PROGRAMA UNIVERSITARIO DE IMAGINATION

Práctica 9 RVfpga
Entrada/Salida dirigida por interrupciones

1. INTRODUCCIÓN

En esta práctica se introduce el concepto de interrupciones y se muestra cómo usarlas en RVfpga. Las interrupciones se pueden generar por software o hardware. Esta práctica se centra en las interrupciones hardware que se activan por el cambio de valor en un pin físico. Específicamente, se comienza en la Sección 2 describiendo las diferencias entre **Entrada/Salida programada** y **Entrada/Salida dirigida por interrupciones**. Luego, se explica el funcionamiento del Controlador de Interrupciones del Sistema RVfpga, que es parte del core SweRV EH1 (Sección 3). En la Sección 4 se indica cómo configurar las interrupciones externas utilizando el Paquete de Soporte de Periféricos (PSP) y el Paquete de Soporte de Placas (BSP) de Western Digital, que son paquetes software que incluyen controladores para periféricos hardware. Por último, se presentan algunos programas de ejemplo (Sección 5) y se proponen algunos ejercicios (Sección 6) para utilizar y ampliar las interrupciones hardware del Sistema RVfpga.

2. ENTRADA/SALIDA PROGRAMADA VS. ENTRADA/SALIDA DIRIGIDA POR INTERRUPCIONES

Existen varios métodos para interactuar con los periféricos: Entrada/Salida programada, Entrada/Salida dirigida por interrupciones y acceso directo a memoria (DMA). En las prácticas 2-8, se utilizó **Entrada/Salida programada** para interactuar con los periféricos. En Entrada/Salida programada, el programa de usuario sondea continuamente la interfaz de Entrada/Salida y, dependiendo de su estado, reacciona en consecuencia. Por ejemplo, el *Ejercicio Básico* de la práctica 6 utilizaba Entrada/Salida programada leyendo continuamente los interruptores 0 y 1 para controlar la velocidad y dirección de un bloque de cuatro LEDs encendidos que se movía repetidamente de un lado a otro. La Entrada/Salida programada es muy sencilla de implementar y requiere muy poco soporte hardware, pero el sondeo continuo de la interfaz de Entrada/Salida mantiene al procesador ocupado haciendo un trabajo inútil.

La **Entrada/Salida dirigida por interrupciones** solventa este inconveniente y permite al programa reaccionar sólo cuando se produce un evento en el periférico. En este esquema, el periférico es responsable de enviar una señal (llamada **interrupción**) al procesador cuando ocurre algún evento - por ejemplo, el desbordamiento de un temporizador, la recepción de un carácter en una interfaz UART, la conmutación en un pulsador, etc. Cuando no ocurre ningún evento (es decir, no hay ninguna interrupción), el procesador continúa realizando trabajo útil. Cuando el procesador recibe una interrupción, detiene el programa que estaba ejecutando e invoca una *rutina de servicio de interrupción* (ISR por sus siglas en inglés), también llamada *gestor de interrupciones*. Una ISR es esencialmente una función con argumentos `void` que gestiona la interrupción, es decir, lee el nuevo valor del pulsador, realiza alguna acción relacionada con el desbordamiento del temporizador, etc.

Los procesadores generalmente soportan modos de gestión de interrupciones monovectoriales y multivectoriales. En el modo monovectorial (Figura 1), todas las interrupciones invocan la misma ISR. Así, cuando se produce una interrupción, el procesador detiene el programa principal y salta a la ISR común, que primero determina el origen de la interrupción y luego ejecuta el código ISR específico que corresponde a la causa de la interrupción identificada. En el modo multivectorial (Figura 2), cada interrupción invoca una ISR diferente. Así, cuando se genera una interrupción, primero se determina la causa de la interrupción y luego el programa salta a la ISR que corresponde a la causa identificada.

