Taller de Sistemas Embebidos
STM32 MCU - Interrupts

Información relevante

Taller de Sistemas Embebidos

Asignatura correspondiente a la **actualización 2023** del Plan de Estudios 2020 y resoluciones modificatorias, de Ingeniería Electrónica de FIUBA

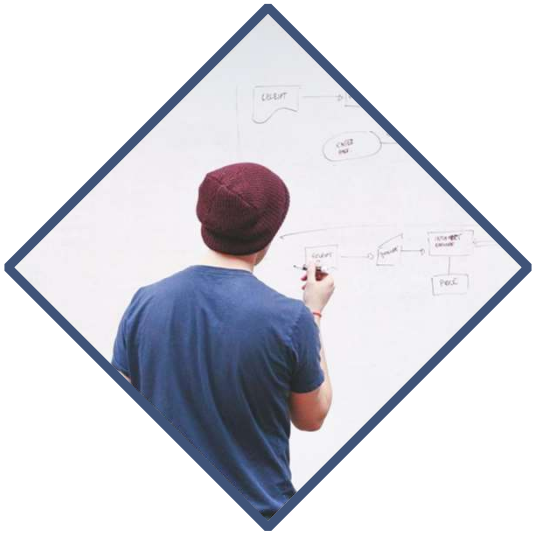
Estructura Curricular de la Carrera

El **Proyecto Intermedio** se desarrolla en la asignatura **Taller de Sistemas Embebidos**, la cual tiene un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**

Más información . . .

. . . sobre la **actualización 2023** . . . https://www.fi.uba.ar/grado/carreras/ingenieria-electronica/plan-de-estudios

. . . sobre el **Taller de Sistemas Embebidos** . . . https://campusgrado.fi.uba.ar/course/view.php?id=1217



¡Hola!

Soy Juan Manuel Cruz
Taller de Sistemas Embebidos
Consultas a: jcruz@fi.uba.ar

1

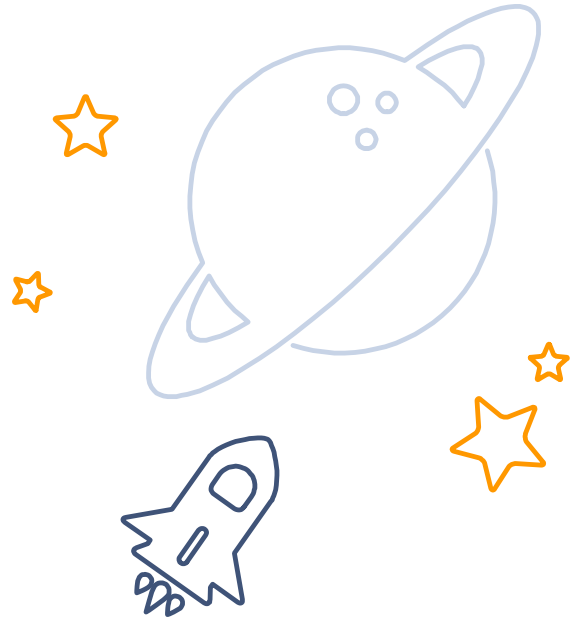
Introducción

Actualización 2023 del Plan de Estudios 2020 y resoluciones . . .

Conceptos básicos

- Referencia:
  - Programming whith STM32, Getting Started with the Nucleo Board and C/C++ - Donal Norris (Author)
  - Chapter 6: Interrupts
    - Este capítulo cubre las interrupciones. Como su nombre lo indica, se ocupan de detener la ejecución normal del programa al reconocer un evento predeterminado y luego atender ese evento.

Solución Adecuada



. . . lo más simple posible, previa determinación del objetivo de excelencia a cumplir, obviamente contando con la documentación debida y recurriendo a la metodología de trabajo adecuada

2

Documentación debida

1er Cuatrimestre de 2024, dictado por primera vez . . .

“

Programming whith STM32, Getting Started with the Nucleo Board and C/C++ - Donal Norris (Author)

Chapter 6: Interrupts

Interrupts

DEF

- Una interrupción, desde una perspectiva de alto nivel, es cualquier proceso que detiene momentáneamente la ejecución de un programa en curso, y requiere que la MCU ejecute código especializado relacionado con el evento que inició la interrupción.
  - Este código especializado se conoce como controlador de interrupciones.
  - El evento que causa la interrupción se denomina evento asincrónico, lo que significa que no tiene relación causal con el código del programa que se ejecuta normalmente.
  - En breve verá que los periféricos del temporizador suelen ser la fuente de estos eventos asincrónicos.
- Los canales de comunicación de datos, así como los procesos de acceso directo a la memoria (DMA), también suelen utilizar interrupciones.

Interrupts

- Una interrupción puede originarse en el hardware, como es el caso cuando se trata de un temporizador.
- También puede ser puramente impulsado por software, lo que suele ocurrir cuando se detecta una terminación anormal del programa, como un intento de dividir por cero o un intento de acceder a una ubicación de memoria inexistente.
- Las interrupciones de software en la mayoría de los lenguajes informáticos normalmente se denominan excepciones.
- Sin embargo, en la terminología ARM, todas las interrupciones, ya sean iniciadas por hardware o software, se denominan excepciones.

