Interrupciones y excepciones en RISC-V

Sistemas con Microprocesadores

Mg.Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

http://www.microprocesadores.unt.edu.ar/procesadores

Cronograma

Actividad	Inicio	Descripción	Completa
Presentación	24/08	Presentación y reglamento	✔
Tema 1	24/08	Estructura de las computadoras	✔
Tema 2	26/08	Proyecto con un microcontrolador	✔
Tema 3	31/08	Descripción funcional del ISA RV32IM	✔
Tema 4	07/09	Programación en lenguaje ensamblador	✔
Tema 5	18/09	Diseño del camino de datos de un procesador	✔
Tema 6	30/09	Diseño de la unidad de control del procesador	✔
Tema 7	05/10	Tecnologias y sistema de memoria	✔
Tema 8	09/10	Dispositivos de entrada/salida	✔
Tema 9	19/10	Entradas/salidas digitales (GPIO)	✔
Tema 10	28/10	Generalidades de las interrupciones	✔
Tema 11	04/11	Interrupciones y excepciones en RISC-V	←
Tema 12	13/10	Manejo de señales con temporizadores	
Tema 13	18/11	Transferencias de datos	
Proyectos	30/11	Seminarios de proyectos	

Arquitectura privilegiada en RISC-V

- Todos los aspectos básicos relacionados con excepciones, interrupciones, traps y privilegios se definen en el "The RISC-V Instruction Set Manual: Volume II - Privileged Architecture”
- Para soportar esta arquitectura es necesario que el núcleo implemente la extensión optativa Zicsr:
  - Define hasta 4096 registros de control y estado (CSRs)
  - Tambien define 6 instrucciones para leer y escribir estos registros
- En la arquitectura privilegiada se definen nuevos registros de control y estado (CSRs) y el comportamiento del procesador vinculado a los valores de estos registros

Arquitectura privilegiada en RISC-V

- Como en otras áreas, en estándar para la arquitectura privilegiada RISC-V es modular
  - Algunos aspectos siempre están implementados
  - Otros aspectos dependen de extensiones optativas
- Se necesitan tambien algunos detalles de la implementación del fabricante
  - En el curso utilizaremos como microcontrolador el GD32VF103 de la firma GigaDevices
  - Implementa núcleo RISC-V Bumblebee, una variante del N201 desarrollado por Nuclei System Technology

Entorno de ejecución

- Se define como entorno de ejecución al conjunto de componentes de hardware y software que permiten que un programa se ejecute
- Sin embargo, el sistema operativo es un programa y por lo tanto puede, o no, formar parte del entorno de ejecución:
  - En las computadoras de propósito general el sistema operativo se considera como una parte del entorno de ejecución
  - En los sistemas embebidos habitualmente el sistema operativo forma parte de la aplicación y no se considera como una parte del entorno de ejecución

Reset

- La arquitectura RISC-V no define un comportamiento para el reset del procesador
  - Cada plataforma define como asigna el valor inicial al procesador
- El procesador GD32VF103 asigna al PC el valor inicial **`0x0000.0000`** en forma fija
  - La secuencia de configuración inicial debe comenzar siempre en esa dirección
- Sin embargo si define valores iniciales para algunos registros de configuración y estado
  - Por ejemplo el procesado debe iniciar en modo máquina

Ejemplo de aplicación (1-reset)

- Se define una rutina en ensamblador que se ejecuta inmediatamente después del reset
  - Para esto la rutina debe definirse en la sección **.init** en lugar de la sección **`.text`**
- El enlazador utiliza un archivo de instrucciones (linker script) que define como se debe construir la imagen final de memoria
  - Generalmente este archivo es provisto por el fabricante del procesador
  - Este archivo define que la sección **`.init`** se debe ubicar al inicio de la memoria flash, en la dirección **`0x0000.0000`**