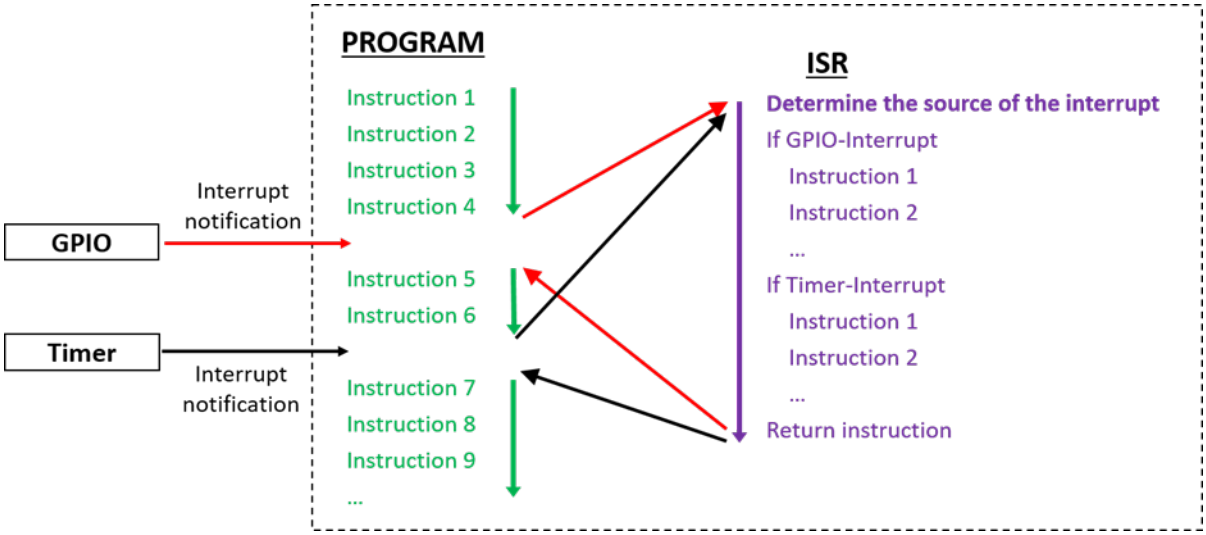

Figura 1. Ejemplo con 2 interrupciones en modo monovectorial

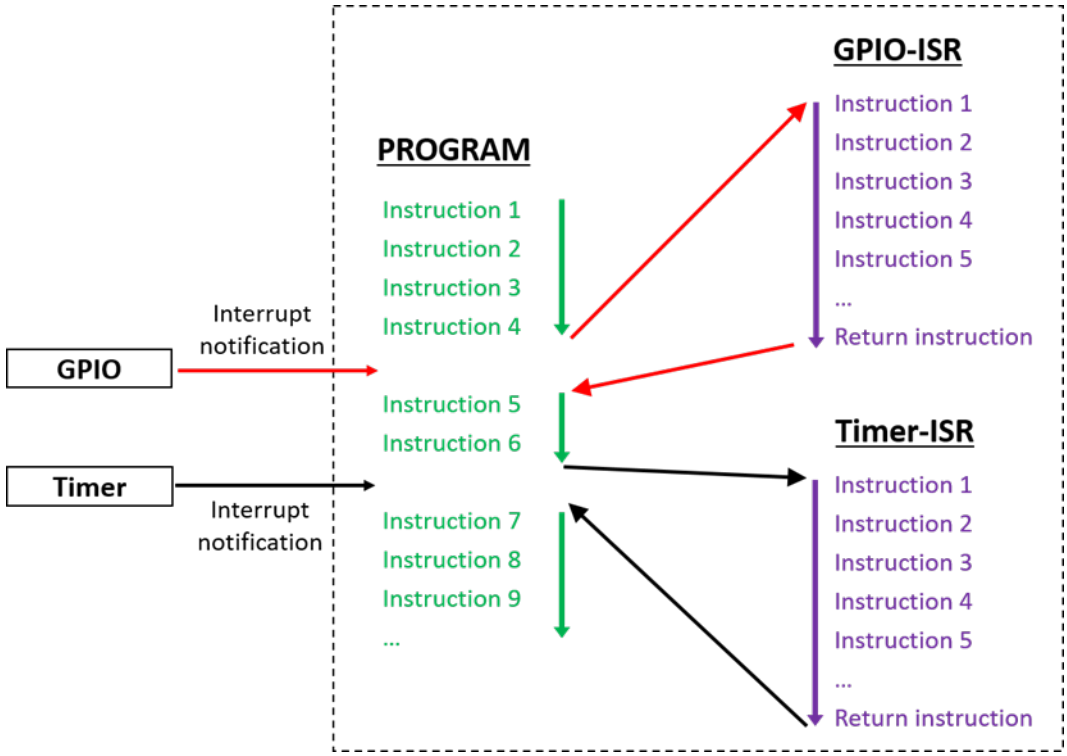

Figura 2. Ejemplo con 2 interrupciones en modo multivectorial

Los procesadores normalmente permiten que se dé prioridad a las interrupciones. No sólo se gestionarán primero las interrupciones de mayor prioridad, sino que una interrupción de mayor prioridad adelantará a una interrupción de menor prioridad aun estando esta última ya en proceso de gestión. Por ejemplo, supongamos que a una interrupción por conmutación de un pulsador se le fija prioridad 5, que a una interrupción por desbordamiento de temporizador se le otorga prioridad 7 y que el umbral se establece en 4 (por lo que ambas prioridades están por encima del umbral). Si el programa está ejecutando su flujo normal y se presiona el pulsador, se producirá una interrupción y el procesador llamará a la ISR, que leerá los datos del pulsador y procederá a la gestión de la interrupción.

Si un temporizador se desborda mientras la ISR del pulsador está activa, la propia ISR se interrumpirá para que el procesador pueda gestionar inmediatamente el desbordamiento del temporizador. Cuando termine, volverá a atender la interrupción del pulsador antes de retornar al programa principal[1].

3. CONTROLADOR DE INTERRUPCIÓN PROGRAMABLE PROVISTO POR SWERV EH1

El core SweRV EH1 soporta interrupciones como se describe en las siguientes referencias y como se resume a continuación:

- **[PRM v1.1]** Revisión 1.1 (31 de mayo de 2019), Capítulo 5, RISC-V SweRV™ EH1 Programmer's Reference Manual

- **[ISM v1.12]** Versión 20211028-signoff (29 de octubre de 2021), Capítulo 3, "The RISC-V Instruction Set Manual - Volume II: Privileged Architecture

Las interrupciones externas en el core SweRV EH1 (ver [PRM v1.7]) están modeladas en gran medida según la especificación RISC-V PLIC (*Platform-Level Interrupt Controller.* Sin embargo, el controlador de interrupciones está asociado con el core, no con la plataforma. Por lo tanto, se utiliza el término PIC (*Programmable Interrupt Controller*), más general, para referirse al controlador disponible en el core SweRV EH1. El PIC tiene las siguientes características principales:

- Admite hasta 255 fuentes de interrupción externas (desde 1 (prioridad más alta) hasta 255 (prioridad más baja)); cada fuente tiene su propia habilitación.
- Más allá de la numeración de las fuentes, proporciona 15 niveles de prioridad adicionales; se dispone de dos esquemas de prioridad: 1-15 (en donde 1 es la prioridad más baja), o 0-14 (en donde 14 es la prioridad más baja). A cada fuente se le puede asignar una prioridad.
- Proporciona soporte para umbral de prioridad programable que permite desactivar las interrupciones de menor prioridad.
- Soporte para interrupciones externas vectorizadas, encadenamiento de interrupciones e interrupciones anidadas.

Figura 3 muestra una versión simplificada del sistema de interrupciones del Sistema RVfpga. Todas las unidades funcionales que generan interrupciones se llaman **fuentes de interrupción externas**. Las fuentes de interrupción externas indican una solicitud de interrupción mediante el envío de una señal asíncrona al **PIC** con señales que terminan en *_irq* (una abreviatura para la solicitud de interrupción). En esta práctica de laboratorio, se muestra cómo utilizar las interrupciones del temporizador y de la GPIO; estas unidades generan interrupciones utilizando las señales *ptc_irq* y *gpio_irq*, respectivamente.

Cada fuente de interrupción externa se conecta a una interfaz dedicada (situada dentro del PIC), una estructura hardware responsable de sincronizar la solicitud de interrupción con el dominio del reloj del core y de convertir la señal de solicitud a un formato de solicitud de interrupción común (es decir, o activo en alto/bajo o disparado por nivel) para el PIC. El PIC sólo puede tratar a la vez una solicitud de interrupción por cada fuente de interrupción. El

[1] D. Harris and S. Harris. "*Digital Design and Computer Architecture*". Second Edition – 2012. Morgan Kaufmann Publishers (San Francisco, CA, United States). ISBN:978-0-12-394424-5.

PIC evalúa todas las solicitudes de interrupción pendientes y habilitadas y elige la interrupción de mayor prioridad con el identificador de fuente más bajo. Luego compara esta prioridad con el umbral de prioridad programable y, para soportar interrupciones anidadas, la prioridad del gestor de interrupciones si es que hay alguno en ejecución. Si la prioridad de la solicitud elegida es superior a ambos umbrales, el PIC envía una notificación de interrupción al core, que detiene la ejecución del programa principal y salta a la correspondiente ISR, como se ilustra en la Figura 1 (modo monovectorial) y en la Figura 2 (modo multivectorial).

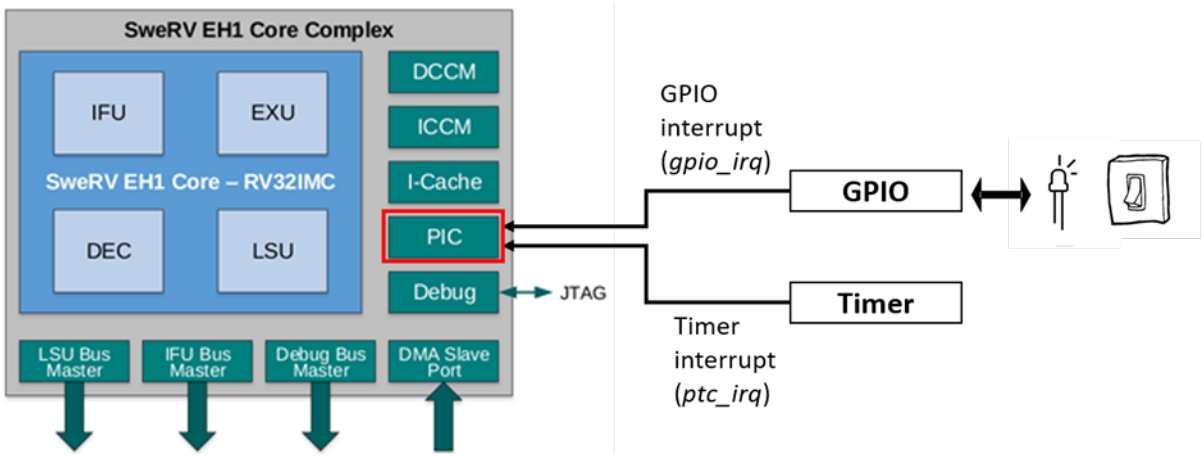

Figura 3. Sistema de interrupción del Sistema RVfpga

Las principales funcionalidades del PIC se resumen en los siguientes pasos básicos:

1) Activación/desactivación: el PIC permite activar/desactivar las interrupciones externas
2) Configuración: el PIC se puede configurar para escuchar interrupciones externas de distinta polaridad (activa-alta/activa-baja) o tipo (activada por flanco / activada por nivel). El PIC también permite asignar ISRs a diferentes direcciones de memoria.
3) Filtrado y asignación de prioridades: el PIC permite asignar niveles de prioridad a las interrupciones. Cuando el programa principal se está ejecutando, el PIC selecciona la interrupción activada con el nivel de prioridad más alto.
4) Notificación: una vez que el PIC selecciona la interrupción con la mayor prioridad, notifica al core que detenga la ejecución del programa principal para saltar a la rutina que gestiona la interrupción elegida.
5) Adelantamiento: si se habilitan las interrupciones anidadas, se permite que una interrupción de mayor prioridad adelante a otra cuya gestión está en curso.

