

Gestión de interrupciones en microcontroladores ARM Cortex-M

Apellidos, nombre	Capella Hernández, Juan Vicente (jcapella@disca.upv.es)
Departamento	Depto. de Informática de Sistemas y Computadores (DISCA)
Centro	Universitat Politècnica de València

1 Resumen de las ideas clave

En este artículo se introduce al lector en la gestión de interrupciones de los microcontroladores ARM Cortex-M. De forma que sea capaz de configurar de forma básica el sistema de interrupciones y definir manejadores de interrupción. Se mostrarán y explicarán a lo largo del artículo ejemplos en lenguaje C utilizándose el entorno de desarrollo Keil uVision.

2 Introducción

El sistema de interrupciones nos permitirá desarrollar programas que aprovechen la potencia de los microcontroladores ARM Cortex-M [1], dado que hará posible liberar la CPU de realizar determinadas tareas (p.e muestrear periféricos periódicamente), ya que cuando necesiten ser atendidos lanzarán una interrupción, de forma que la CPU pasará a ejecutar el manejador de interrupción correspondiente (dejando lo que estuviera haciendo en ese momento, véase figura 1). De esta forma ya no será necesario colocar en el programa principal bucles vacíos en espera de que se pulse un botón o que cambie el estado de un periférico pudiendo dedicar la CPU a tareas más provechosas.

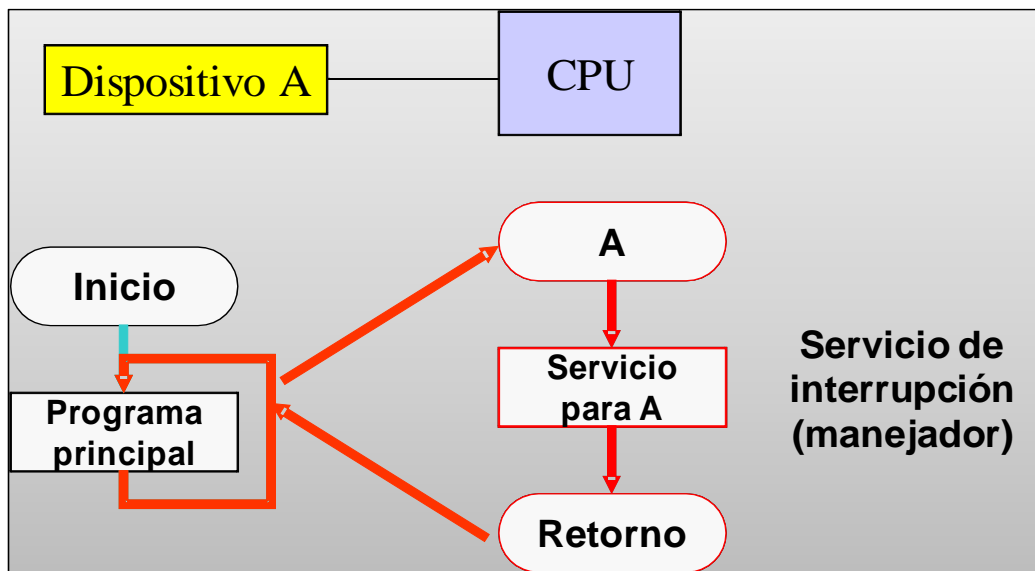


Imagen 1. Funcionamiento interrupción

En este artículo nos centraremos en un tipo concreto de interrupciones que son las interrupciones externas, es decir, las provocadas por fuentes externas al microcontrolador [2].

3 Objetivos

Una vez que el alumno se lea con detenimiento este documento, será capaz de:

- Conocer el concepto de interrupción.
- Diferenciar las fuentes de interrupción.
- Configurar el sistema de interrupciones.
- Sintetizar manejadores de interrupción.

4 Desarrollo

A continuación se desarrollarán cada uno de los aspectos indicados en la introducción y objetivos, realizando las explicaciones de la forma más práctica y guiada posible.