Ejemplo de aplicación (1-reset)

```
# Función para generar una demora constante
    .text
delay:
    la      t0, 500000
loop_delay:
    nop
    addi    t0, t0, -1
    bne     t0, zero, loop_delay
    ret

# Función principal que se ejecuta al completar el reset
    .section .init
    .global  _start
    .type    start, @function
_start:
    jal     habilitar_reloj
    li      a0, 1
    jal     configurar_led
main_loop:
    li      a0, 1
    jal     cambiar_led
    jal     delay
    j       main_loop
```

Niveles de privilegio en RISC-V

- Se llaman modos y pueden existir hasta cuatro niveles: Máquina, Hipervisor, Supervisor y Usuario
- Es obligatorio implementar el modo Máquina
  - Se puede acceder a toda la funcionalidad del procesador
  - El procesador inicia siempre en este modo
- Si implementa el modo Usuario entonces existen instrucciones para modificar los registros de estado
- Se define la instrucción **`ecall`** para pedirle servicios al supervisor del entorno de ejecución
- Se definen las instrucciones **`mret`** y **`sret`** para terminar un handler que ejecuta en modo maquina o supervisor

Delegación de privilegios en RISC-V

- Todas las excepciones son atendidas, por defecto, por el modo con el mayor nivel de privilegio: el modo Máquina
- Un modo con mayores niveles de privilegio puede delegar la atención de ciertas excepciones o interrupciones a un modo menos privilegiado
  - El modo Máquina puede delegar al Sistema Operativo
- Esto permite que un entorno de ejecución emule las capacidades de otro entorno de ejecución
  - Un procesador con acceso no alineado a memoria puede, por software, emular a un procesador con acceso no alineado por hardware en forma transparente para el Sistema operativo
- Este es el objetivo principal de modo Hipervisor

Excepciones en RISC-V: CLINT

- En la Arquitectura Privilegiada se define un controlador muy básico denominado Core Local Interruptor (CLINT) que provee:
  - Soporte para interrupciones locales y excepciones
  - Funcionalidades básicas de temporización
  - Anidamiento usando los niveles de privilegios
  - Conexiones fijas de interrupciones externas como fuentes locales de interrupciones rápidas
- Este sistema está pensado para ser muy simple y disminuir al máximo la latencia en los traps

Excepciones en RISC-V: CLINT

- En RISC-V las excepciones no pueden ser enmascaradas, siempre producen un trap y ejecutan el handler asociado
- Pueden ocurrir varias excepciones al mismo tiempo, la prioridad es fija y está definida en el estándar
- Se puede configurar un único handler para todas las excepciones o utilizar un esquema vectorizado
  - En nuestro procesador, para las excepciones, solo se puede utilizar el esquema de handler único.
  - El modo vectorizado solo está disponible para las interrupciones
- Se utiliza el registro **`mtvec`** para definir el modo y para la dirección de inicio de la rutina o del vector de interrupciones

Respuesta a las excepciones

- Almacenar el valor actual del **`pc`** en el registro de estado **`mepc`**
- Almacenar el identificador de la excepción en el registro de estado **`mcause`**
- Cambiar los niveles de privilegio y ajustar el registro de estado **`mstatus`**
- Cargar en el **`pc`** la dirección de la primera instrucción del handler a partir del registro de control **`mtvec`**

Estructura del handler

- Salvar el estado de los registros en la pila
- Analizar el registro **`mcause`** para determinar el origen de la excepción
- Si no se trata de un error fatal:
  - Ejecutar los pasos para subsanar el error
- Si se trata de un error fatal:
  - Es posible iniciar un programa alternativo, o reiniciar el mismo programa, cambiando el valor del registro **`mepc`**
- Recuperar el estado de los registros de la pila
- Terminar el trap con la instrucción **`mret`**

Ejemplo de aplicación (2-excepciones)

- En la configuración inicial se define la dirección de inicio de la rutina de servicio de las excepciones usando el registro **`mtvec`**
  - Este procesador exige una alineación de 64 bytes en la rutina de servicio definida en el registro **`mtvec`**
    - Los últimos 6 bits de la dirección deben ser siempre cero
- El lazo principal hace parpadear el led verde
- Al presionar la tecla cancelar se intenta ejecuta una instrucción inválida
- La rutina de servicio prende el led rojo como indicación de error y queda en un lazo infinito