ESTO ESTA TEXTUAL EN EL PARCIAL DE GABI

Interrupts

- El periférico integrado (IP) ARM Cortex-M contiene un dispositivo de hardware llamado NVIC controlador de interrupción vectorial anidado (nested vectored interrupt controller) cuyo propósito es gestionar excepciones.
- La Figura 6-1 es un diagrama de bloques que ilustra cómo se conecta la NVIC con otros componentes clave de Cortex-M, incluido el núcleo de Cortex-M y otros periféricos.
- Puede ver dos tipos de conexiones por cable en el lado izquierdo de la Figura 6-1.

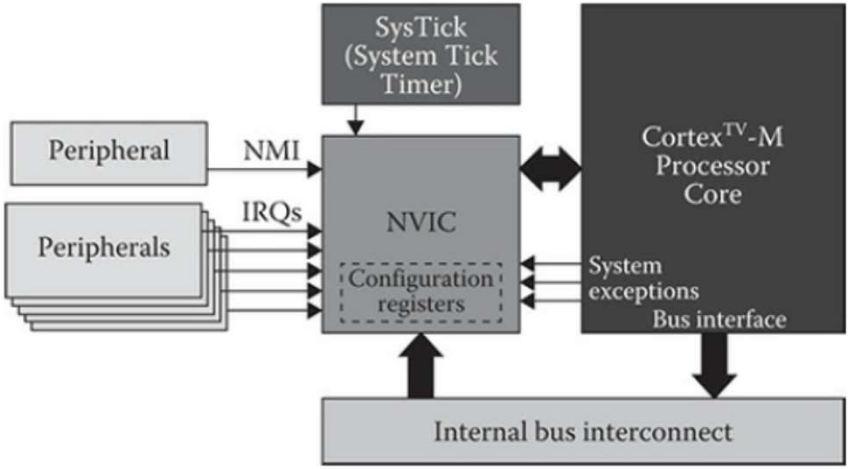


Figure 6-1 *NVIC block diagram.*

Interrupts

- Una sola línea etiquetada como NMI significa interrupción no enmascarable y se utiliza para permitir que un periférico indique al NVIC que inicie incondicionalmente el proceso de interrupción.
- El otro tipo de interrupción es IRQ, que significa solicitud de interrupción (interrupt request).
- Este tipo de interrupción se puede enmascarar, lo que significa que el NVIC no tiene que responder inmediatamente a una IRQ si se ha establecido un bit de registro de IRQ de enmascaramiento correspondiente.
- Por lo general, solo hay una NMI (non-maskable interrupt), pero hay muchas líneas IRQ que dependen de la capacidad del modelo NVIC específico.

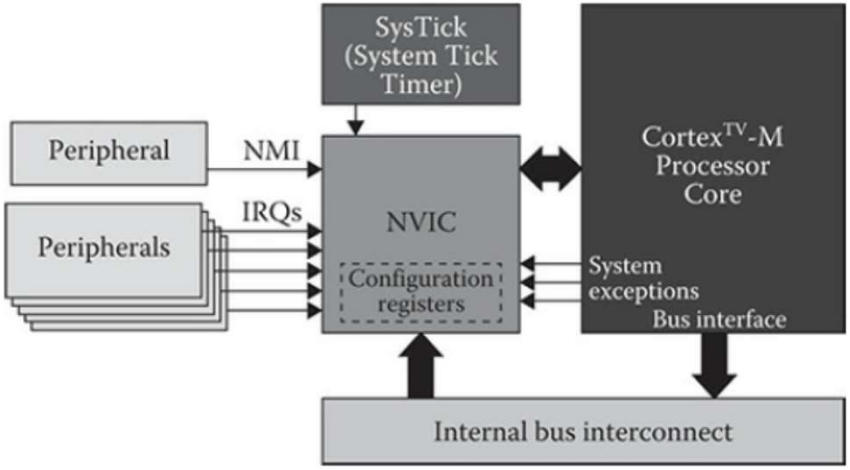


Figure 6-1 *NVIC block diagram.*

Interrupts

- El NVIC también está conectado entre el núcleo Cortex-M con líneas de E/S dedicadas y la estructura de interconexión de bus interno estándar.
- Esto significa que el NVIC es completamente compatible con CMCIS y utiliza el mismo tipo de registros de control y configuración que se detallaron en la discusión sobre GPIO.
- El NVIC está completamente integrado en el marco del software HAL y utiliza sus propias estructuras C para configurar una variedad de registros.
- El software del controlador para el NVIC se discutirá en un programa de demostración. En este punto, discutiré algunos detalles importantes sobre el NVIC.