4. CONFIGURACIÓN DE LAS INTERRUPCIONES EXTERNAS EN EL SweRV EH1

Al igual que cualquier otro periférico, el PIC se configura utilizando registros mapeados en memoria que son accesibles al usuario mediante instrucciones de load/store. Utilizar el sistema de interrupciones a nivel de registro sería posible pero muy complejo; afortunadamente, el Paquete de Soporte del Procesador (PSP) y el Paquete de Soporte de la Placa (BSP) de WD (https://github.com/westerndigitalcorporation/riscv-fw-infrastructure) incluyen varias funciones que proporcionan un enfoque mucho más simple para implementar programas que utilizan interrupciones. En la Tabla 1 se describen las principales funciones y macros que se requieren para configurar las interrupciones externas. Para llevar a cabo un análisis más exhaustivo, el Apéndice al final de este documento

proporciona una descripción de los diferentes registros disponibles y los pasos a seguir para la configuración a nivel de registro y el uso del PIC.

Tabla 1. Funciones y macros básicas para configurar las interrupciones externas

Cabecera	Descripción
`void` **`pspInterruptsSetVectorTableAddress`**`(` `void* pVectTable);`	Prepara la dirección de la tabla de vectores

`void`
`pspExternalInterruptSetVectorTableAddress``(`
`void* pExtIntVectTable);`
	Prepara dirección de la tabla de vectores de las interrupciones externas

`void` **`bspInitializeGenerationRegister`**`(` `u32_t uiExtInterruptPolarity)`	Pone el Registro de Generación en su estado inicial
`void` **`bspClearExtInterrupt`**`(` `u32_t uiExtInterruptNumber)`	Borra el disparador que genera la interrupción externa
`void` **`pspExtInterruptSetPriorityOrder`**`(` `u32_t uiPriorityOrder);`	Establece el orden de prioridad (estándar o reservado)
`void` **`pspExtInterruptsSetThreshold`**`(` `u32_t uiThreshold);`	Establece el umbral de prioridad de las interrupciones externas en el PIC

`void`
`pspExtInterruptsSetNestingPriorityThreshold``(`
`u32_t uiNestingPriorityThreshold);`
	Establece el umbral de prioridad de anidamiento de las interrupciones externas en el PIC

`void` **`pspExtInterruptSetPolarity`**`(`
`u32_t uiIntNum,`
`u32_t uiPolarity);`
	Establece la polaridad (activa en alto o activa en bajo) de una líena de interrupción dada

`void` **`pspExtInterruptSetType`**`(`
`u32_t uiIntNum,`
`u32_t uiIntType);`
	Establece el tipo (disparada por nivel o por flanco) de una línea de interrupción dada

`void` **`pspExtInterruptClearPendingInt`**`(` `u32_t uiIntNum);`	Borra la indicación de interrupción pendiente para una línea de interrupción dada

`void` **`pspExtInterruptSetPriority`**`(`
`u32_t uiIntNum,`
`u32_t uiPriority);`
	Establece la prioridad de una línea de interrupción dada

`void` **`pspExternalInterruptEnableNumber`**`(` `u32_t uiIntNum);`	Habilita una líena de interrupción específica en el PIC
`void` **`pspInterruptsEnable`**`(` `void);`	Habilita las interrupciones (en todos los niveles de privilegio) independientemente de su estado previo
`void` **`pspInterruptsDisable`**`(` `u32_t *pOutPrevIntState);`	Deshabilita las interrupciones y devuelve el estado actual de interrupción en cada uno de los niveles de provilegio

Más adelante en esta práctica se proporcionan ejemplos de rutinas de servicio de interrupción (ISR). Éstas siguen los pasos descritos a continuación para configurar las interrupciones en el Sistema RVfpga, en base a las funciones de la Tabla 1. Tenga en cuenta que, además de configurar el PIC, también deben configurarse los periféricos que generan las interrupciones externas (esto se describirá más adelante para cada uno de los periféricos que se utilizan en los ejemplos y los ejercicios).

INICIALIZACIÓN POR DEFECTO DEL SISTEMA DE INTERRUPCIÓN:

1. En el modo multivectorial, establezca la dirección base de la tabla de direcciones de las interrupciones vectorizadas externas. Utilice las funciones `pspInterruptsSetVectorTableAddress` y

`pspExternalInterruptSetVectorTableAddress`.

2. Establecezca el Registro de Generación en su estado inicial. Utilice la función `bspInitializeGenerationRegister`.
3. Asegúrese de que los disparadores de interrupciones externas estén desactivados. Utilice la función `bspClearExtInterrupt`.
4. Establezca los valores por defecto para el orden de prioridad (función `pspExtInterruptsSetPriorityOrder`), el umbral (función `pspExtInterruptsSetThreshold`) y el umbral de prioridad de anidamiento (función `pspExtInterruptsSetNestingPriorityThreshold`).

INICIALIZACIÓN DE CADA FUENTE DE INTERRUPCIÓN:

1. Para cada fuente de interrupción, establezca la polaridad (activo a alto / activo a bajo) y el tipo (disparada por nivel o por flanco) utilizando las funciones `pspExtInterruptSetPolarity` y `pspExtInterruptSetType`
2. Borre cualquier interrupción pendiente utilizando la función `pspExtInterruptClearPendingInt`.
3. Establezca el nivel de prioridad para cada fuente de interrupción externa mediante la función `pspExtInterruptSetPriority`.
4. Habilite las interrupciones para la fuente de interrupción externa apropiada con la función `pspExternalInterruptEnableNumber`.
5. En el modo multivectorial, para cada fuente de interrupción externa, escriba la dirección del controlador correspondiente en la tabla de direcciones de interrupciones vectorizadas externas.

TAREA AVANZADA: Para comprender mejor estas funciones básicas, consulte el código PSP ubicado en *.platformio/packages/framework-wd-riscv-sdk/psp* y el código BSP ubicado en *.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp*. De especial interés son los archivos que se listan a continuación, algunos de ellos incluidos en el subdirectiorio *api_inc*.

- *bsp_external_interrupts.h*: creación de interrupciones externas en RVfpga
- *psp_interrupts_eh1.h*: proporciona información y las APIs para registrar las ISR en el core EH1
- *psp_ext_interrupts_eh1.h*: define las interfaces de interrupciones externas de psp para el SweRV EH1
- *psp_macros_eh1.h*: define las macros del PSP para el SweRV EH1
- *psp_csrs_eh1.h*: definiciones de los CSRs del SweRV EH1

También se recomienda analizar por lo menos una de estas funciones hasta el nivel de registro. Para ello, puede utilizar la información proporcionada en el apéndice, que describe cómo el PIC del core SweRV EH1 configura y gestiona las interrupciones externas a nivel de registro.

TAREAS AVANZADAS: También se recomienda que analice y ejecute la demostración sobre interrupciones externas proporcionada por Western Digital en https://github.com/westerndigitalcorporation/riscv-fw-infrastructure y disponible como un proyecto de PlatformIO en: *[RVfpgaPath]/RVfpga/Labs/Lab9/WD_demo_external_int_Original*. Si todo funciona correctamente, debería ver los siguientes mensajes en la terminal serie:

```
Hello from SweRV core running on NexysA7
Core list:
        EH1 = 11
        EL2 = 16
Running demo on core 11...
----------------------------------------
SweRVolf version 255.255255 (SHA 000000ef) (dirty 128)
----------------------------------------
External Interrupts tests passed successfully
```

5. EJEMPLOS

En esta sección se proporcionan ejemplos de conversión de programas con Entrada/Salida programada a programas con Entrada/Salida dirigida por interrupciones. Se muestran tres ejemplos que ilustran los diferentes problemas inherentes a la Entrada/Salida programada (primer y segundo ejemplo) y a continuación se indica cómo estos problemas pueden ser resueltos fácilmente utilizando un esquema de Entrada/Salida dirigido por interrupciones (tercer ejemplo).

