2. 3. Interrupciones.

Usualmente los periféricos de un microcontrolador tienen unas “banderas” indicadoras del estado de operación. Las banderas son bits específicos de un módulo periférico que se ponen en uno para indicar un evento o estado. Estas banderas pueden, generalmente, ser leídas por el procesador en cualquier momento para tomar una acción. Muchas veces la lectura de las banderas son leídas durante la ejecución de un proceso y los eventos no se leen inmediatamente cuando estos ocurren. Algunos periféricos activan las banderas mientras el evento ocurre, pero luego las banderas cambian de estado tan pronto el evento desaparece, haciendo que el procesador no se de cuenta del evento.

Las interrupciones son una forma de indicarle al procesador que un evento ha ocurrido por parte de un periférico de forma inmediata en el momento que ocurre el evento, de tal manera que el procesador puede ir inmediatamente a tratar el evento.

El STM32F103C8T6 tiene un módulo controlador de interrupciones llamado Nvic (Nested Vector Interrupt Controller). Este módulo se encarga de manejar los indicadores de eventos y de interrupciones de todos los periféricos, inclusive el microcontrolador es capaz de ir a un estado de sueño para bajo consumo y ser despertado por un evento previamente programado. También el módulo controlador es capaz de generar una interrupción o un evento por software, configurando los registros que el módulo tiene para esto.

La activación de la generación de interrupción se resume en fijar las características de la interrupción definiendo una estructura que lleve esta información del tipo `NVIC_InitTypeDef`, luego inicializando sus valores, y luego llamando la función de inicialización de interrupción `NVIC_Init(&NVIC_InitStruct)`. Los valores de inicialización son:

- `uint8_t NVIC_IRQChannel`: El canal IRQ que se va a habilitar. Puede ser cualquiera de la lista que sigue.
- `uint8_t NVIC_IRQChannelPreemptionPriority`: Define la prioridad de canal IRQ. Este es un valor entre 0 y 15.
- `uint8_t NVIC_IRQChannelSubPriority`: Nivel de sub prioridad del canal IRQ.
- `FunctionalState NVIC_IRQChannelCmd`: Puede ser `ENABLE` or `DISABLE`, para habilitar o deshabilitar la interrupción, respectivamente.

La lista de los canales que se pueden programar para una interrupción pueden ser:

- `ADC1_2_IRQn`
- `USB_HP_CAN1_TX_IRQn`
- `USB_LP_CAN1_RX0_IRQn`
- `CAN1_RX1_IRQn`
- `CAN1_SCE_IRQn`
- `EXTI9_5_IRQn`
- `TIM1_BRK_IRQn`
- `TIM1_UP_IRQn`
- `TIM1_TRG_COM_IRQn`

- TIM1_CC_IRQn
- TIM2_IRQn
- TIM3_IRQn
- TIM4_IRQn
- I2C1_EV_IRQn
- I2C1_ER_IRQn
- I2C2_EV_IRQn
- I2C2_ER_IRQn
- SPI1_IRQn
- SPI2_IRQn
- USART1_IRQn
- USART2_IRQn
- USART3_IRQn
- EXTI15_10_IRQn
- RTCAlarm_IRQn
- USBWakeUp_IRQn

De esta lista de canales los nombres son autoexplicativos. Los canales EXTI9_5_IRQn y EXTI15_10_IRQn son interrupciones externas de los pines 5 al 9 y de los pines 10 al 15, respectivamente. Los demás pines pueden programar su interrupción independientemente, EXTI0, EXTI1, EXTI2, EXTI3, y EXTI4. Cada evento puede ser programado en modo de flanco de subida, de bajada o de ambos. EXTI0 es el evento del bit 0 de cualquier puerto, ya sea A, B, ..., o D. Igual ocurre para EXTI1 para el bit 1, etc.

Una vez iniciada la configuración de interrupción se debe habilitar la interrupción en el periférico y luego hacer una función de tratamiento a la interrupción. Las funciones deben tener un nombre específico de acuerdo a la interrupción y al periférico. La lista siguiente muestra los nombres de las funciones de interrupción de periféricos.

- WWDG_IRQHandler // Window Watchdog
- PVD_IRQHandler // PVD through EXTI Line detect
- TAMPER_IRQHandler // Tamper
- RTC_IRQHandler // RTC
- FLASH_IRQHandler // Flash
- RCC_IRQHandler // RCC
- EXTI0_IRQHandler // EXTI Line 0
- EXTI1_IRQHandler // EXTI Line 1
- EXTI2_IRQHandler // EXTI Line 2
- EXTI3_IRQHandler // EXTI Line 3
- EXTI4_IRQHandler // EXTI Line 4
- DMA1_Channel1_IRQHandler // DMA1 Channel 1
- DMA1_Channel2_IRQHandler // DMA1 Channel 2
- DMA1_Channel3_IRQHandler // DMA1 Channel 3
- DMA1_Channel4_IRQHandler // DMA1 Channel 4
- DMA1_Channel5_IRQHandler // DMA1 Channel 5
- DMA1_Channel6_IRQHandler // DMA1 Channel 6
- DMA1_Channel7_IRQHandler // DMA1 Channel 7
- ADC1_2_IRQHandler // ADC1_2
- USB_HP_CAN1_TX_IRQHandler // USB High Priority or CAN1 TX