Ejemplo de aplicación (2-excepciones)

```
# Rutina de servicio que se ejecuta en respuesta a una excepción
    .align    6
error_handler:
    li        a0, 1
    jal       prender_salida        # Prende el led rojo de la placa
halt:
    j         halt                  # Lazo infinito

# Función principal que se ejecuta al completar el reset
    .section .init
    .global  _start
_start:
    jal       configurar_placa      # Configura el hardware de la placa
    la        t0,error_handler
    csrw      CSR_MTVEC, t0         # Define la rutina para las excepciones
main_loop:
    li        a0, 3
    jal       cambiar_salida        # Cambia el estado del led verde
    jal       delay                 # Prende el led rojo de la placa
    li        a0, 10
    jal       leer_entrada          # Lee el estado de la tecla cancelar
    beqz      a0, main_loop         # Repite el lazo si la tecla no esta apretada
    .word     0xFFFFFFFF            # Ejecuta una instrucción inválida
```

Interrupciones en RISC-V: CLINT

- Se definen dos fuentes internas de interrupciones y se las diferencia de las fuentes externas al procesador
- El temporizador del sistema:
  - Está formado por un contador, un registro donde se almacena un valor y un comparador
  - Cuando el contador alcanza el valor almacenado en el registro, el comparador genera una interrupción
  - Se utiliza para asignar cuotas de tiempo un proceso
- La interrupción por software:
  - Consiste simplemente en un flip-flop que se gestiona por software
  - Cuanto el software lo pone en 1 se genera una interrupción
  - Se utiliza para la comunicación entre núcleos en un procesador

El temporizador del sistema

- Esta formado por
  - Un contador **`mtime`** que utiliza el reloj del procesador
  - Un registro **`mtimecmp`** que se escribe por software
  - Un comparador que genera una interrupción cuando el valor del contador **`mtime`** es mayor o igual que el valor del registro **`mtimecmp`**
- Esto permite programar una interrupción que ocurrirá en un futuro perfectamente definido
  - Habitualmente la utiliza el sistema operativo para asignar las cuotas de tiempo a cada proceso
  - Tambien se puede utilizar para generar una interrupción periódica
- En la rutina de servicio se debe actualizar el valor del registro **`mtimecmp`** para programar la próxima interrupción

Ejemplo de aplicación (3-temporizador)

- Se utiliza el temporizador del sistema para producir una interrupción periódica
- Para la primera interrupción se carga el comparador con la suma del periodo al valor actual de temporizador
- En cada interrupción se carga el comparador con la suma del periodo al valor anterior del mismo comparador
  - Esto evita un error acumulativo si hay latencia en la atención de la interrupción
- Como el programa principal ahora no hace nada se puede utilizar la instrucción **`wfi`** para esperar una interrupción en un modo que consume menos energía

Ejemplo de aplicación (3-temporizador)

```
# Función para inicializar el timer con un periodo determinado
# a0: Valor del periodo del temporizador
    .global timer_start
    .type   timer_start,@function
timer_start:
    la      t0, SYSTIMER_BASE   # Se apunta a la base del timer
    addi    a1, t0, MTIMER      # Calcula la dirección de MTIMER
    addi    a2, t0, MTIMERCMP   # Calcula la dirección de
MTIMERCMP
    j       add_double          # Suma el periodo al registro

# Rutina de atención de la interrupción del timer
# a0: Valor del periodo del temporizador
    .global timer_update
    .type   timer_update,@function
timer_update:
    la      t0, SYSTIMER_BASE  # Se apunta a la base del timer
    addi    a1, t0, MTIMERCMP  # Calcula la dirección de MTIMERCMP
    addi    a2, t0, MTIMERCMP  # Calcula la dirección de MTIMERCMP
    j       add_double         # Suma el periodo al registro
```