A. Programa LED-Switch_C-Lang

El programa *LED-Switch_C-Lang* (véase Figura 4) invierte el estado del LED más a la derecha cada vez que se produce una transición de 0→ 1 en el interruptor más a la derecha. El programa está disponible en
[RVfpgaPath]/RVfpga/Labs/Lab9/LED-Switch_C-Lang.c

Después de la inicialización de los periféricos, el programa entra en un bucle infinito que compara el estado actual del interruptor con su estado anterior y, en caso de que se detecte una transición de 0→ 1, invierte el estado del LED (obsérvese que, cuando se produce una transición de 1→ 0, no ocurre nada).

En los ejemplos y ejercicios anteriores escritos en C, se definieron macros para acceder a los registros de ENTRADA/SALIDA (`READ_GPIO`, `READ_Reg`, `WRITE_GPIO`, `WRITE_Reg`, etc.). En este ejemplo, se utilizan en cambio dos macros definidas en el PSP con el mismo propósito: `M_PSP_READ_REGISTER_32`, que lee un registro de 32 bits proporcionado como argumento, y `M_PSP_WRITE_REGISTER_32`, que escribe un registro de 32 bits con el valor proporcionado en el segundo argumento. Recuerde que, para poder utilizar estas macros, debe incluir la línea `framework = wd-riscv-sdk` en el archivo *platformio.ini* (este es el valor por defecto cuando se crea un proyecto con RVfpga como objetivo) y la línea `#include "psp_api.h"` al principio del programa (Figura 4, línea 1).

```c
#include "psp_api.h"

#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

int main ( void )
{
    int LED_state, Sw_current_state, Sw_previous_state;

    /* Configure LEDs and Switches */
    M_PSP_WRITE_REGISTER_32(GPIO_INOUT, 0xFFFF);

    /* Init states */
    LED_state = 0;
    M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
    Sw_previous_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;

    while (1) {
        /* Invert LED-0 when SW-0 goes high */
        Sw_current_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;
        if(Sw_current_state==1 && Sw_previous_state==0){
            LED_state = !LED_state;
            M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
        }
        Sw_previous_state = Sw_current_state;
    }

    return(0);
}
```

Figura 4. Programa *LED-Switch_C-Lang*

TAREA: Analizar el programa *LED-Switch_C-Lang* para entenderlo en detalle. Si es necesario, puede usar el depurador para analizar el programa paso a paso.

El programa funciona correctamente, pero es muy ineficiente, ya que el procesador no hace otra cosa que leer/escribir los interruptores/LEDs. Obviamente, se quiere que el procesador haga más cosas que sólo comunicarse con los dispositivos de ENTRADA/SALIDA.

B. Programa LED-Switch_7SegDispl_C-Lang

En este segundo ejemplo, *LED-Switch_7SegDispl_C-Lang*, se amplía el programa *LED-Switch_C-Lang* con un segundo periférico: los displays de 7 segmentos. El programa realiza dos tareas:

- Al igual que el primer ejemplo, invierte el LED de la derecha cada vez que se produce una transición 0→ 1 en el interruptor de la derecha.
- Muestra una cuenta ascendente en los displays de 7 segmentos de 8 dígitos, que se incrementa alrededor de una vez por segundo. Observe que, por simplicidad, se crea el retardo de un segundo con un bucle `for` (en el Ejercicio 1 se utilizará el temporizador de la Práctica 8 para este propósito).

Puede ver este programa en la Figura 5 y lo puede encontrar en:
[RVfpgaPath]/RVfpga/Labs/Lab9/LED-Switch_7SegDispl_C-Lang.c

Después de algunas inicializaciones, el programa entra en un bucle infinito que compara el estado actual del interruptor con el anterior y, en caso de que se detecte una transición 0→ 1, invierte el estado del LED. A continuación se incrementa el valor mostrado en los displays de 7 segmentos de 8 dígitos y se genera un retardo. Véase el cuadro rojo de la Figura 5.

```c
#include "psp_api.h"

#define SegEn_ADDR    0x80001038
#define SegDig_ADDR   0x8000103C

#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

int main ( void )
{
    int i, LED_state, Sw_current_state, Sw_next_state, count=0;

    /* Configure LEDs and Switches */
    M_PSP_WRITE_REGISTER_32(GPIO_INOUT, 0xFFFF);

    /* Configure 7-Seg Displays */
    M_PSP_WRITE_REGISTER_32(0x80001038, 0x0);

    /* Init states */
    LED_state = 0;
    M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
    Sw_current_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;

    while (1) {
        /* Invert LED-0 when SW-0 goes high */
        Sw_next_state = (M_PSP_READ_REGISTER_32(GPIO_SWs) >> 16) & 0x1;
        if(Sw_current_state==0 && Sw_next_state==1){
            LED_state = !LED_state;
            M_PSP_WRITE_REGISTER_32(GPIO_LEDs, LED_state);
        }
        Sw_current_state = Sw_next_state;

        /* Increase 7-Seg Displays */
        M_PSP_WRITE_REGISTER_32(SegDig_ADDR, count);
        count++;

        /* Delay */
        for(i=0;i<1000000;i++);
    }

    return(0);
}
```

Figura 5. Programa *LED-Switch_7SegDispl_C-Lang*

TAREA: Analizar el programa *LED-Switch_7SegDispl_C-Lang* para entenderlo en detalle. Si es necesario, puede usar el depurador para analizar el programa paso a paso.

Tenga en cuenta que, en este caso, el programa ni siquiera funciona correctamente en algunas situaciones. Por ejemplo, una transición 0 →1→ 0 en el interruptor que se produce dentro del bucle de retardo no será detectada nunca. Además, se sigue teniendo el mismo problema que en el ejemplo anterior: el procesador está ocupado todo el tiempo sólo leyendo/escribiendo los dispositivos o creando un retardo.

¿Cómo se podrían mejorar estas situaciones? La respuesta es la **ENTRADA/SALIDA dirigida por interrupciones**. En el siguiente ejemplo y en los ejercicios propuestos en la siguiente sección, se muestra cómo resolver todos estos problemas e implementar programas que sean más eficientes y funcionen correctamente en todas las situaciones.

C. Programa LED-Switch_7SegDispl_Interrupts_C-Lang

En este último ejemplo (*[RVfpgaPath]/RVfpga/Labs/Lab9/LED-Switch_7SegDispl_Interrupts_C-Lang.c*), se muestra cómo usar ENTRADA/SALIDA dirigida por interrupciones para leer el estado del interruptor de más a la derecha. El uso de esta estrategia resuelve el problema de que el programa no detecte las transiciones del interruptor que ocurren durante el bucle de retardo. Sin embargo, observe que el problema de tener el procesador ocupado en un bucle de retardo aún persiste (se abordará este problema en el Ejercicio 1).

La nueva función **`main`**, que se muestra en la Figura 7, realiza las siguientes tareas:

- Inicializar el sistema de interrupción:
    - Inicialización por defecto de las interrupciones: invoca la función

`DefaultInitialization` (línea 119), que se muestra en la Figura 8.
    - Establece un umbral específico, invocando la función `pspExtInterruptsSetThreshold(5)` (línea 120). Las interrupciones externas cuya prioridad no supere este umbral se ignorarán.
- Inicializar la línea de interrupción externa IRQ4:
    - Inicializar la línea IRQ4: invoca la función `ExternalIntLine_Initialization` (línea 123) para la línea 4 de interrupción, con una prioridad de 6 y `GPIO_ISR` como Rutina de Servicio de Interrupción. Se analiza esta función en la Figura 9.
    - Conectar la IRQ4 con la línea de interrupción de la GPIO (línea 124). Esto se hace activando el bit 0 de la palabra 0x80001018 (etiquetada como `Select_INT` en el ejemplo). Este registro mapeado en memoria del Controlador del Sistema contiene 2 bits (ver Figura 6): el bit 0, llamado *irq_gpio_enable*, que se usa cuando está a 1 para conectar la línea de interrupción de la GPIO con IRQ4; y el bit 1, llamado *irq_ptc_enable*, que cuando está a 1 conecta la línea de interrupción del temporizador con IRQ3. Por ahora, basta con que el usuario conozca esta funcionalidad de alto nivel; más adelante, en el Ejercicio 2, se explicará en detalle la implementación Verilog, de modo que pueda modificarla como parte de ese ejercicio.