- `USB_LP_CAN1_RX0_IRQHandler // USB Low  Priority or CAN1 RX0`
- `CAN1_RX1_IRQHandler        // CAN1 RX1`
- `CAN1_SCE_IRQHandler        // CAN1 SCE`
- `EXTI9_5_IRQHandler         // EXTI Line 9..5`
- `TIM1_BRK_IRQHandler        // TIM1 Break`
- `TIM1_UP_IRQHandler         // TIM1 Update`
- `TIM1_TRG_COM_IRQHandler    // TIM1 Trigger and Commutation`
- `TIM1_CC_IRQHandler         // TIM1 Capture Compare`
- `TIM2_IRQHandler            // TIM2`
- `TIM3_IRQHandler            // TIM3`
- `TIM4_IRQHandler            // TIM4`
- `I2C1_EV_IRQHandler         // I2C1 Event`
- `I2C1_ER_IRQHandler         // I2C1 Error`
- `I2C2_EV_IRQHandler         // I2C2 Event`
- `I2C2_ER_IRQHandler         // I2C2 Error`
- `SPI1_IRQHandler            // SPI1`
- `SPI2_IRQHandler            // SPI2`
- `USART1_IRQHandler          // USART1`
- `USART2_IRQHandler          // USART2`
- `USART3_IRQHandler          // USART3`
- `EXTI15_10_IRQHandler       // EXTI Line 15..10`
- `RTCAlarm_IRQHandler        // RTC Alarm through EXTI Line`
- `USBWakeUp_IRQHandler       // USB Wakeup from suspend`

Por dentro de la función que trata la interrupción se debe borrar la bandera activada para que no vuelva a entrar a ejecutar la función sin haber un nuevo evento, además se deben escribir allí las instrucciones que tengan lugar a hacer un proceso debido al evento que se genera. Hay que evitar escribir en estas funciones llamados a otras funciones o hacer bucles infinitos, ya que el procesador va a tratar un evento y es posible que lleguen varios más.

El STM32F103C8T6 tiene dos tipos de prioridad: Prioridad preferente y sub prioridad (preemption and sub priority). Cuando dos interrupciones ocurren al mismo tiempo, la que tiene mayor prioridad preferente se ejecuta primero. Si las dos tienen igual prioridad preferente, la que tiene mayor sub prioridad se ejecuta primero. Si las dos tienen igual prioridad preferente e igual sub prioridad, la que llegó primero es la que se ejecuta.

Este microcontrolador usa cuatro bits para almacenar las prioridades de preferente y sub. Si se usan los cuatro bits para almacenamiento de prioridad preferente, no habrá espacio para la sub prioridad. Igualmente, si se usan los cuatro bits para almacenamiento de la sub prioridad, no habrá espacio para la prioridad preferente. Este proceso se llama agrupado de prioridad. Hay cinco diferentes grupos.

- Grupo 0: 0 bits para preferente y 4 bits para sub prioridad.
- Grupo 1: 1 bits para preferente y 3 bits para sub prioridad.
- Grupo 2: 2 bits para preferente y 2 bits para sub prioridad.
- Grupo 3: 3 bits para preferente y 1 bits para sub prioridad.
- Grupo 4: 4 bits para preferente y 0 bits para sub prioridad.

De esta forma se puede llamar la función NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup) para determinar el grupo de interrupción. El grupo por defecto es el 2. Una vez se escoge el grupo, se debe establecer las prioridades. Por ejemplo, para darle una prioridad 2 al periférico TIM1, se escribe el siguiente código.

```
NVIC_InitTypeDef NVIC_InitStruct;
NVIC_InitStruct.NVIC_IRQChannel = TIM1_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 2;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

A menor número de prioridad, mayor es la prioridad. Por ejemplo, prioridad 1 es mayor que prioridad 3.

El siguiente ejemplo muestra una interrupción desde el pin de entrada cuatro del puerto GPIOB.

```
// configuracion de interrupcion del pin GPIOB04:
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource4);
    EXTI_InitTypeDef EXTI_InitStruct;
    EXTI_InitStruct.EXTI_Line = EXTI_Line4;
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_Init(&EXTI_InitStruct);

    NVIC_InitTypeDef NVIC_Struct;
    NVIC_Struct.NVIC_IRQChannel = EXTI4_IRQn;
    NVIC_Struct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_Struct.NVIC_IRQChannelSubPriority = 0;
    NVIC_Struct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_Struct);

void EXTI4_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line4) != RESET) {
               //GPIO_ResetBits(GPIOB, GPIO_Pin_12);// enciende led
               EXTI_ClearITPendingBit(EXTI_Line4);
       }
}
```