UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Interrupciones con el periférico EXTI en los microcontroladores STM32F4 de la arquitectura ARM Cortex-M

Apellidos, nombre	Perles Ivars, Àngel (aperles@disca.upv.es)
Departamento	Informática de Sistemas y Computadores
Centro	Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Resumen de las ideas clave

Se presenta el uso del periférico EXTI (*External interrupt/event controller*) disponible en los microcontroladores STM32F4 de la empresa StMicroelectronics que siguen la arquitectura ARM Cortex-M. A continuación se resumen las ideas clave que se van a trabajar:

- Propósito del periférico EXTI para la detección de cambios en señales digitales.
- Configuración del subsistema de interrupciones para el aprovechamiento del EXTI.
- Opciones de planteamientos para crear un manejador de interrupción adecuado.

2 Introducción

Una necesidad típica de los sistemas basados en microcontrolador es la detección eficiente de cambios en señales digitales. Ejemplos típicos de esta necesidad es la atención a botoneras, pulsadores, alarmas, finales de carrera, etc.

El plantear la atención a estas señales mediante técnica de encuesta o *polling* es muy ineficiente desde el punto de vista de aprovechamiento de la CPU (desperdicio de tiempo) y de la energía (modos de bajo consumo y encendido por evento).

Cada microcontrolador particular que se precie, suele aportar un periférico dedicado a este propósito. En el caso de los microcontroladores STM32F4 de la empresa StMicroelectronics, se dispone del periférico EXTI, que está diseñado para resolver esta problemática de manera eficiente.

Aquí se propone aprovechar EXTI mediante el servicio de interrupciones. El mecanismo de interrupción permite que las necesidades de los periféricos y los eventos importantes sean notificados a la CPU y los pueda atender inmediatamente.

3 Objetivos

Una vez leído y practicado este documento, el lector será capaz de:

- Configurar los periféricos EXTI y relacionarlos con otros periféricos.
- Configurar el sistema de interrupciones para usar EXTI.
- Crear manejadores básicos para aprovechar EXTI.

4 Requisitos previos

Para poder asimilar los conceptos, son necesarios estos requisitos:

- Programación en lenguaje C para sistemas embebidos.

- Electrónica digital básica.
- Estándar CMSIS de la arquitectura ARM Cortex-M.
- Interrupciones en la arquitectura ARM Cortex-M.

5 Desarrollo

5.1 El periférico EXTI

El EXTI consiste en una serie de detectores de flancos capaces de generar eventos o interrupciones. Estos detectores están conectados a pines de los puertos (de EXTI0 a EXTI15) o a elementos como la alarma del reloj de tiempo real, el evento de despertar del puerto Ethernet, etc. En el manual de referencia del fabricante [1] se describe ampliamente este elemento.

El EXTI se puede configurar para que genere evento/interrupción cuando detecta flanco de subida, de bajada o ambos.

Para su programación, se puede recurrir a las funciones del firmware de St contenidas en el módulo *stm32f4xx_exti.c* y documentadas en [2]. En la tabla 1se resumen estas funciones.

Tabla 1. Funciones CMSIS para gestionar EXTI.

`void EXTI_DeInit (void);` Desinicializa los registros del periférico poniéndolos al valor por defecto después de RESET.
`void EXTI_Init (EXTI_InitTypeDef *EXTI_InitStruct);` Inicializa el periférico EXTI de acuerdo con los parámetros establecidos en EXTI_InitStruct.
`void EXTI_StructInit (EXTI_InitTypeDef *EXTI_InitStruct);` Rellena EXTI_InitStruct con los valores por defecto de RESET.
`void EXTI_GenerateSWInterrupt(uint32_t EXTI_Line);` Generar un interrupción software sobre la línea especificada.
`FlagStatus EXTI_GetFlagStatus (uint32_t EXTI_Line);` Comprueba si el flag/indicador asociado a la línea EXTI está establecido o no.
`void EXTI_ClearFlag (uint32_t EXTI_Line);` Limpia el flag/indicador de la línea EXTI indicada.
`ITStatus EXTI_GetITStatus (uint32_t EXTI_Line);` Comprueba si la línea EXTI indicada está activada o no.
`void EXTI_ClearITPendingBit (uint32_t EXTI_Line);` Limpiar el bit de petición pendiente del la línea EXTI.