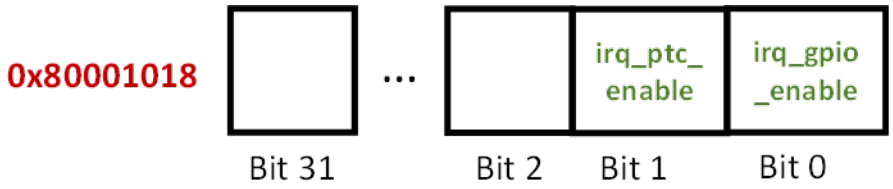

Figura 6. Registro 0x80001018 del Sistema RVfpga.

- Inicializar los periféricos (en este ejemplo, la GPIO y los displays de 7 segmentos):
    - Invoca la función `GPIO_Initialización` en la línea 127. Se analizará esa función en la Figura 10.
    - Habilitar los ocho displays de 7 segmentos (línea 128).
- Habilitar las interrupciones:
    - Invoca la función `pspInterruptsEnable` (línea 131) y la macro `M_PSP_SET_CSR` (línea 132). Las constantes D_PSP_MIE_NUM y D_PSP_MIE_MEIE_MASK están definidas por el PSP de WD.
- Finalmente, se escriben los displays de 7 segmentos y se establece un retardo dentro de un bucle que se repite siempre (líneas 134-141).

```
114  int main(void)
115  {
116    int count=0, i;
117
118    /* INITIALIZE THE INTERRUPT SYSTEM */
119    DefaultInitialization();                    /* Default initialization */
120    pspExtInterruptsSetThreshold(5);            /* Set interrupts threshold to 5 */
121
122    /* INITIALIZE INTERRUPT LINE IRQ4 */
123    ExternalIntLine_Initialization(4, 6, GPIO_ISR);   /* Initialize line IRQ4 with a priority of 6. Set GPIO_ISR as the Interrupt Service Routine */
124    M_PSP_WRITE_REGISTER_32(Select_INT, 0x1);         /* Connect the GPIO interrupt to the IRQ4 interrupt line */
125
126    /* INITIALIZE THE PERIPHERALS */
127    GPIO_Initialization();                      /* Initialize the GPIO */
128    M_PSP_WRITE_REGISTER_32(SegEn_ADDR, 0x0);   /* Initialize the 7-Seg Displays */
129
130    /* ENABLE INTERRUPTS */
131    pspInterruptsEnable();                      /* Enable all interrupts in mstatus CSR */
132    M_PSP_SET_CSR(D_PSP_MIE_NUM, D_PSP_MIE_MEIE_MASK);  /* Enable external interrupts in mie CSR */
133
134    while (1) {
135      /* Increase 7-Seg Displays */
136      M_PSP_WRITE_REGISTER_32(SegDig_ADDR, count);
137      count++;
138
139      /* Delay */
140      for(i=0;i<50000000;i++);
141    }
142  }
```

Figura 7. Función *main*.

La función **`DefaultInitialization`**, que se muestra en la Figura 8, lleva a cabo los pasos explicados en el apartado "INICIALIZACIÓN POR DEFECTO DEL SISTEMA DE INTERRUPCIÓN" de la Sección 4:

- Configura la tabla de vectores (líneas 53 y 56). Obsérvese que, en este ejemplo, el array `G_Ext_Interrupt_Handlers` almacena la tabla de vectores.
- Inicializa el registro utilizado para disparar las IRQ (línea 59).
- Borra todas las interrupciones externas (en nuestro caso IRQ3 e IRQ4) en las líneas 61-65. Las constantes D_BSP_FIRST_IRQ_NUM y D_BSP_LAST_IRQ_NUM están definidas por el BSP de WD a 3 y 4, respectivamente.
- Establece el umbral y las prioridades por defecto (líneas 68, 71 y 74). De nuevo, las constantes utilizadas por estas funciones son definidas por el PSP de WD.

```
48  void DefaultInitialization(void)
49  {
50    u32_t uiSourceId;
51
52    /* Register interrupt vector */
53    pspInterruptsSetVectorTableAddress(&M_PSP_VECT_TABLE);
54
55    /* Set external-interrupts vector-table address in MEIVT CSR */
56    pspExternalInterruptSetVectorTableAddress(G_Ext_Interrupt_Handlers);
57
58    /* Put the Generation-Register in its initial state (no external interrupts are generated) */
59    bspInitializeGenerationRegister(D_PSP_EXT_INT_ACTIVE_HIGH);
60
61    for (uiSourceId = D_BSP_FIRST_IRQ_NUM; uiSourceId <= D_BSP_LAST_IRQ_NUM; uiSourceId++)
62    {
63      /* Make sure the external-interrupt triggers are cleared */
64      bspClearExtInterrupt(uiSourceId);
65    }
66
67    /* Set Standard priority order */
68    pspExtInterruptSetPriorityOrder(D_PSP_EXT_INT_STANDARD_PRIORITY);
69
70    /* Set interrupts threshold to minimal (== all interrupts should be served) */
71    pspExtInterruptsSetThreshold(M_PSP_EXT_INT_THRESHOLD_UNMASK_ALL_VALUE);
72
73    /* Set the nesting priority threshold to minimal (== all interrupts should be served) */
74    pspExtInterruptsSetNestingPriorityThreshold(M_PSP_EXT_INT_THRESHOLD_UNMASK_ALL_VALUE);
75  }
```

Figura 8. Función *DefaultInitialization*

La función **`ExternalIntLine_Initialization`**, que se muestra en la Figura 9, realiza los pasos explicados en el apartado " INICIALIZACIÓN DE CADA FUENTE DE INTERRUPCIÓN" de la Sección 4:

- Configura el tipo y la polaridad de la interrupción de la IRQ4 (las constantes

utilizadas por estas funciones están definidas por el PSP de WD) y borra cualquier posible interrupción pendiente en la interfaz correspondiente (líneas 81, 84 y 87).
- Establece la prioridad para la IRQ4 (línea 90).
- Habilita las interrupciones de IRQ4 en el PIC en la línea 93.
- Registra la Rutina de Servicio de Interrupción de la GPIO (GPIO_ISR) en la tabla de vectores (en la línea 96), que se almacena en el array `G_Ext_Interrupt_Handlers`.

```
78  void ExternalIntLine_Initialization(u32_t uiSourceId, u32_t priority, pspInterruptHandler_t pTestIsr)
79  {
80    /* Set Gateway Interrupt type (Level) */
81    pspExtInterruptSetType(uiSourceId, D_PSP_EXT_INT_LEVEL_TRIG_TYPE);
82
83    /* Set gateway Polarity (Active high) */
84    pspExtInterruptSetPolarity(uiSourceId, D_PSP_EXT_INT_ACTIVE_HIGH);
85
86    /* Clear the gateway */
87    pspExtInterruptClearPendingInt(uiSourceId);
88
89    /* Set IRQ4 priority */
90    pspExtInterruptSetPriority(uiSourceId, priority);
91
92    /* Enable IRQ4 interrupts in the PIC */
93    pspExternalInterruptEnableNumber(uiSourceId);
94
95    /* Register ISR */
96    G_Ext_Interrupt_Handlers[uiSourceId] = pTestIsr;
97  }
```