4.1 Configuración del sistema de interrupciones

En primer lugar tendremos que activar y configurar adecuadamente el puerto GPIO donde se encuentre el pin donde conectaremos la fuente externa de interrupción, esto lo haremos tal y como hemos hecho en prácticas anteriores. Para ello daremos primero señal de reloj al puerto implicado, llamando adecuadamente a la función: `void RCC_AHB1PeriphClockCmd (uint32_t RCC_AHB1Periph, FunctionalState NewState)`. A continuación tendremos que indicar la configuración que queremos darle a dicho puerto rellenando los campos vistos en clase de la estructura llamada `GPIO_InitStructure` de tipo `GPIO_InitTypeDef`, configurando el pin donde conectemos la fuente de interrupción como entrada (`GPIO_Mode_IN`) y desactivando las resistencias de pull up y pull down (`GPIO_PuPd_NOPULL`), para finalmente invocar la función `void GPIO_Init (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)`, donde indicaremos como primer parámetro el puerto GPIO que estamos configurando y como segundo la estructura que habremos rellenado en el paso anterior.

Por otro lado, tendremos que activar dando reloj al sistema de interrupciones con la instrucción:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
```

Y a continuación configurar el sistema de interrupciones adecuadamente para nuestros propósitos. En el caso que nos ocupa tendremos que conectar la línea 0 de interrupción externa al pin 0:

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
```

Ahora configuraremos la línea 0, habilitándola en modo interrupción por flanco de subida (se precisará tener declarada una estructura llamada `EXTI_InitStructure` de tipo `EXTI_InitTypeDef`):

```
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;  
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;  
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01;  
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;  
NVIC_Init(&NVIC_InitStructure);
```

Podemos consultar en los manuales de programación ARM Cortex-M [3] todas los campos de estas estructuras que estamos manejando, así como los posibles valores definidos que podemos asignar.

Finalmente, toda esta configuración sería recomendable encapsularla dentro de una función de configuración para facilitar la comprensión y mantenimiento del código de nuestros proyectos, como se muestra en la figura 1.