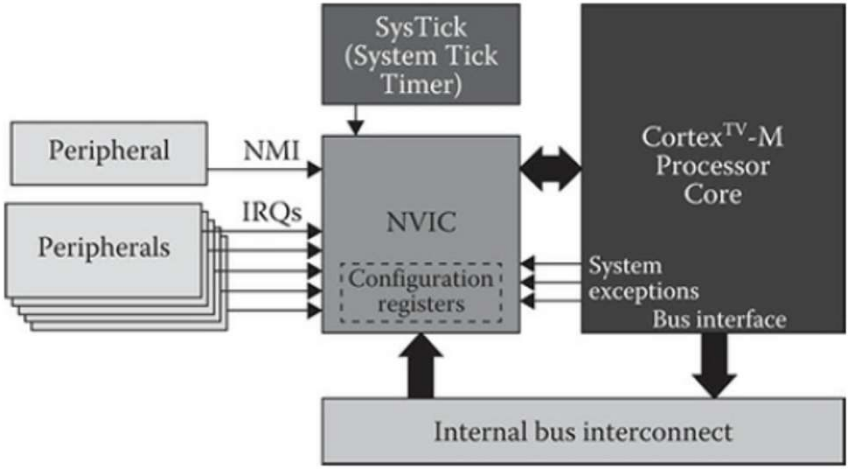


Figure 6-1 *NVIC block diagram.*

TEXTUAL EN EL PARCIAL

NVIC Specifications

- La especificación general STM NVIC establece que un solo NVIC puede gestionar hasta 256 canales de interrupción.
  - De los 256 canales, 240 pueden ser externos, mientras que 16 son siempre internos.
  - En realidad, la parte NVIC de la MCU STMF302R8, que controla la placa de desarrollo del proyecto, solo admite 52 canales.
  - Además, hay 16 líneas de interrupción de hardware que tienen conexiones directas al NVIC.
  - Cualquiera de las 51 líneas GPIO de la MCU se puede configurar como una interrupción enmascarable, lo que significa que pueden convertirse en un canal de interrupción.
  - Debería pensar en un canal de interrupción más como una construcción de software que como una implementación de hardware.

NVIC Specifications

- De hecho, la estructura de interrupción C utiliza la palabra "canal" (channel) para reforzar este concepto.

- A las interrupciones también se les puede asignar un nivel de prioridad.
  - Esto significa que una interrupción de mayor prioridad puede interrumpir una interrupción de menor prioridad.
  - Esta acción da significado a la palabra “anidado” en el acrónimo NVIC.
  - Un registro de 8 bits tiene niveles de prioridad de interrupción, lo que significa que puede haber un máximo de 256 niveles de prioridad.
  - No puedo concebir un sistema integrado práctico que alguna vez necesite tantos niveles de prioridad de interrupción.
  - Probablemente sería imposible diseñar un sistema así, y mucho menos implementarlo.

PRIORIDAD

Interrupt Process

- Es hora de volver a una discusión detallada del proceso de interrupción, como ya presenté el NVIC.
- Hay algunas definiciones que deben establecerse antes de continuar con esta sección:
  - Armar: permite que un dispositivo de hardware active una interrupción.
  - Desarmar: No permite que un dispositivo active una interrupción.
  - Habilitar: Permite interrumpir el proceso.
  - Desabilitar: No permite el proceso de interrupción.
- La habilitación y deshabilitación de interrupciones se logran fácilmente en el lenguaje C utilizando los métodos EnableInterrupts() y DisableInterrupts(), respectivamente.

TEXTUAL

Interrupt Process

- Una fuente potencial de interrupción puede desencadenar una interrupción siempre que las interrupciones estén habilitadas y se haya establecido un bit de activación específico asociado con esa fuente potencial en el registro de control NVIC apropiado.
- Esto se considera armar un dispositivo específico. Por el contrario, restablecer ese mismo bit de activación del registro se considerará desarmar el dispositivo. Hay una desviación importante de este procedimiento al considerar la interrupción SysTick. No tiene bit de habilitación debido a su importancia de que el MCU siempre reconozca sus eventos.
- Las prioridades de interrupción se establecen configurando los bits apropiados en el registro NVIC BASEPRI. Por ejemplo, si una interrupción específica se establece en un nivel de prioridad igual a 3, entonces cualquier interrupción con un nivel de prioridad de 0, 1 o 2 interrumpirá la interrupción de nivel 3. Sin embargo, si el registro BASEPRI se establece en 0, entonces la función de prioridad de interrupción se desactiva y todas las interrupciones se procesan a medida que las recibe el NVIC.

Interrupt Process

- Hay cinco condiciones previas que deben cumplirse antes de que se pueda generar una interrupción:
  - El dispositivo de interrupción potencial debe estar armado, es decir, el bit de activación debe estar activado.
  - NVIC debe estar habilitado.
  - Las interrupciones globales deben estar habilitadas.
  - La prioridad de interrupción debe ser mayor que el nivel actual.
  - Debe ocurrir un evento de activación de hardware real.
- Estas cinco condiciones pueden ocurrir en cualquier orden.
- Sin embargo, todos deben ser verdaderos simultáneamente para que se reconozca y procese una interrupción.

1) ARMAR
2) HABILITAR
3) HABILITAR
4) PRIORIDAD
5) EVENTO

Interrupt Process

- Si se reconoce una interrupción, ocurrirán los siguientes cinco eventos:
  - 1) completar — Se completará la instrucción que se está ejecutando actualmente.
  - 2) SUSPENDER — Se suspenderá el hilo actualmente en ejecución.
  - 3) Guardar 12 + 18 reg. — El contenido de 12 registros se almacenará en la pila. Se almacenarán 18 palabras adicionales si la unidad de punto flotante (FPU) estaba activa cuando se reconoció la interrupción.
  - 4) ¿LR? — El registro LR se establece en un valor específico. (LR es un registro de pila especial).
  - El registro de la PC se carga con la dirección de la rutina de servicio de interrupción (ISR). → INTERRUPT SERVICE ROUTINE
- La Figura 6-2 es una representación gráfica de los cinco eventos enumerados anteriormente.

