GPIO1->FIOPIN ^= 1<<29;
}
```

Ejemplo de programa que utiliza interrupciones externas sin usar CMSIS

A continuación 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.

Este ejemplo no utiliza las funciones CMSIS para configurar el NVIC por lo que es necesario definir los registros necesarios al no encontrarse definidos en el LPC17xx.H.

Se presentan dos versiones, la primera en la que se definen los registros de la forma más sencilla posible y la segunda en la que se definen de forma similar a la utilizada en el LPC17xx.H.

Ejemplo con definición sencilla de registros

En negrita se presenta la definición de los registros del NVIC

```
#include <LPC17xx.H>

// El programa conmuta un LED conectado al pin P1.29

// Definicion del los registros IPR5 del NVIC
#define LPC_NVIC_IPR5 *((volatile unsigned int *) (0xE000E414UL))

// definicion del registros ISER0 del NVIC
#define LPC_NVIC_ISER0 *((volatile unsigned int *) (0xE000E100UL))

// Funcion 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;

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

   // Configuramos prioridad 1 a la interrupcion EINT3 (IRQ21) --> IPR5
   LPC_NVIC_IPR5 |= (0x01 << 3)<< (8 * 1);

   // Habilitar la interrupcion EINT3  --> ISER0.21
   LPC_NVIC_ISER0 = 1<<21;
}

// Programa principal
void main(void)
{

   //Funcion de inicialización
   Config();

   // Programa principal
   while (1);
}

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

   // Conmutar el LED  --> GPIO1 FIOPIN          P1.29
   LPC_GPIO1->FIOPIN ^= 1<<29;
}
```

Ejemplo con definición de registros similar a la utilizada en LPC17xx.H

En negrita se presenta la definición de los registros del NVIC

```
#include <LPC17xx.H>

// El programa conmuta un LED conectado al pin P1.29

// Definicion de los registros IPR del NVIC
typedef struct{
  __IO uint32_t IPR[28];
} LPC_NVIC_IPR_TypeDef;

#define LPC_NVIC_IPR  ((LPC_NVIC_IPR_TypeDef*) (0xE000E400UL))

// definicion de los registros ISER del NVIC
typedef struct{
  __IO uint32_t ISER0;
  __IO uint32_t ISER1;
  __IO uint32_t ISER2;
  __IO uint32_t ISER3;
} LCP_NVIC_ISER_TypeDef;

#define LPC_NVIC_ISER (( LCP_NVIC_ISER_TypeDef*) (0xE000E100UL))

// Funcion 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;

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

  // Configuramos prioridad 1 a la interrupcion EINT3 (IRQ21) --> IPR5
  LPC_NVIC_IPR5 |= (0x01 << 3)<< (8 * 1);

  // Habilitar la interrupcion EINT3  --> ISER0.21
  LPC_NVIC_ISER0 = 1<<21;
}

// Programa principal
void main(void)
{

  //Funcion de inicialización
  Config();