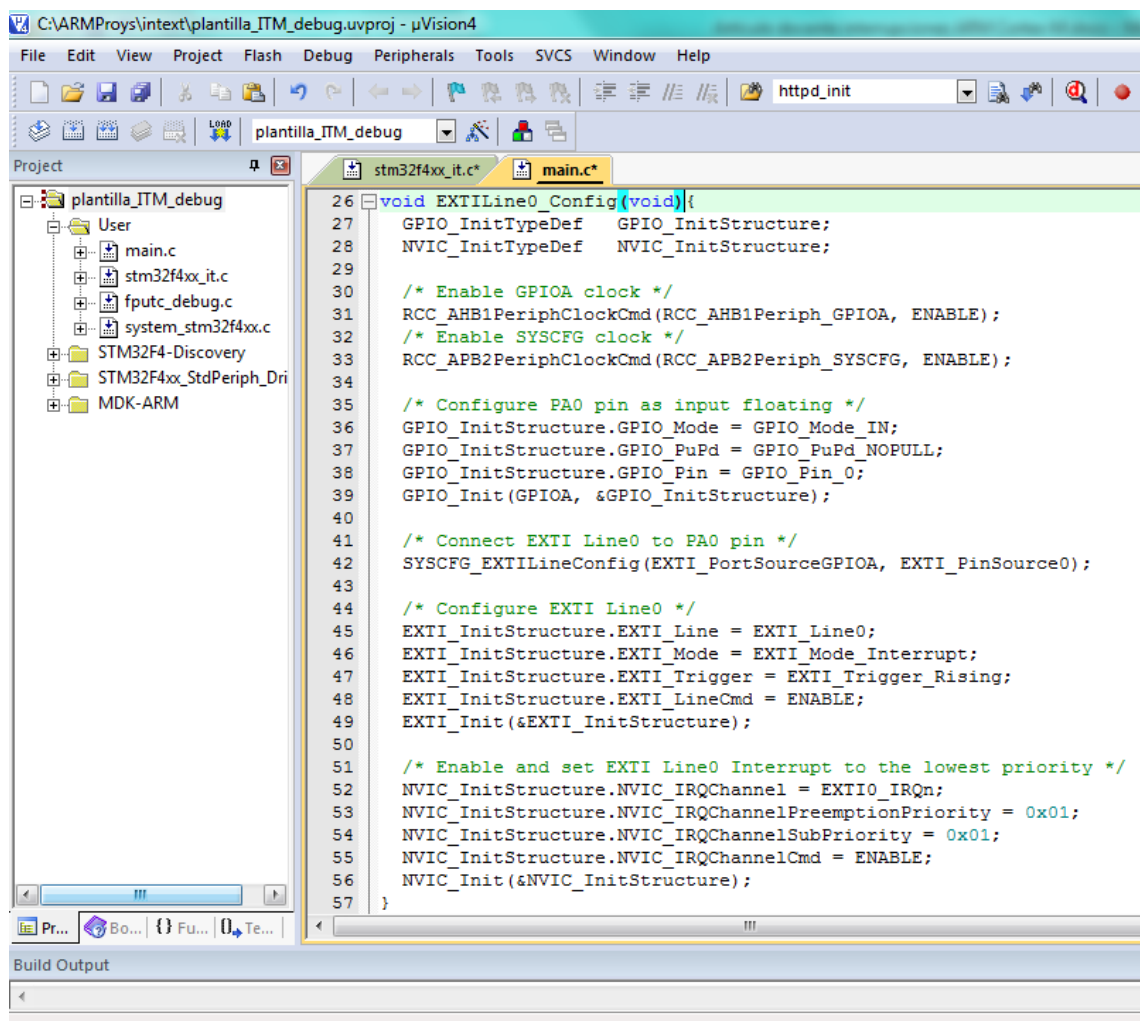
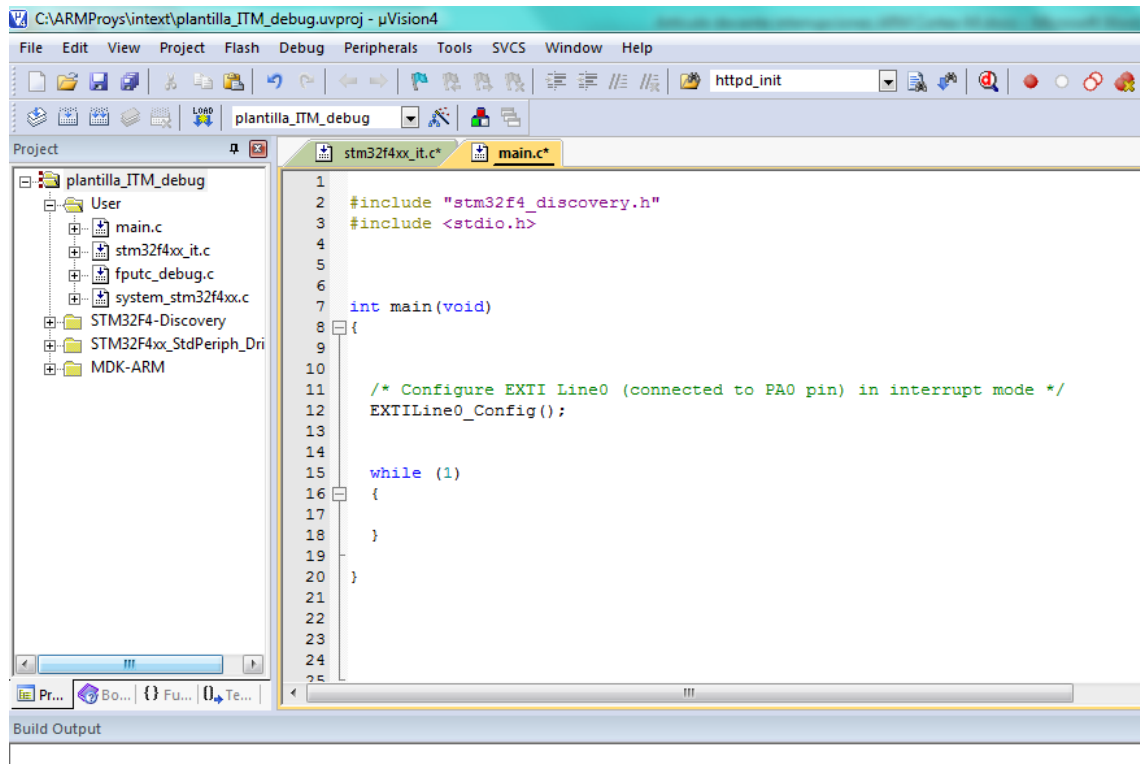