5.2 Configuración de las interrupciones

Para configurar el sistema de interrupciones se deben seguir sistemáticamente los pasos típicos, que son:

1. Desconectar sistema de interrupciones.

2. Programar el dispositivo fuente.
3. Habilitar interrupción fuente.
4. Conectar sistema de interrupciones.

Finalizado el paso 4, el sistema está listo para que se produzcan las interrupciones.

Como ejemplo, imagínese que se desea que el pulsador azul de la placa STM32F4Discovery de St genere una interrupción asociado al sistema EXTI. Como el pulsador está conectado al pin PA0, se debe seleccionar obligatoriamente EXTI0 para este propósito.

Para configurar todo el sistema, se propone el siguiente listado:

```
#include "stm32f4xx.h"

int main(void)
{
   EXTI_InitTypeDef   EXTI_InitStructure;
   GPIO_InitTypeDef   GPIO_InitStructure;
   NVIC_InitTypeDef   NVIC_InitStructure;

   /* PASO 1: Disable interrupt */

   NVIC_DisableIRQ(EXTI0_IRQn);

   /* PASO 2: Configure target device */

   /* Enable GPIOA clock */
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
   /* Configure PA0 pin as input floating */
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
   GPIO_Init(GPIOA, &GPIO_InitStructure);

   /* Configure EXTI Line0 (connected to PA0 pin) in interrupt
mode */
   /* Enable SYSCFG clock */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
   /* Connect EXTI Line0 to PA0 pin */
```

```
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);

    /* Configure EXTI Line0 */
    EXTI_InitStructure.EXTI_Line = EXTI_Line0;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* PASO 3: Configure NVIC related interrupt */

    /* Enable and set EXTI Line0 Interrupt to the lowest priority
*/
    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);

    /* PASO 4: Enable interrupt */

    NVIC_EnableIRQ(EXTI0_IRQn);

    /* SUPER-LOOP */
    while (1)
    {
       //forever tasks!
    }
}
```

Obsérvese como se siguen los pasos típicos de configuración de un sistema de interrupciones. En el paso 2 correspondiente a la configuración de periféricos, se configura primero el pin adecuadamente, a continuación se enlaza el pin con el periférico EXTI0 y, finalmente, se configura EXTI0 para que responda al estímulo deseado.

5.3 El servicio de interrupción

Para dar servicio a la interrupción, será necesario proporcionar el manejador.

El manejador será una simple función C con un nombre predefinido. En este caso, la función se deberá llamar *void EXTI0_IRQHandler(void)* para EXTI0, y debería colocarse en el archivo *stm32f4xx_it.c*.

En el siguiente listado se muestra una propuesta de manejador.

```
void EXTI0_IRQHandler(void)
{
  if(EXTI_GetITStatus(EXTI_Line0) != RESET)
  {
    // Do work

    /* Clear the EXTI line 0 pending bit */
    EXTI_ClearITPendingBit(EXTI_Line0);
  }
}
```

Obsérvese, que es necesario comprobar qué fuente es la que ha generado la interrupción y, tras la acción asociada al servicio, limpiar el flag indicador correspondiente. Este patrón será típico en la mayoría de interrupciones asociadas a periféricos. Se denomina *borrado por software* del flag de petición en contraposición a los casos en que el flag de petición es borrado automáticamente al servirse la interrupción.

Ahora queda rellenar el manejador con la tarea encomendada.

5.3.1 Haciendo el trabajo en el manejador

En el propio manejador puede ejecutarse la acción deseada. Recuérdese que este aspecto es tremendamente crítico para la correcta operación del sistema, pues la interrupción detiene momentáneamente el resto del sistema.