Interrupt Process

MCU = µC

- El punto clave a destacar es que la NVIC, al reconocer una solicitud de interrupción válida, interactuará automáticamente con la MCU utilizando el firmware almacenado dentro de la NVIC.
- No se necesita ningún código especial, lo que hace que el procesamiento de la interrupción sea lo más rápido posible.
- Sólo se necesitan 12 ciclos de reloj, como se muestra en la Figura 6-2, para pasar de reconocer una interrupción a ejecutar la primera instrucción en el ISR.
- Esto significa que hay un tiempo de latencia de aproximadamente 1,5 µs para una MCU con frecuencia de 80 MHz.

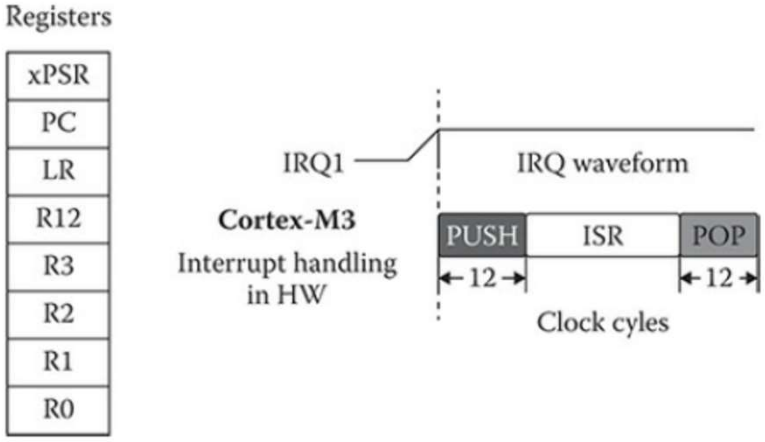


Figure 6-2 *Interrupt processing sequence.*

Interrupt Process

- Esta serie de eventos se denomina cambio de contexto y el hardware NVIC la invoca automáticamente.
  - El resultado de un cambio de contexto es que el subproceso en ejecución actualmente se convierte en un subproceso en segundo plano y el ISR se convierte en un subproceso en primer plano.
  - El cambio de contexto también funciona bastante bien con interrupciones priorizadas donde el ISR de menor prioridad se convierte en un subproceso en segundo plano y el ISR de mayor prioridad se convierte en un subproceso de primer plano.
- También existe el concepto de solicitud de interrupción pendiente, que usted debe conocer.
  - Una interrupción solicitante se mantiene en un estado pendiente si se detecta si las interrupciones globales aún no están habilitadas o si la interrupción solicitante tiene una prioridad menor que una interrupción actualmente activa.

Interrupt Process

- Una vez que se cumplan las cinco condiciones previas, el estado de la interrupción solicitante cambiará de pendiente a activo.

- También es muy importante borrar todos los bits de activación del dispositivo o, de lo contrario, se repetirán solicitudes de interrupción continuas.
  - Se llama reconocimiento para borrar un bit de activación y se realiza mediante una instrucción de software.
  - El único dispositivo que tiene un reconocimiento automático es la interrupción SysTick debido a su papel crítico en el funcionamiento de la MCU y su naturaleza recurrente de operación.
  - La instrucción de reconocimiento casi siempre se incluye en el ISR.
- Serían útiles algunas palabras sobre la naturaleza de un ISR para comprender qué papel desempeña en el proceso de interrupción.

Interrupt Process

- El ISR contiene todo el código necesario para procesar y completar cualquier acción que provocó la interrupción inicial.
- Un ISR suele ser un módulo de software muy compacto diseñado para una sola tarea que respalda el dispositivo de interrupción.
- Sin embargo, hay una implementación en la que solo un ISR grande da servicio a muchas fuentes de interrupción.
- En este caso, se emplea una técnica de encuesta (polling) para determinar la fuente exacta de la interrupción y utilizar solo esa parte del ISR grande para dar servicio al dispositivo de interrupción.
- Este no es el caso de las MCU STM. La implementación de STM utiliza ISR pequeños e independientes para dar servicio a cada dispositivo de interrupción que lo solicita.

Interrupt Process

- Hay una lista de direcciones para estos pequeños ISR almacenadas en una tabla de vectores.
  - Esta característica da lugar a la palabra "vector" en el nombre de NVIC.
- Quiero mencionar un área donde el proceso de interrupción vectorial STM realmente utiliza encuesta.
- Este sería el caso en el que varios pines GPIO en el mismo puerto podrían emitir por separado su propia interrupción.
  - Dado que el puerto GPIO solo tiene una dirección vectorial, le corresponde al software encuestar todos los bits de activación del puerto para determinar qué bit está armado (solicitó la interrupción).