Imagen 1. Función de configuración

La función de configuración deberá ser invocada desde el programa principal main() antes de entrar en el bucle while(1), como se puede observar en la figura 2. Dicho bucle while(1) del main, podría quedar vacío al utilizar interrupciones si la funcionalidad la programamos en los manejadores de interrupción.



```
1
2 #include "stm32f4_discovery.h"
3 #include <stdio.h>
4
5
6
7 int main(void)
8 {
9
10
11 /* Configure EXTI Line0 (connected to PA0 pin) in interrupt mode */
12 EXTI_Line0_Config();
13
14
15 while (1)
16 {
17
18 }
19
20 }
21
22
23
24
25
```

Imagen 2. Programa principal resultante al utilizar interrupciones

4.2 Programando los manejadores de interrupción

Una vez configurado apropiadamente el sistema de interrupciones, solo nos queda programar el/los manejadores de la/s interrupción/es que puedan producirse, es decir, especificar qué debe hacerse (qué instrucciones ejecutar) cuando se produzca una interrupción de un tipo u otro. Por tanto los manejadores de interrupción no son más que funciones que se ejecutan en respuesta a interrupciones.

En nuestro caso, tendremos que programar el manejador de la interrupción externa 0. Para ello deberemos localizar en el módulo "stm32f4xx_it.c" del proyecto una función con cabecera: void EXTI0_IRQHandler(void) (figura 3) y darle implementación de acuerdo con el diseño de nuestro sistema (puede ser incrementar una variable, poner un determinado valor en una variable para indicar algo, llamar a una función, etc. recomendándose en general incluir en el manejador solamente las instrucciones estrictamente necesarias, para facilitar que la CPU pueda volver cuanto antes al programa principal).