En un microcontrolador como los ARM Cortex-M se pueden ejecutar aplicaciones complejas con muchísimas interrupciones que se interfieren unas a otras y al bucle principal, por tanto, el código a ejecutar en el manejador debe ser la más rápidas y eficiente posible, con lo que se intenta minimizar este efecto.

Como ejemplo de ejecución en el manejador, el siguiente manejador permite cambiar el estado de un LED.

```
#include <led.h>

// ...
```

```
void EXTI0_IRQHandler(void)
{
  static uint8_t phase = 0;

  if(EXTI_GetITStatus(EXTI_Line0) != RESET)
  {
    /* Toggle LED */
    if (phase == 0) {
        led_on();
        phase = 1;
    } else {
        led_off();
        phase = 0;
    }

    /* Clear the EXTI line 0 pending bit */
    EXTI_ClearITPendingBit(EXTI_Line0);
  }
}
```

5.3.2 División en dos niveles

Una práctica habitual con las interrupciones consiste en dividir una tarea en dos niveles. Para ello, la interrupción atiende lo inmediato y anota que ha habido una petición, y el bucle de tareas periódicas atiende, por consulta, el trabajo pesado. Se trata de combinar las ventajas de la interrupción con la técnica de *polling*.

Como ejemplo, imagínese un reproductor de CDs con sus botones "play"',... . Para la llamada "experiencia de usuario"', no hay nada más frustrante que una botonera medio sorda, por lo que es prioritario atender a los botones; por otra parte, atender continuamente a los botones consume un precioso tiempo de CPU que es necesario invertir en otras cosas.

Para implementar esta idea, véase el siguiente manejador en el que la única acción será cambiar el estado de la variable.

```
extern uint8_t _button_state;

void EXTI0_IRQHandler(void)
{
```

```
  if(EXTI_GetITStatus(EXTI_Line0) != RESET)
  {
    _button_state = 1;  // button pressed

    /* Clear the EXTI line 0 pending bit */
    EXTI_ClearITPendingBit(EXTI_Line0);
  }
}
```

Destacar el uso ineludible del atributo *volatile* en la variable compartida.

El atributo *extern* para la variable, que permite el acceso a una misma variable desde varios módulos. Esta es una práctica de programación poco recomendable, pero a la que obligan las necesidades de este tipo de aplicaciones.

En el siguiente listado se muestra un fragmento de programa principal en el que se aportan funciones para usar una variable compartida entre el bucle principal y un manejador de interrupción. Las modificaciones de la variable se comprueban, cuando se pueda, en el superbucle.

```
#include <stdio.h>
#include "stm32f4xx.h"
#include <led.h>

volatile uint8_t _button_state = 0;

uint8_t button_GetPressed(void)
{
   return _button_state;
}

void button_Reset(void)
{
   _button_state = 0;   // not pressed
}

int main(void)
{
   // ...
```

```
        while (1)
        {
           // check if the button was pressed in the past
           if (button_GetPressed()) {
              printf("Se ha pulsado el botoncito\n");
              button_Reset();
           }

           // simulate other tasks
           for (i=0;i<100000000;i++) {};
           printf("Tarari!\n");

        }; //forever i que no siga res!

    }
```

6 Cierre

A lo largo de este objeto de aprendizaje se ha visto cómo aprovechar adecuadamente el periférico EXTI de los microcontroladores STM32F4 de la empresa StMicroelectronics.

Como se ha visto, la configuración del periférico para generar una interrupción sigue pasos sistemáticos en los que la diferencia respecto a otros periféricos es la configuración propiamente dicha del periférico.

Se describe también la manera de aprovechar adecuadamente el servicio de interrupción como parte integrante de una aplicación más compleja.

7 Bibliografía

[1] StMicroelectronics. RM0090 Reference manual for STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx. 2014. url: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf .

[2] StMicroelectronics. St STM32F4 DSP and standard peripherals library (copia local generada a partir del doxygen). 2013. url: http://www.disca.upv.es/aperles/arm_cortex_m3/curset/STM32F4xx_DSP_StdPeriph_Lib_V1.0.1/html/index.html .