Ejemplo de aplicación (3-temporizador)

```
# Rutina de servicio para la interrupción
    .text
    .align   6
timer_handler:
    li       a0, PERIODO
    jal      timer_update              # Define el nuevo valor para el comparador
    li       a0, 3
    jal      cambiar_salida            # Cambia el estado del led verde de la placa
    mret                               # Retorna al programa principal

# Función principal que se ejecuta al completar el reset
    .globl   _start
    .section .init
_start:
    jal      configurar_placa          # Configura el hardware de la placa
    la       t0, timer_handler
    csrw     CSR_MTVEC, t0             # Define la rutina para las interrupciones
    la       t0, PERIODO
    jal      timer_start               # Define el valor inicial del comparador
    li       t0, MIE_MTIE
    csrs     CSR_MIE, t0               # Habilita la interrupción del temporizador
    csrsi    CSR_MSTATUS,MSTATUS_MIE   # Habilita las interrupciones del procesador

main_loop:
    wfi
    j        main_loop
```

Ejemplo de aplicación (4-completo)

- Se pueden combinar los dos ejemplos anteriores
  - Se utiliza el temporizador para hacer parpadear el led verde
  - Al presionar la tecla *cancelar* se ejecuta una instrucción inválida
- Las dos excepciones producen un trap a la misma rutina de servicio
- Se puede utilizar el registro **`mcause`** para implementar la identificación por software
  - Si el origen es el temporizar se actualiza el comparador
  - Sino asume un error, prende el led rojo y queda en un lazo

Ejemplo de aplicación (4-completo)

```
# Rutina de servicio para la interrupción
    .text
    .align 6
    .type  timer_handler,@function
timer_handler:
    csrr   a0, CSR_MCAUSE       # Lee el registro de causa
    la     a1, 0x80000007
    beq    a0, a1, timer_event  # Si es la interrupción del temporizador
    li     a0, 3
    jal    apagar_salida        # Apaga el led verde de la placa
    li     a0, 1
    jal    cambiar_salida       # Prende el led rojo de la placa
halt:
    j      halt                 # Lazo infinito

timer_event:
    li     a0, PERIODO
    jal    timer_update         # Define el nuevo valor para el comparador
    li     a0, 3
    jal    cambiar_salida       # Cambia el estado del led verde de la placa
    mret                        # Retorna al programa principal
```

Ejemplo (5-completo-en-c)

- El ejemplo anterior también se puede escribir en lenguaje C
  - En este caso las configuraciones se hacen desde el main
- El fabricante provee funciones que permiten leer o escribir los registros de control y estado del procesador
  - Simplemente encapsulan la instrucción en ensamblador en una función C
- También provee funciones para leer el registro del temporizador del sistema
  - Pero no provee una función para leer o escribir el registro de comparación
  - La escritura del registro de comparación debe seguir una secuencia específica, por lo tanto se debe hacer en ensamblador
  - Se pueden embeber instrucciones en ensamblador dentro del lenguaje C, e incluso vincularlas a variables y funciones del C

Ejemplo de aplicación

```c
/**
 * @brief Función para leer el valor del registro de comparación del temporizador del sistema
 *
 * @return Valor actual del registro de comparación del temporizador
 */
uint64_t timer_compare(void) {
    uint32_t hi = *(volatile uint32_t *)(TIMER_CTRL_ADDR + TIMER_MTIMECMP + 4);
    uint32_t lo = *(volatile uint32_t *)(TIMER_CTRL_ADDR + TIMER_MTIMECMP);
    return ((uint64_t)hi << 32) | lo;
}
```

```c
/**
 * @brief Función para actualizar el registro de comparación del temporizador en forma segura
 *
 * @param value Nuevo valor para el registro de compraación del temporizador del sistema
 */
void timer_update(uint64_t value) {
    volatile void * addr = (uint8_t *)(TIMER_CTRL_ADDR + TIMER_MTIMECMP);
    // Se escribe la parte baja con 0xFFFFFFFF para prevenir una falsa comparación
    asm volatile("sw %0, 0(%1)" : : "r"(-1U), "r"(addr));
    // Se actualiza la parte alta
    asm volatile("sw %0, 0(%1)" : : "r"((uint32_t)(value >> 32)), "r"(addr + 4));
    // Se actualiza la parte baja
    asm volatile("sw %0, 0(%1)" : : "r"((uint32_t)(value)), "r"(addr));
}
```