Figura 9. Función *ExternalIntLine_Initialization*

La función **`GPIO_Initialization`**, que se muestra en la Figura 10, realiza las siguientes tareas:
- Configura los pines GPIO como entrada/salida e inicializa los LEDs a 0 (líneas 103 y 104).
- Configura las interrupciones de la GPIO. (Para entender mejor la funcionalidad de cada registro GPIO, utilice la Especificación Básica de GPIO, disponible en: *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Periféricos/gpio/docs/gpio_spec.pdf*.)
    - *RGPIO_INTE*: determina qué pines de propósito general generan una interrupción (línea 107).
    - *RGPIO_PTRIG*: determina el flanco que genera una interrupción (línea 108).
    - *RGPIO_INTS*: borra las interrupciones de todos los pines (línea 109).
    - *RGPIO_CTRL*: el bit menos significativo de este registro habilita la generación de interrupciones (línea 110).

```
100  void GPIO_Initialization(void)
101  {
102    /* Configure LEDs and Switches */
103    M_PSP_WRITE_REGISTER_32(GPIO_INOUT, 0xFFFF);        /* GPIO_INOUT */
104    M_PSP_WRITE_REGISTER_32(GPIO_LEDs, 0x0);            /* GPIO_LEDs */
105
106    /* Configure GPIO interrupts */
107    M_PSP_WRITE_REGISTER_32(RGPIO_INTE, 0x10000);       /* RGPIO_INTE */
108    M_PSP_WRITE_REGISTER_32(RGPIO_PTRIG, 0x10000);      /* RGPIO_PTRIG */
109    M_PSP_WRITE_REGISTER_32(RGPIO_INTS, 0x0);           /* RGPIO_INTS */
110    M_PSP_WRITE_REGISTER_32(RGPIO_CTRL, 0x1);           /* RGPIO_CTRL */
111  }
```

Figura 10. Función *GPIO_Initialization*.

Finalmente, se invoca la ISR (es decir, la función **`GPIO_ISR`** que se muestra en la Figura 11) cuando se dispara una interrupción en la GPIO. Esta ISR (Rutina de Servicio de Interrupción) realiza las siguientes tareas:

- Se lee el estado actual de los LEDs (línea 35).
- Los LEDs se invierten y enmascaran (líneas 36-37).
- Se escriben los LEDs con el nuevo valor (línea 38).
- Se borra la interrupción de la GPIO (línea 41).
- Se borra la interrupción externa de la IRQ4 (línea 44).

```
30  void GPIO_ISR(void)
31  {
32    unsigned int i;
33
34    /* Write the LED */
35    i = M_PSP_READ_REGISTER_32(GPIO_LEDs);            /* RGPIO_OUT */
36    i = !i;                                           /* Invert the LEDs */
37    i = i & 0x1;                                      /* Keep only the right-most LED */
38    M_PSP_WRITE_REGISTER_32(GPIO_LEDs, i);            /* RGPIO_OUT */
39
40    /* Clear GPIO interrupt */
41    M_PSP_WRITE_REGISTER_32(RGPIO_INTS, 0x0);          /* RGPIO_INTS */
42
43    /* Stop the generation of the specific external interrupt */
44    bspClearExtInterrupt(4);
45  }
```

Figura 11. Función *GPIO_ISR*.

TAREA: Analizar el programa *LED-Switch_7SegDispl_Interrupts_C-Lang* para entenderlo en detalle. Puede comparar la implementación con las explicaciones de la Sección 4 y, si es necesario, usar el depurador para analizar el programa paso a paso.

6. EJERCICIOS

Ejercicio 1. Modificar el programa *LED-Switch_7SegDispl_Interrupts_C-Lang* para incluir una segunda fuente de interrupción, en este caso generada por el temporizador. Recuerde que un temporizador puede actuar como un generador PWM, como un temporizador o como un contador, por lo que generalmente se denomina unidad PTC.

- En el Sistema RVfpga, la interrupción del temporizador se conecta a la IRQ3 poniendo a uno el bit 1 (*irq_ptc_enable*) de la palabra 0x80001018 (ver Figura 6).
- Crear una función que inicialice las interrupciones del PTC, similar a `GPIO_Initialization` del ejemplo anterior.
- Crear una segunda ISR denominada `PTC_ISR`. Debería ser similar a `GPIO_ISR` del programa *LED-Switch_7SegDispl_Interrupts_C-Lang*, pero debe invocarse usando IRQ3. `PTC_ISR` debe gestionar y borrar la interrupción del temporizador.

Una vez que se haya implementado y depurado el programa, utilice las funciones de PSP `pspExtInterruptsSetThreshold(threshold)` y `pspExtInterruptSetPriority(interrupt_source, priority)` para analizar diferentes combinaciones de las prioridades y el umbral. Obsérvese que incluso se pueden modificar las prioridades en tiempo de ejecución; por ejemplo, se puede mostrar la cuenta de los displays de 7 segmentos hasta 10, y luego dejar de contar modificando la prioridad de la fuente de interrupción externa apropiada.

Ejercicio 2. Modificar RVfpgaNexys para incluir una tercera fuente de interrupción procedente de la segundo GPIO que diseñó en la Práctica 6 para controlar los pulsadores de la placa (GPIO2). Son posibles dos enfoques diferentes para realizar este ejercicio:

- Puede conectar la interrupción de la GPIO2 a una fuente de interrupción externa no utilizada. El SweRV EH1 proporciona hasta 255 líneas de interrupción diferentes y hasta ahora sólo se han usado 2 de ellas. El inconveniente de este enfoque es que se necesita modificar las librerías de WD.

- Puede conectar la interrupción GPIO2 a IRQ4, de modo que el módulo GPIO (que se conecta a los LED e interruptores) y GPIO2 (que se conecta a los pulsadores) utilicen un modo de interrupción monovectorial. Aunque el modo multivectorial es preferible en algunas situaciones, la ventaja de este enfoque es que se puede reutilizar el BSP.

A continuación, se dan una serie de pautas para el segundo enfoque, proporcionando detalles sobre la implementación de bajo nivel de las interrupciones en el Sistema RVfpga.

Figura 12 muestra el circuito que conecta las diversas fuentes de interrupción (interrupción GPIO, interrupción del temporizador - y las fuentes de interrupción disponibles originalmente en el core SweRVolf, que aquí no se analizan ni se utilizan) con *IRQ4* e *IRQ3*. Específicamente, *IRQ4* está conectada a la GPIO cuando *irq_gpio_enable* = 1 (Figura 6), mientras que *IRQ3* se conecta al temporizador cuando *irq_ptc_enable* = 1 (Figura 6). Cuando *irq_gpio_enable* = *irq_ptc_enable* = 0, *IRQ4* e *IRQ3* están conectadas con las fuentes de interrupción originales de SweRVolf, que no se utilizan en esta práctica (si está interesado en utilizar estas fuentes de interrupción, puede encontrar más información en https://github.com/chipsalliance/Cores-SweRVolf).