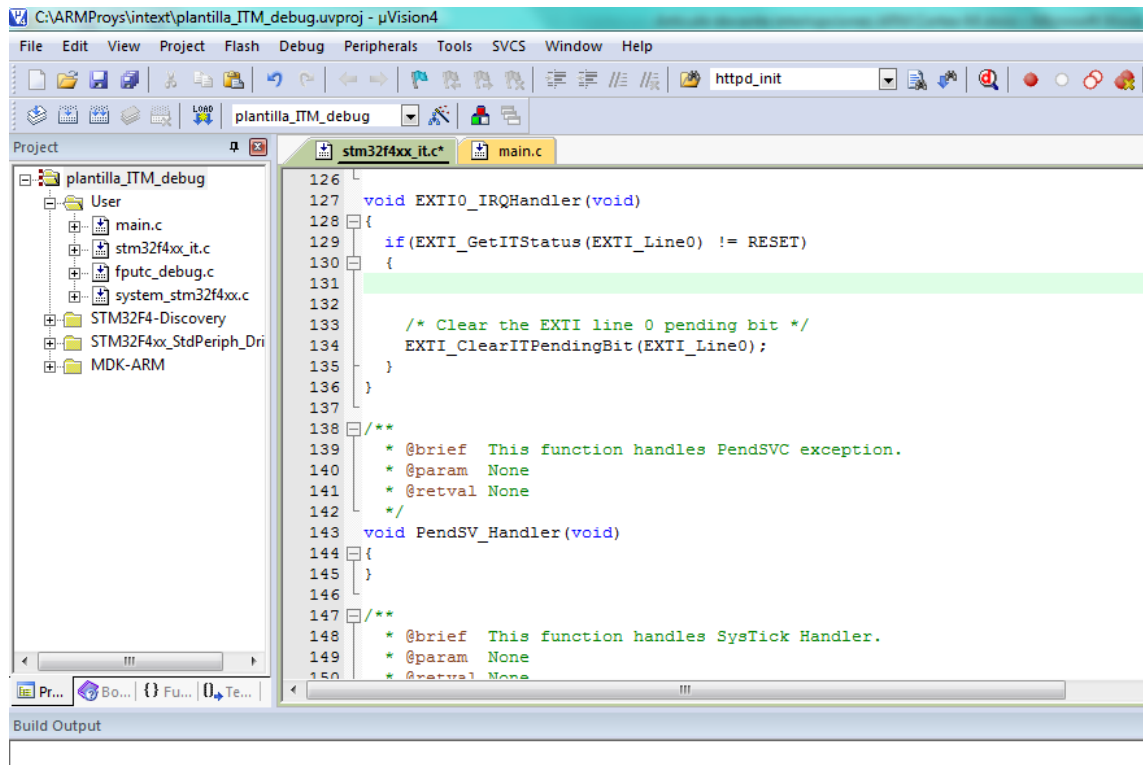


Imagen 3. Manejador de la interrupción externa EXTI0

En la figura 4 se muestra parcialmente el fichero "startup_stm32f4xx.s", en el que podemos encontrar el listado completo de las funciones predefinidas como manejadores de interrupción.

```

_Vectors      DCD     __initial_sp           ; Top of Stack
_Vectors      DCD     Reset_Handler        ; Reset Handler
_Vectors      DCD     NMI_Handler          ; NMI Handler
_Vectors      DCD     HardFault_Handler    ; Hard Fault Handler
_Vectors      DCD     MemManage_Handler    ; MPU Fault Handler
_Vectors      DCD     BusFault_Handler     ; Bus Fault Handler
_Vectors      DCD     UsageFault_Handler   ; Usage Fault Handler
_Vectors      DCD     0                    ; Reserved
_Vectors      DCD     0                    ; Reserved
_Vectors      DCD     0                    ; Reserved
_Vectors      DCD     0                    ; Reserved
_Vectors      DCD     SVC_Handler          ; SVCcall Handler
_Vectors      DCD     DebugMon_Handler     ; Debug Monitor Handler
_Vectors      DCD     0                    ; Reserved
_Vectors      DCD     PendSV_Handler       ; PendSV Handler
_Vectors      DCD     SysTick_Handler      ; SysTick Handler

; External Interrupts
_Vectors      DCD     WWDG_IRQHandler      ; Window WatchDog
_Vectors      DCD     PVD_IRQHandler       ; PVD through EXTI Line detection
_Vectors      DCD     TAMPER_IRQHandler    ; Tamper and TimeStamps through the EXTI line
_Vectors      DCD     RTC_WKUP_IRQHandler  ; RTC Wakeup through the EXTI line
_Vectors      DCD     FLASH_IRQHandler     ; FLASH
_Vectors      DCD     RCC_IRQHandler       ; RCC
_Vectors      DCD     EXTI0_IRQHandler     ; EXTI Line0
_Vectors      DCD     EXTI1_IRQHandler     ; EXTI Line1
_Vectors      DCD     EXTI2_IRQHandler     ; EXTI Line2

```

Imagen 4. Vista parcial del fichero startup_stm32f4xx.s

5 Cierre

A lo largo de este objeto de aprendizaje hemos tratado las cuestiones básicas relacionadas con la configuración y gestión de interrupciones en microcontroladores ARM Cortex-M.

Para comprobar que realmente has aprendido las bases del sistema de interrupciones es el momento de que te pongas manos a la obra e intentes crear un proyecto donde configures el sistema de interrupciones como se ha explicado a lo largo del artículo y programes el manejador de la interrupción externa de forma que cuando se produzca dicha interrupción cambies de estado una salida digital (p.e para apagar/encender un led).

¡¡ÁNIMO!!.

6 Bibliografía

[1] ARM Limited. Cortex-M4 technical reference manual, 2010. URL: <http://www.arm.com/>

[2] J Yiu: The Definitive Guide to ARM® Cortex®-M3 and Cortex-M4 Processors, 3rd Edition, 2013.

[3] Microelectronics, St. STM32F3xxx and STM32F4xxx Cortex-M4 programming Manual, 2013. URL: http://www.st.com/web/en/resource/technical/document/programming_manual/DM00046982.pdf