Interrupt Process

- Cada ISR está escrito para dar servicio al dispositivo de interrupción de la manera más eficiente posible y emitirá el reconocimiento e invocará el cambio de contexto.
  - Esto último se logra fácilmente mediante un comando en lenguaje C que tiene un formato de lenguaje ensamblador similar a BX LR, que muy rápidamente devolverá todos los valores de registro almacenados a sus respectivos registros y así restaurará la MCU a su estado previo a la interrupción.
  - El objetivo del diseño de ingeniería de software es que los ISR se ejecuten lo más rápido posible y regresen al hilo de primer plano anterior.
    - Esto significa que nunca se deben colocar bucles de espera en un ISR.
- Cualquier dato que se genere o modifique en un ISR debe almacenarse en ubicaciones de memoria compartida globalmente, ya que de lo contrario no estará disponible para ningún otro módulo de software debido a la naturaleza de cómo el lenguaje C implementa el ISR framework.

- Las interrupciones externas, como las que se obtienen a través de pines GPIO, se conectan al NVIC mediante un controlador de interrupciones externo (EXTI).
  - El uso de un EXTI permite la multiplexación de muchos pines GPIO en las entradas de interrupción externas del NVIC. La Figura 6-3 es un diagrama de bloques del EXTI.
- El controlador EXTI en la MCU STM32F302R8 realiza las siguientes actividades:
  - Maneja hasta 23 eventos de software separados y/o activadores de hardware conectados a 16 líneas externas de interrupción/evento

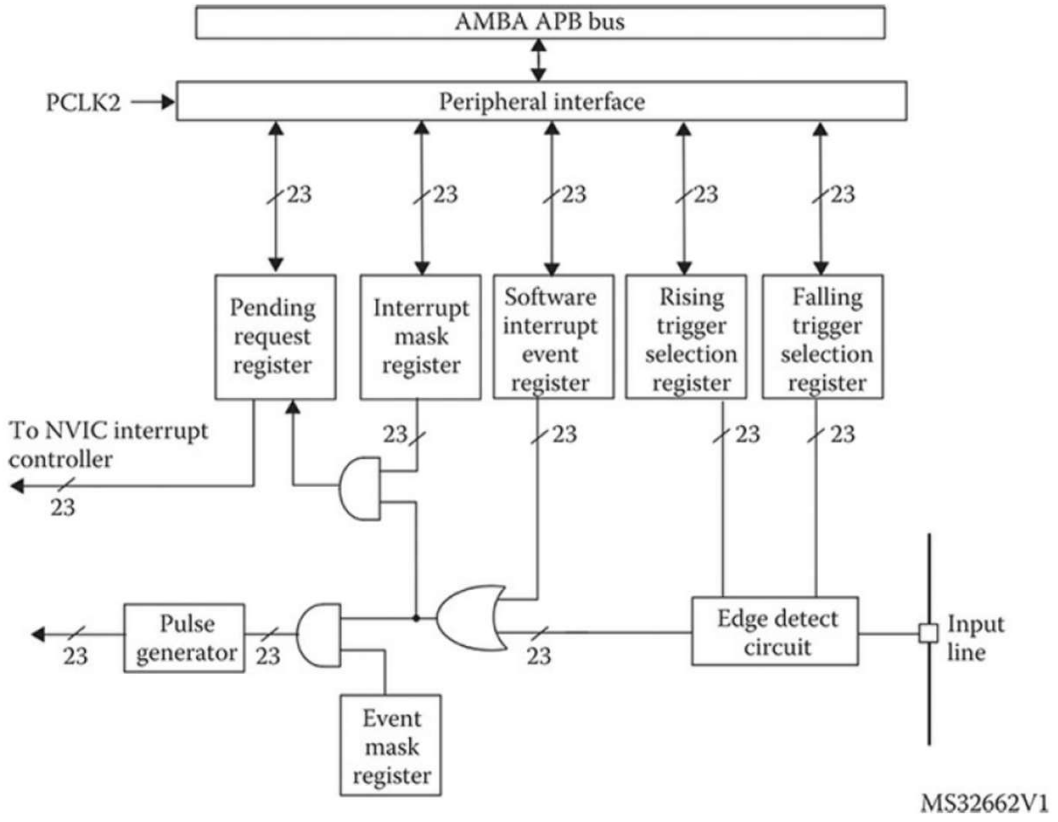


Figure 6-3 *EXTI block diagram.*

- Proporciona un bit de estado para cada línea
- Proporciona activadores y máscaras independientes para cada línea
- Puede detectar pulsos externos con un período de reloj más corto que el reloj del bus APB2, que impulsa el EXTI

Cualquiera de los pines GPIO STM32F302R8 51 se puede utilizar como fuente de interrupción. Por supuesto, sólo hasta 23 podrían estar activos en un momento dado. Esas interrupciones externas deberían poder manejar fácilmente cualquier diseño embebido imaginable.