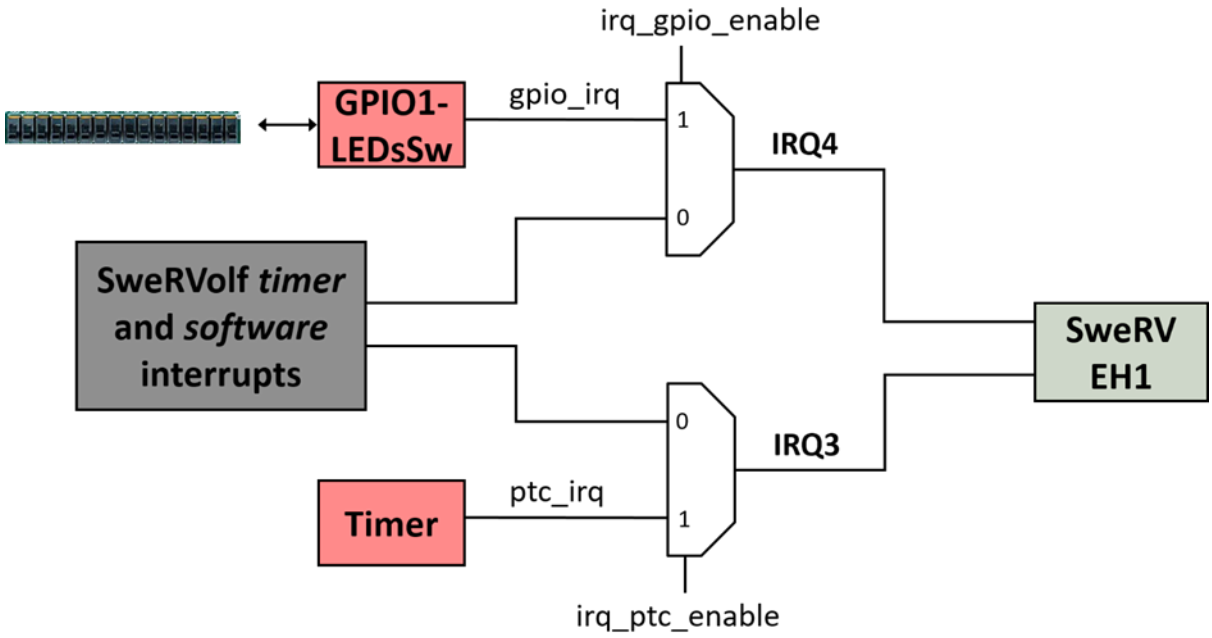

Figura 12. Implementación lógica: conexión de las interrupciones de GPIO y temporizador con IRQ4 e IRQ3 respectivamente

Figura 13 muestra el segmento de código Verilog del módulo **swervolf_core** que implementa la conexión entre las fuentes de interrupción y las *IRQ4* e *IRQ3*. La interrupción de la GPIO se conecta a *IRQ4* cuando la señal *irq_gpio_enable* es 1 (parte superior del cuadro rojo). La interrupción del temporizador se conecta a *IRQ3 cuando* la señal *irq_ptc_enable es 1 (parte inferior del cuadro rojo*). Cuando ambas señales son 0 (código no resaltado en la figura), las fuentes de interrupción implementadas en SweRVolfX se conectan a *IRQ3* e *IRQ4*.

```
123     always @(posedge i_clk) begin
124        o_wb_ack <= i_wb_cyc & !o_wb_ack;
125
126        nmi_int   <= 1'b0;
127        nmi_int_r <= nmi_int;
128
129        // GPIO Interrupt through IRQ4. Enable by setting bit 0 of word 0x80001018
130        if (irq_gpio_enable & gpio_irq) begin
131            sw_irq4 <= 1'b1;
132        end
133
134        // Timer (PTC) Interrupt through IRQ3. Enable by setting bit 1 of word 0x80001018
135        if (irq_ptc_enable & ptc_irq) begin
136            sw_irq3 <= 1'b1;
137        end
138
139        // SweRVolf simple timer and software interrupts. Enable by resetting bits 0 and 1 of word 0x80001018
140        if (!irq_gpio_enable & !irq_ptc_enable) begin
141
142            if (sw_irq3_edge)
143              sw_irq3 <= 1'b0;
144            if (sw_irq4_edge)
145              sw_irq4 <= 1'b0;
146
147            if (irq_timer_en)
148              irq_timer_cnt <= irq_timer_cnt - 1;
149
150            if (irq_timer_cnt == 32'd1) begin
151              irq_timer_en <= 1'b0;
152              if (sw_irq3_timer)
153                sw_irq3 <= 1'b1;
154              if (sw_irq4_timer)
155                sw_irq4 <= 1'b1;
156              if (!(sw_irq3_timer | sw_irq4_timer))
157                nmi_int <= 1'b1;
158            end
159
160        end
```

Figura 13. Implementación Verilog: se resaltan en rojo las conexiones de las interrupciones de GPIO y del temporizador con IRQ4 e IRQ3, respectivamente.

En este ejercicio se debe ampliar la implementación anterior (Figura 12) para incluir una nueva fuente de interrupción conectada a *IRQ4* como muestra la Figura 14.

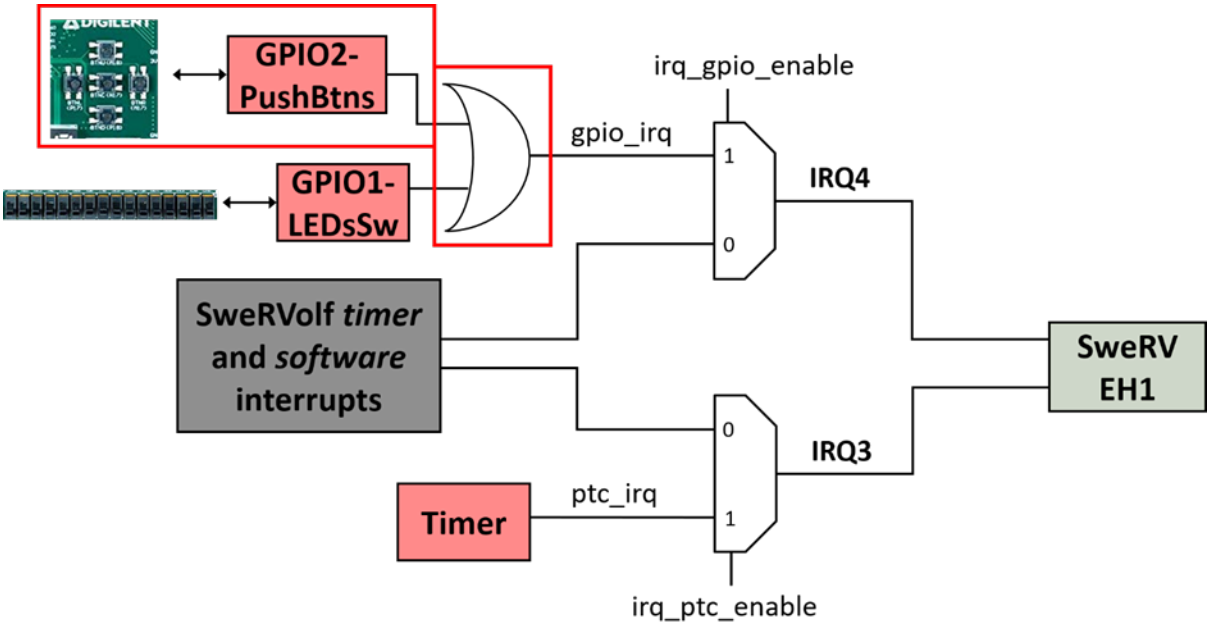

Figura 14. Implementación lógica: conexión de una segunda fuente de interrupción (proporcionada por la GPIO que lee los pulsadores) con IRQ4

A continuación se destacan algunas otras regiones del código Verilog que también se deberían comprender, aunque no es necesario modificarlas en este ejemplo.

- Las fuentes de interrupción se insertan en el procesador SweRV en la línea 599 del módulo **swervolf_core** (Figura 15). Aunque hay disponibles cuatro fuentes de interrupción, en esta práctica sólo nos interesan las fuentes *sw_irq4* y *sw_irq3*.

```
600         .extintsrc_req ({4'd0, sw_irq4, sw_irq3, spi0_irq, uart_irq}),
```

Figura 15. Fuentes de interrupción enviadas a SweRV

- Las señales de habilitación, *irq_gpio_enable* e *irq_ptc_enable* (accesibles en la dirección 0x80001018, ver Figura 6), son escritas por el core en las líneas 192-196 del módulo **swervolf_syscon** (Figura 16).

```
192             6: begin //0x18-0x1B
193                 if (i_wb_sel[0])
194                   irq_gpio_enable <= i_wb_dat[0];
195                   irq_ptc_enable <=  i_wb_dat[1];
196             end
```

Figura 16. Escritura del registro 0x80001018 del core SweRV

Estas señales de habilitación, *irq_gpio_enable* e *irq_ptc_enable*, las lee el módulo **swervolf_syscon** del core en las líneas 248-249 (ver Figura 17).

```
248         //0x18-0x1B
249         6 : o_wb_rdt <= {30'd0, irq_ptc_enable, irq_gpio_enable};
```

Figura 17. Lectura del registro 0x80001018 en el core SweRV

Ejercicio 3. Utilice la versión extendida de RVfpgaNexys que diseñó en el ejercicio anterior para implementar un programa en C que muestre una cuenta binaria ascendente en los LEDs, comenzando en 1. Cree un retardo con el temporizador, utilizando interrupciones, para esperar entre la visualización de cada valor incrementado de modo que los valores sean visibles por el ojo humano. Lea BTNC y úselo para cambiar la velocidad de la cuenta, y lea Switch[0] y utilícelo para reiniciar la cuenta cada vez que se pulsa.