Las interrupciones en RISC-V: PLIC

- Como extensión se define un controlador denominado Platform Interrupt Controller (PLIC) que complementa al CLINT y provee:
  - Capacidad para rutear una interrupción global a múltiples procesadores (o núcleos) del sistema
  - Un sistema de arbitraje para permitir la atención de una interrupción en sistema muti-núcleo
  - Niveles de prioridad independientes para cada interrupción
  - Capacidad para enmascarar los pedidos con prioridades inferiores a un nivel determinado
- Este conjunto está pensado para sistemas de propósito general con sistemas operativos complejos

Las interrupciones en RISC-V: CLIC

- Como alternativa se define un controlador denominado Core Local Interrupt Controller (CLIC) que reemplaza al CLINT y provee:
  - Soporte para múltiples interrupciones externas
  - Anidamiento independiente del nivel de privilegio
  - Niveles de prioridad independientes para cada interrupción
  - Capacidad para enmascarar los pedidos con prioridades inferiores a un nivel determinado
  - Posibilidad de conectarse a un controlador global PLIC
- El CLIC está pensado para sistemas embebidos simples con o sin sistemas operativos de tiempo real

Las interrupciones en el Bumblebee

- El procesador Bumblebee integrado en el GD32VF103 implementa un ECLIC: una versión mejorada del CLIC
- Por compatibilidad el procesador inicia en modo CLINT con una única rutina de servicio para todas las excepciones
- En este modo el ECLIC esta deshabilitado y solo funcionan las interrupciones internas
- Para recibir interrupciones externas se debe habilitar el ECLIC
- Esta configuración solo afecta a las interrupciones, mientras que las excepciones siguen comportándose de la misma forma que si el ECLIC estuviera apagado
- Cuando el ECLIC esta habilitado gestiona todas las interrupciones, incluidas las internas del temporizador y de software
- Para cada fuente se puede decidir si se utiliza una rutina de servicio común a todas las interrupciones o el esquema vectorizado