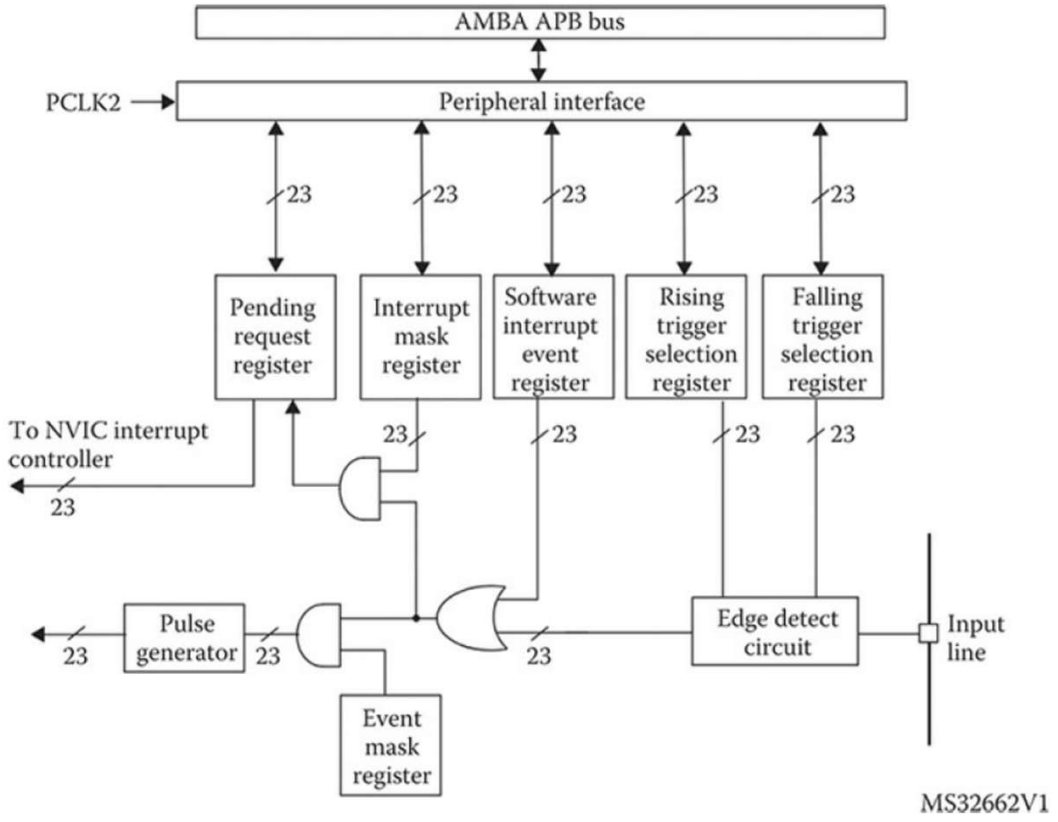


Figure 6-3 *EXTI block diagram.*

- También existen otras cinco líneas EXTI que están conectadas de la siguiente manera:
  - Línea EXTI 16 conectada a la salida PVD
  - Línea EXTI 17 conectada al evento de alarma RTC
  - Línea EXTI 18 conectada al evento de activación USB OTG FS
  - Línea EXTI 21 conectada a los eventos RTC Tamper y TimeStamp
  - Línea EXTI 22 conectada al RTC Wakeup