  // Programa principal
  while (1);
}

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

  // Conmutar el LED  --> GPIO1 FIOPIN          P1.29
  LPC_GPIO1->FIOPIN ^= 1<<29;
}
```

Resumen de Directivas de ensamblador

En este apartado se ofrece un resumen de las principales directivas de ensamblador que se utilizan en el desarrollo de programas para ensamblador

name* EQU *expr

La directiva EQU asigna un nombre simbólico a una constante. Por ejemplo, se puede emplear para asignar el nombre STACK_TOP al valor 0x10004000, mediante la línea STACK_TOP EQU 0x10004000.

AREA *section*{,*attr*}...

La directiva AREA permite definir distintas secciones de código o datos. El parámetro section corresponde con un nombre que el programador asigna a la sección. Este nombre opcionalmente puede ir entre barras '|'. A continuación puede llevar una serie de atributos separados por comas que definen el tipo de sección. Algunos de los más habituales son:

- CODE: Define una sección de código que contendrá código máquina.
- DATA: Define una sección de datos.
- READONLY: Indica que la sección es de sólo lectura.
- READWRITE: Indica que la sección es de lectura y escritura.

Por ejemplo:

AREA RESET, CODE Define un área denominada *RESET* para código.
AREA Seccion_datos, DATA Define un área denominada *Seccion_datos* para datos.

{*label*} DCD *expr*{,*expr*}

Define datos en memoria. Puede manejar tanto valores como etiquetas. A conrinuacion se muestran varios ejemplos donde se puede emplear

DCD STACK_TOP Define un dato al que asigna el valor de la etiqueta STACK_TOP. Es la primera directiva DCD que aparece en la sección de código por lo que asignará el valor en las direcciones 0x0000_0000 a 0x0000_0003.

DCD start Define un dato al que asigna el valor de la etiqueta start. Esta etiqueta contiene la dirección de la primera instrucción del programa, que se ejecutará tras un reset. Almacena dicho valor en las direcciones 0x0000_0004 a 0x0000_0007.

SPACE

Reserva una cantidad de memoria que se introduce a continuación

SPACE 0x00000010 Reserva 16 bytes de memoria

ENTRY

La directiva ENTRY declara un punto de entrada a un programa.

END

Informa al ensamblador que se ha llegado al final del código fuente.

A continuación se muestra parte del código del fichero Startup_ARMCM3.s, escrito en ensamblador.

```
;//-------- <<< Use Configuration Wizard in Context Menu >>> ------------------

; <h> Stack Configuration
;   <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>
; </h>

Stack_Size      EQU     0x00000400

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp

Heap_Size       EQU     0x00000C00

                AREA    HEAP, NOINIT, READWRITE, ALIGN=3
__heap_base
Heap_Mem        SPACE   Heap_Size
__heap_limit

                PRESERVE8
                THUMB

; Vector Table Mapped to Address 0 at Reset

                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  __Vectors_End
                EXPORT  __Vectors_Size

__Vectors       DCD     __initial_sp              ; Top of Stack
                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            ;  0:  Watchdog Timer
                DCD     RTC_IRQHandler            ;  1:  Real Time Clock
                DCD     TIM0_IRQHandler           ;  2:  Timer0 / Timer1
                DCD     TIM2_IRQHandler           ;  3:  Timer2 / Timer3
                DCD     MCIA_IRQHandler           ;  4:  MCIa
                DCD     MCIB_IRQHandler           ;  5:  MCIb
                DCD     UART0_IRQHandler          ;  6:  UART0 - DUT FPGA
                DCD     UART1_IRQHandler          ;  7:  UART1 - DUT FPGA
                DCD     UART2_IRQHandler          ;  8:  UART2 - DUT FPGA
                DCD     UART4_IRQHandler          ;  9:  UART4 - not connected
                DCD     AACI_IRQHandler           ; 10: AACI / AC97
                DCD     CLCD_IRQHandler           ; 11: CLCD Combined Interrupt
                DCD     ENET_IRQHandler           ; 12: Ethernet
                DCD     USBDC_IRQHandler          ; 13: USB Device
                DCD     USBHC_IRQHandler          ; 14: USB Host Controller
                DCD     CHLCD_IRQHandler          ; 15: Character LCD
                DCD     FLEXRAY_IRQHandler        ; 16: Flexray
                DCD     CAN_IRQHandler            ; 17: CAN
                DCD     LIN_IRQHandler            ; 18: LIN
                DCD     I2C_IRQHandler            ; 19: I2C ADC/DAC
                DCD     0                         ; 20: Reserved
                DCD     0                         ; 21: Reserved
```