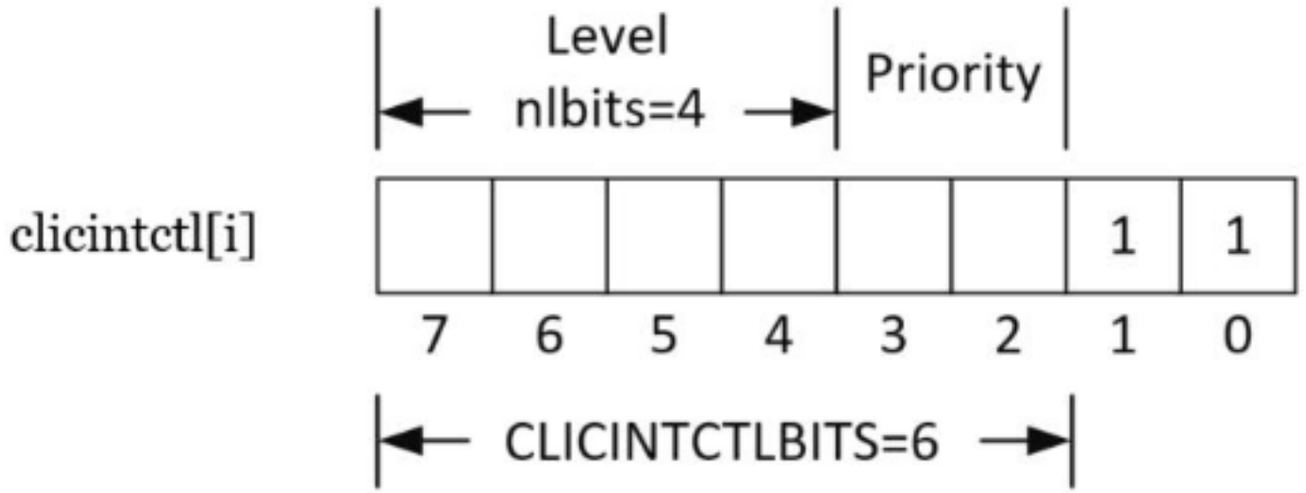
Interrupciones por flanco o nivel

- Las interrupciones en el ECLIC se puede configurar para dispararse por flanco o por nivel
- Cuando se configuran por nivel el bit de interrupción pendiente ip refleja el nivel del pedido de interrupción
  - Las escrituras en este bit son ignoradas
  - El pedido solo puede ser borrado por el dispositivo
- Cuando se configuran por flanco el bit memoriza el flanco definido como activo
  - Se puede borrar o incluso fijar por software
  - Se puede elegir el flanco ascendente o descendente
  - Se borra automáticamente al atender la interrupción

Las prioridades de las interrupciones

- El procesador Bumblebee utiliza un esquema de nivel y prioridad para resolver la simultaneidad y el anidamiento de las interrupciones
- Utiliza 8 bits para definir la prioridad de las interrupciones
  - Un numero mas grande implica mayor prioridad
  - No todos los bits están realmente implementados
    - Se puede consultar cuantos bits están implementados usando el campo **`CLICINTCTLBITS`** del registro **`clicinfo`**
  - Se puede configurar cuantos bits se utilizan como nivel
    - Si se configura el valor 0 entonces todas las interrupciones tienen un solo nivel y no hay anidamiento
    - Se si configura un numero mayor que la cantidad real de bits implementados entonces solo se utilizan los implementados

Las prioridades de las interrupciones



CLICINTCTLBITS	nlbits	clicintctl[i]	the value of level
0	2	. . . . .	255
1	2	L	127, 255
2	2	LL	63, 127, 191, 255
3	3	LLL	31, 63, 95, 127, 159, 191, 223, 255
4	1	LPPP. . . .	127, 255

Configurando las interrupciones

- Utilizar el registro **`mtvec`** para definir la rutina de servicio de las excepciones y habilitar el ECLIC
  - La dirección de la rutina de servicio debe tener los últimos 6 bits en cero
  - Los dos últimos bits del registro se deben configurar en 1 para habilitar el ECLIC
- Utilizar el registro **`mtvt2`** para definir la rutina de servicio común para las interrupciones no vectorizadas
  - La dirección de la rutina de servicio debe tener los últimos 2 bits en cero
  - El último bit del registro se debe configurar en 1 para utilizar este registro como puntero a la rutina de servicio
- Utilizar el registro **`mtvt`** para definir el vector de interrupciones
- Utilizar el registro **`cliccfg`** para definir cuantos bits se utilizan para definir los niveles de las interrupciones

Configurando una interrupción

- Si la interrupción será vectorizada definir el punto de entrada a la rutina de servicio en la entrada **`[i]`** correspondiente del vector de instrucciones
- Utilizar el registro **`clicintctl[i]`** para definir el nivel y la prioridad de la interrupción
- Utilizar el registro **`clicintattr[i]`** para el flanco o nivel de disparo y si la interrupción será vectorizada o no
- Utilizar el registro **`clicintip[i]`** para borrar cualquier pedido pendiente de interrupción
- Utilizar el registro **`clicintie[i]`** para habilitar la interrupción
- Finalmente habilitar las interrupciones en el procesador usando el campo **`mie`** del registro **`mstatus`**

Ejemplo de aplicación (6-vectorizadas)

- El ECLIC también gestiona las interrupciones externas
  - Al habilitar el ECLIC cambia la forma de habilitar la interrupción del temporizador
  - Ahora se habilita por los registros del ECLIC y no por el registro de control MIE
- Se debe definir el vector de interrupciones
  - Se debe definir la dirección de la rutina de servicio en la entrada correspondiente del vector de interrupciones
  - El temporizador del sistema se identifica como la fuente 7 de interrupciones en el ECLIC