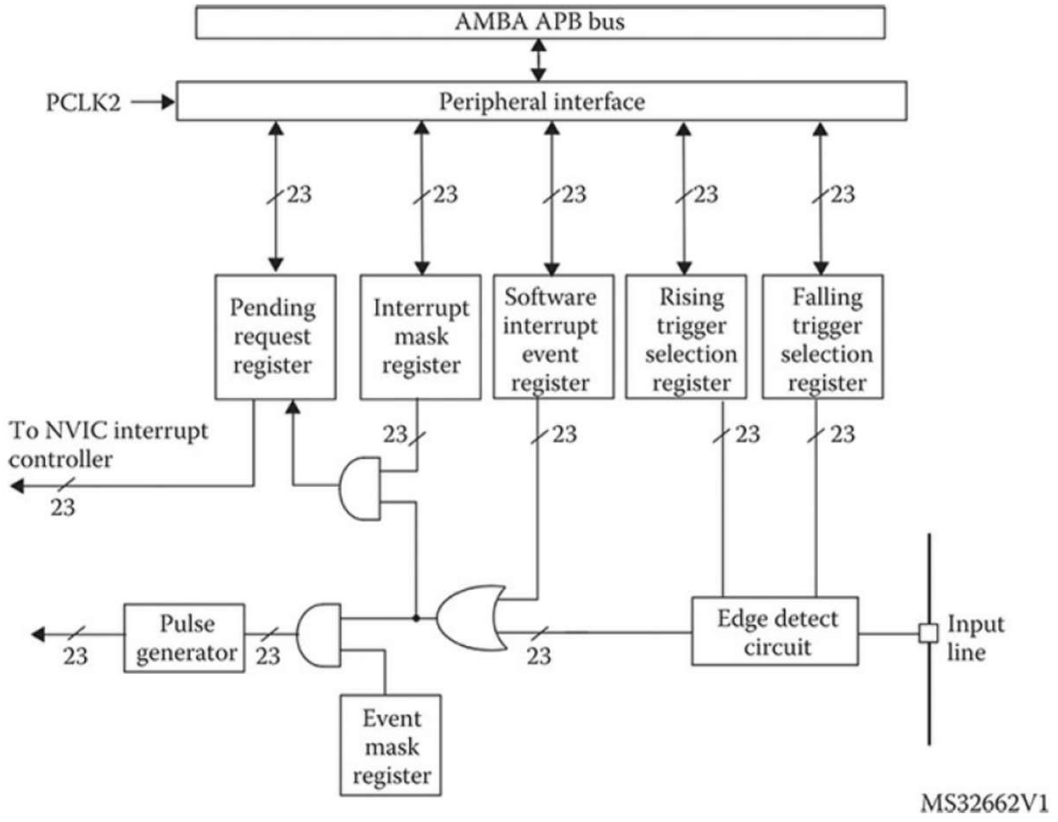


Figure 6-3 *EXTI block diagram.*

- La Figura 6-4 muestra la relación entre los pines GPIO, las entradas del controlador de interrupciones externo y las IRQ de NVIC.
  - Debe tener en cuenta que el pin GPIO PC13 está agrupado junto con muchos otros pines en una IRQ NVIC común llamada EXTI15_10_IRQ. Esto es bastante importante en la demostración del programa de interrupción.
  - En este punto, debería haber adquirido conocimientos suficientes para apreciar una demostración de interrupción sencilla que se analiza en la siguiente sección.

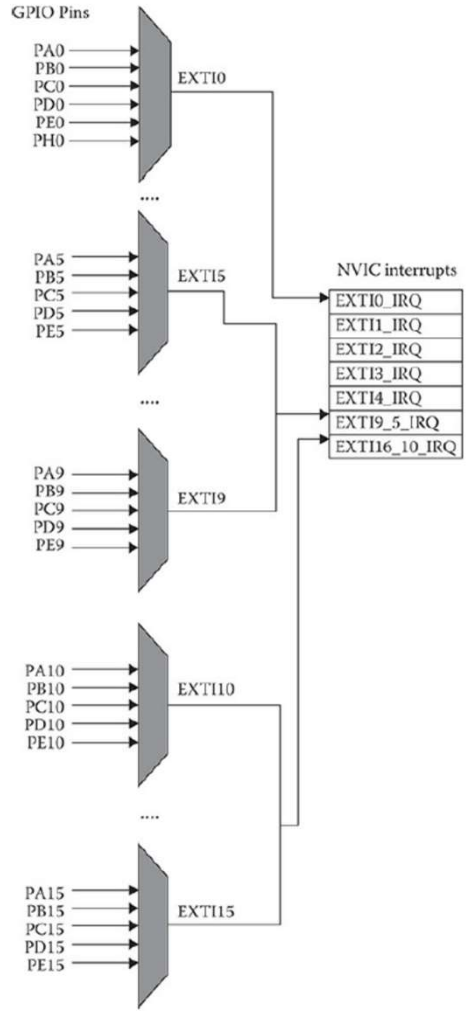


Figure 6-4 *GPIO pins, external interrupt controller, and NVIC IRQs.*

Interrupt Demonstration

- Esta demostración de interrupción se creará de manera que un usuario pueda presionar el botón azul en la placa Nucleo-64 y hacer que el LED del usuario parpadee en un intervalo de alternancia diferente, es decir, presionarlo una vez para que parpadee dos veces por segundo y presionarlo nuevamente. para que parpadee una vez cada dos segundos.
- El programa aprovecha al máximo el marco HAL para invocar la interrupción GPIO adecuada para el pin GPIO PC13 conectado al botón azul del usuario. El ISR contiene un código que alterna LD2, un LED verde, que está conectado al pin GPIO PA5.
- La siguiente es una lista de código completa para el archivo main.c contenido en el proyecto. A continuación del listado se incluye una discusión de las partes clave del código.

```
// STM disclaimer goes here
// Includes
#include "main.h"
#include "stm32f3xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

// Private variables

/* USER CODE BEGIN PV */
// Private variables
uint16_t tDelay = 250;  // also used as an external variable
/* USER CODE END PV */

// Private function prototypes
void SystemClock_Config(void);
static void MX_GPIO_Init(void);

/* USER CODE BEGIN PFP */
// Private function prototypes

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  // MCU Configuration

  // Reset of all peripherals, Initialize the flash and the Systick.
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  HAL_Delay(tDelay);

  }
  /* USER CODE END 3 */

}

// System Clock Configuration
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

  // Initialize the CPU, AHB and APB buss clocks
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
```

```
  // Initializes the CPU, AHB and APB buss clocks
  RCC_ClkInitStruct.ClockType =
  RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
  |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  // Configure the Systick interrupt time
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  // Configure the Systick
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
     PA2   ------> USART2_TX
     PA3   ------> USART2_RX
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : USART_TX_Pin USART_RX_Pin */
  GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error re-
turn state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line
number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
```

Interrupt Demonstration

```
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

- Examinar el método MX_GPIO_Init revela que no es muy diferente al código del mismo método utilizado en el capítulo anterior. Los dos cambios en el método son los siguientes:
  - El pin GPIO PC13 se ha configurado para generar una interrupción cuando se presiona con la siguiente declaración:

```
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
```

  - A continuación, la declaración asignó una prioridad a la interrupción externa:

```
HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
```

Interrupt Demonstration

```
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

- Luego se permitió que el NVIC reconociera una interrupción de la PC13 mediante la siguiente declaración:

```
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

- Esos son todos los cambios necesarios que se deben aplicar al método MX_GPIO_Init.
- Lo único que queda es agregar el código del controlador ISR al proyecto.
  - Este ISR debe colocarse en el archivo stm32f3xx_it.c. El prototipo de ISR es void EXTI15_10_IRQHandler (void).
- La lista completa del archivo stm32f3xx_it.c se muestra a continuación:

EXTI15_10_IRQHandler(void). The complete stm32f3xx_it.c file listing is shown below:

```
// STM disclaimer goes here
// Includes
#include "stm32f3xx_hal.h"
#include "stm32f3xx.h"
#include "stm32f3xx_it.h"

/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

// External variables
extern uint16_t tDelay;

/**
* Cortex-M4 Processor Interruption and Exception Handlers
*/

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/**
* STM32F3xx Peripheral Interrupt Handlers
* Add here the Interrupt Handlers for the used peripherals.
*
* For the available peripheral interrupt handler names,
* please refer to the startup file (startup_stm32f3xx.s).
*/

/**
* @brief This function handles RCC global interrupt.
*/
void RCC_IRQHandler(void)
{
  /* USER CODE BEGIN RCC_IRQn 0 */

  /* USER CODE END RCC_IRQn 0 */
  /* USER CODE BEGIN RCC_IRQn 1 */
  /* USER CODE END RCC_IRQn 1 */
}

/**
* @brief This function handles EXTI line[15:10] interrupts.
*/
void EXTI15_10_IRQHandler(void)
{
/* USER CODE BEGIN EXTI15_10_IRQn 0 */
    for(int i = 0; i < 65535; i++); // short delay for debounce
          if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13)) // no defines used
          {
             if(tDelay == 250)
                    tDelay = 1000;
                 else
                    tDelay = 250;
          }
/* USER CODE END EXTI15_10_IRQn 0 */

  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
/**** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Interrupt Demonstration

- Debería ver inmediatamente que el ISR tiene sólo siete declaraciones, lo que concuerda muy bien con mi discusión anterior de que los ISR deben ser lo más concisos posible y estar dedicados a una sola tarea.
  - La primera declaración es un bucle “for” sin operación (nop) que introduce un retraso muy breve para evitar el rebote del botón pulsador.
  - La siguiente declaración es un “if" compuesto que lee el nivel de tensión conectado al botón pulsador.
    - Será alto porque el programa ya está en el controlador de interrupciones.
    - Si la variable externa C tDelay se establece en 250, se restablecerá a 1000 y viceversa.
    - La última declaración en el ISR activa el controlador NVIC IRQ para el controlador externo conectado al pin PC13.

Interrupt Demonstration

Test Run

- Construí el proyecto y lo descargué en la placa Nucleo-64 usando el procedimiento detallado varias veces en capítulos anteriores.
- Inicialmente observé que el LED parpadeaba dos veces por segundo después de la descarga inicial.
- Luego presioné el botón azul y observé que el LED comenzó a parpadear a un ritmo de una vez cada dos segundos.
- Al presionarlo nuevamente se volvió a la velocidad de parpadeo de dos veces por segundo.
- Esta fue una prueba convincente de que la demostración de interrupción estaba funcionando como se esperaba.

Interrupt Demonstration

- Es cierto que esta es una demostración muy simple, pero contiene todos los elementos necesarios para soportar programas mucho más complejos controlados por interrupciones.
- Descubre que las interrupciones son el mejor amigo de un desarrollador integrado.
- La mayoría de mis proyectos de desarrollo integrados utilizan interrupciones.
- No usarlos le priva de una herramienta importante para crear programas integrados eficientes.
- No es exagerado decir que los proyectos integrados modernos no serían prácticos sin emplear al menos una interrupción.

Summary

- Comencé el capítulo discutiendo cómo funciona el proceso de interrupción de STM MCU. Esta discusión incluyó la descripción y el funcionamiento del NVIC y un recorrido por una secuencia completa de procesamiento de interrupciones.
- A continuación siguió una demostración de interrupción en la que un usuario presionó el botón azul ubicado en la placa del proyecto Nucleo-64, lo que provocó que se generara una interrupción. En consecuencia, el código de rutina de servicio de interrupción (ISR) correspondiente alteró el estado de un LED integrado.

Referencias

- Programming with STM32: Getting Started with the Nucleo Board and C/C++ 1st Edición - Donal Norris (Author)
- Nucleo Boards Programming with the STM32CubeIDE, Hands-on in more than 50 projects - Dogan Ibrahim (Author)
- STM32 Arm Programming for Embedded Systems, Using C Language with STM32 Nucleo - Muhammad Ali Mazidi (Author), Shujen Chen (Author), Eshragh Ghaemi (Author)

Manos a la obra con el . . .

. . . Proyecto Intermedio

. . . un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**


Las y los estudiantes preguntarán:
¿en qué lío nos metimos?

¡Muchas gracias!

¿Preguntas?

. . .

Consultas a: jcruz@fi.uba.ar