```
            DCD    0                    ; 22: Reserved
            DCD    0                    ; 23: Reserved
            DCD    0                    ; 24: Reserved
            DCD    0                    ; 25: Reserved
            DCD    0                    ; 26: Reserved
            DCD    0                    ; 27: Reserved
            DCD    CPU_CLCD_IRQHandler      ; 28: Reserved - CPU FPGA CLCD
            DCD    0                    ; 29: Reserved - CPU FPGA
            DCD    UART3_IRQHandler         ; 30: UART3    - CPU FPGA
            DCD    SPI_IRQHandler           ; 31: SPI Touchscreen - CPU FPGA
__Vectors_End

__Vectors_Size  EQU    __Vectors_End - __Vectors

            AREA    |.text|, CODE, READONLY


; Reset Handler

Reset_Handler  PROC
            EXPORT  Reset_Handler           [WEAK]
            IMPORT  SystemInit
            IMPORT  __main
            LDR     R0, =SystemInit
            BLX     R0
            LDR     R0, =__main
            BX      R0
            ENDP


; Dummy Exception Handlers (infinite loops which can be modified)

NMI_Handler    PROC
            EXPORT  NMI_Handler             [WEAK]
            B       .
            ENDP
HardFault_Handler\
            PROC
            EXPORT  HardFault_Handler       [WEAK]
            B       .
            ENDP
MemManage_Handler\
            PROC
            EXPORT  MemManage_Handler       [WEAK]
            B       .
            ENDP
BusFault_Handler\
            PROC
            EXPORT  BusFault_Handler        [WEAK]
            B       .
            ENDP
UsageFault_Handler\
            PROC
            EXPORT  UsageFault_Handler      [WEAK]
            B       .
            ENDP
SVC_Handler    PROC
            EXPORT  SVC_Handler             [WEAK]
            B       .
            ENDP
DebugMon_Handler\
            PROC
            EXPORT  DebugMon_Handler        [WEAK]
            B       .
            ENDP
PendSV_Handler  PROC
            EXPORT  PendSV_Handler          [WEAK]
            B       .
            ENDP
SysTick_Handler PROC
            EXPORT  SysTick_Handler         [WEAK]
            B       .
            ENDP
```

Practica Propuesta

En la presente practica se propone añadir más funcionalidad al programa desarrollado en la practica 3. Así, se añadirán una serie de interrupciones al código para implementar el siguente comportamiento

- Cuando se produzca un flanco de bajada en el pin 18 del conector J14, se conmutara la salida en el conector J13 entre binario y BCD (si en el momento del flanco se encuentra en binario, se pasara a BCD y viceversa).

- Cuando se produzca un flanco de bajada en el pin 19 del conector J14 se cogerá el valor mínimo desde el que se debe realizar la búsqueda de números primos (que se encontrara en los pines descritos en la práctica anterior), y cuando se produzca un flanco de subida en dicho pin, el valor que se obtenga en los pines será considerado como el valor máximo de la búsqueda. Se garantiza en las especificaciones que nunca habrá dos flancos en un tiempo inferior a 50 mseg. y que siempre, tras un reset o una pulsación de SW2, se cogerá en primer lugar el valor minimo y, posteriormente, el valor máximo, en el caso de que se modifique el valor por defecto (65535).

- Cuando se produzca un flanco de bajada pin 17 del conector J14 se cambiara el orden en el que se calculan los números primos. Es decir, si en un momento determinado se están calculando y mostrando los números primos desde el 2 hasta el 65535 en orden ascendente, tras el flanco de bajada se calcularan en orden descendente.

- La temporización se deberá realizar mediante la programación del Timer SYSTICK y no mediante la función delay() que se utilizaba en la practica anterior.

El resto de funcionalidad de la practica anterior no se debe modificar (valores del BCD, comportamiento de los diodos,…)

Además, se pide que se introduzca en la memoria de la práctica la información que se puede extraer del fichero Startup_ARMCM3.s mostrado en la sección "Resumen de Directivas de Ensamblador". Para ello, resulta útil, es recomendable conocer el significado de las directivas que se muestran en ese apartado.

Agradecimientos

La presente practica ha sido realizada utilizando material cedido por profesores del Departamento de Electronica de la Universidad de Alcalá.