Programando en lenguaje C

- El compilador de C debe ejecutar una serie de tareas antes de llamar a la función **`main()`**
  - Preparar el entorno de ejecución inicializando la pila, el heap y las las variables globales
  - Inicializar la librería estándar de c (**`libc`**), por ejemplo abriendo los archivos **`stdin`** y **`stdout`**
  - Configurar aspectos específicos de la plataforma como la velocidad de reloj del procesador
- En los ejemplos que vienen presentando estas tareas no se ejecutan y por lo tanto no se puede ejecutar código en C
  - La rutina de inicio no realiza las llamadas adecuadas a las funciones generadas por la librería estándar para efectuar estas configuraciones
- El fabricante suele proveer todo el código necesario para configurar el entorno del compilador y, a continuación, ejecutar la función **`main()`**

Programando en lenguaje C

- En muchos casos las configuraciones de inicio están muy vinculadas a las rutinas de servicio de interrupciones y excepciones
  - En muchos procesadores el reset es un caso particular de una interrupción
- Es habitual que estas definiciones incluyan el vector de interrupciones
  - Para poder definir el vector de interrupciones se definen funciones con nombres razonablemente representativos que no hacen nada
  - Estas funciones se declaran como weak (débiles) lo que le indica al enlazador que pueden ser redefinidas sin generar un error de símbolo duplicado
- En este contexto para definir la rutina de servicio de una interrupción basta con definir una función propia con el mismo nombre que el fabricante le asignó en el vector de interrupciones

Ejemplo (7-vectorizadas-en-c)

- Los pasos de configuración son los mismos
  - Se pueden usar las bibliotecas provistas por el fabricante para toda la configuración
- Se define la rutina de servicio para atender excepciones
  - Igual que en los ejemplos anteriores se utilizan los atributos para indicar al compilador que es una rutina de servicio
- Sin embargo en la rutina **`eclic_mtip_handler()`** no se utilizan los atributos porque la interrupción no está vectorizada
  - Como la linea 110 del archivo **`main.c`** esta comentada la interrupción es atendida por la rutina **`irq_entry()`** definida en **`entry.s`**
  - Esta rutina utiliza el registro de causa y el vector para llamar a **`eclic_mtip_handler()`** como una función normal
  - Esto se puede verificar con la pila de llamada del depurador

Interrupciones del GPIO

- Las lineas de GPIO se pueden utilizar para generar interrupciones
  - Se pueden seleccionar 16 lineas de GPIO para producir 7 interrupciones diferentes
- Las primeras 5 fuentes de interrupciones se llaman EXTI0 a EXTI4
  - Para cada EXTi se puede seleccionar una cualquiera de las lineas PTxi, es decir las que coinciden en numero con la interrupción
    - Por ejemplo se puede usar PTA0 para EXTI0 y PTB2 para EXTI2
- Las dos fuentes restantes se llaman EXTI5-9 y EXTI10-15
  - Para EXIT5-9 se pueden seleccionar hasta 5 lineas de GPIO de cualquier puerto pero solo una de cada uno de los números del 5 al 9
    - Por ejemplo se pueden usar PTC5, PTB6 y PTC9 para EXTI5-9
  - Para EXTI10-15 se pueden seleccionar hasta 6 lineas con las mismas reglas
- En todos los casos se puede elegir si la interrupción se dispara por flanco ascendente, descendente o ambos flancos
  - En EXTI5-9 y EXTI10-15 esta selección se puede hacer para cada linea de GPIO

Ejemplo de aplicación (8-teclado)

- Se configuran la teclas *aceptar* y *cancelar* para producir una interrupción
  - Por el número de las lineas usadas en las teclas la interrupción será la misma para todas las teclas: EXTI10-15
  - Se configura la teca apretar para que la interrupción ocurra solo al presionar la tecla
  - Se configura la teca cancelar para que la interrupción ocurra solo al soltar la tecla
- En la rutina de servicio se utiliza el registro de estado **`EXTI_PD`** para identificar la o las causas de la interrupción
  - Los pedidos se borran escribiendo un 1 en los bits que ya están en en 1 porque hay un pedido pendiente
  - Esto permite leer implementar un patrón seguro para borrar los pedidos que garantiza que no se pierda un pedido que llega durante el borrado