Con su RVfpgaNexys extendido del Ejercicio 2, dispone ahora de tres posibles fuentes de interrupción:

- **GPIO** (interrupciones de los interruptores)
- **GPIO2** (interrupciones de los pulsadores, que diseñó en el ejercicio anterior, Ejercicio 2)
- **PTC** (el temporizador)

Dado que la implementación extendida de RVfpgaNexys del Ejercicio 2 tiene dos fuentes de interrupción que comparten la misma línea (*IRQ4*), la Rutina de Servicio de Interrupción correspondiente (`GPIO_ISR`) tiene que identificar el dispositivo que generó la interrupción. Puede extraer esa información de los registros GPIO.

APÉNDICE

Este apéndice describe cómo el Controlador de Interrupción Programable (PIC) del core SweRV EH1 gestiona las interrupciones externas a nivel de registro. El PIC utiliza los registros mapeados en memoria que se muestran en la Tabla 2. Hay que tener en cuenta que el espacio de memoria del PIC comienza en la dirección 0xF00C0000; Esta dirección se denomina *RV_PIC_BASE*. Las direcciones se expresan relativas a esta dirección base.

Tabla 2. Mapa de direcciones de los registros mapeados en memoria del PIC

Nombre	Direcciones (relativas a RV_PIC_BASE)	Descripción	Ubicación en el manual
meiplS	0x0004 - 0x0004+Smax*4-1	Registro de nivel de prioridad de las interrupciones externas	Tabla 5-2 de [PRM]
meipX	0x1000 - 0x1000+(*Xmax+1*)*4-1	Registro de interrupciones externas pendientes	Tabla 5-3 de [PRM]
meieS	0x2000 - 0x2000+Smax*4-1	Registro de habilitación de interrupciones externas	Tabla 5-4 de [PRM]
mpiccfg	0x3000 - 0x3003	Registro de configuración de interrupciones externas del PIC	Tabla 5-1 de [PRM]
meigwctrlS	0x4004 - 0x4004+Smax*4-1	Registro de configuración de la interfaz de interrupciones externas (sólo para interfaces configurables)	Tabla 5-11 de [PRM]
meigwclrS	0x5004 - 0x5004+Smax*4-1	Registro de borrado de la interfaz de interrupciones externas (sólo para interfaces configurables)	Tabla 5-12 de [PRM]

Todos los registros tienen 32 bits de ancho y son accesibles a través de las instrucciones de load/store, como es habitual para las Entrada/Salida mapeada en memoria. El tipo de acceso depende de los bits específicos a los que se quiera acceder (esto se puede ver en el RISC-V Programmer's Reference Manual o PRM).

Algunos de los registros tienen nombres parametrizados, que terminan en S o X. Pueden existir varias instancias de estos registros. El parámetro *S* se refiere al número de fuentes de interrupción externas, que en el SweRV EH1 equivale al número de interfaces. Así, los registros que terminan en '*S*' tienen de 1 a 255 instancias de registro disponibles. En esta práctica sólo se utilizan 2 fuentes de interrupción externas: **IRQ3** (asociada al temporizador), e **IRQ4** (asociada a la GPIO). El parámetro *X* se refiere a un grupo de 32 interfaces. Esto no significa que las interfaces estén agrupadas, pero agruparlas reduce el tamaño de la memoria necesaria para ciertos registros de 32 bits en los que 1 bit es suficiente para realizar una acción sobre un grupo de fuentes de interrupción externas. Este es el caso del registro de interrupciones externas pendientes, en el que un bit es suficiente para distinguir si la interrupción ha sido atendida o no. Para obtener más información sobre estos registros, la columna más a la derecha de la Tabla 1 indica el lugar dentro de [PRM] en el que se encuentra la descripción a nivel de bit (interrupción específica).

Además de los registros que se muestran en la Tabla 2, el PIC contiene Registros de Control y de Estado (CSR). El ISA estándar de RISC-V establece un espacio de codificación de 12 bits (*csr[11:0]*) para hasta 4096 CSRs. Por convenio, los 4 bits superiores de la dirección CSR (*csr[11:8]*) se utilizan para codificar la accesibilidad de lectura y escritura de los CSR según el nivel de privilegio. Los dos bits superiores (*csr[11:10]*) indican si el registro es de lectura/escritura (00, 01 o 10) o de sólo lectura (11). Los dos siguientes bits (*csr[9:8]*) codifican el nivel de privilegio más bajo que puede acceder al CSR. Hay disponible más información sobre los CSR en [PRM v1.7] y [ISM v1.11]. Tabla 3 lista los CSR que son útiles

para gestionar las interrupciones externas en el core SwerRV EH1. Se puede acceder a ellos mediante instrucciones de load/store específicas como *csrrw* o *csrrs* (lectura/escritura de CSR y lectura/ajuste de CSR).

Tabla 3. Mapa de direcciones de los CSR RISC-V no estándar del PIC.

Nombre	Número	Descripción	Ubicación
meivt	0xBC8	Registro de tabla de vectores de interrupción externa	Tabla 5-6 de [PRM]
meipt	0xBC9	Registro de umbral de prioridad de interrupción externa	Tabla 5-5 de [PRM]
meicpct	0xBCA	Registro de identificación de la interrupción externa / de activación de captura de nivel de prioridad	Tabla 5-8 de [PRM]
meicidpl	0xBCB	Registro de nivel de prioridad de identificación de la interrupción externa	Tabla 5-9 de [PRM]
meicurpl	0xBCC	Registro de nivel de prioridad actual de las interrupciones externas	Tabla 5-10 de [PRM]
meihap	0xFC8	Registro de puntero de dirección del gestor de interrupciones externas	Tabla 5-7 de [PRM]
mie	0x304	Registro de habilitación de interrupciones de la máquina	Tabla 9-1 de [PRM]

La columna más a la derecha de la Tabla 3 indica el lugar en [PRM v1.7] o [ISM v1.11] en el que se describe la información a nivel de bit del CSR particular (nótese que la descripción de los bits *mstatus* no se proporciona en [PRM v1.7] sino en [ISM v1.11]).

A. Configuración de las interrupciones externas

En esta subsección se resumen los pasos básicos necesarios para configurar una interrupción externa utilizando los registros previamente mencionados:

1. Deshabilite todas las interrupciones externas poniendo a cero el bit *miep* del CSR *mie*.
2. Configure el orden de prioridad mediante la escritura del bit *priord* del registro *mpiccfg.*
3. En el modo multivectorial, si no está configurado, establezca la dirección base de la tabla de direcciones de interrupciones vectorizadas externas escribiendo el campo base del registro *meivt*.
4. Establezca el umbral de prioridad escribiendo el campo *prithresh* del registro *meipt*.
5. Inicialice los umbrales de prioridad de anidamiento escribiendo '0' (o '15' para el orden de prioridad invertido) en el campo *clidpri* del registro *meicidpl* y en el campo *currpri* del registro *meicurpl*.
6. Para cada interfaz configurable *S*, establezca la polaridad (activa-alta/activa-baja) y tipo (disparada por nivel/ disparada por flanco) en el registro *meigwctrlS* y borre el bit IP escribiendo en el registro *meigwclrS* de la interfaz.
7. En el modo multivectorial, para cada fuente de interrupción externa S, escriba la dirección del gestor correspondiente en la tabla de direcciones de interrupciones vectorizadas externa.
8. Establezca el nivel de prioridad para cada fuente de interrupción externa *S* escribiendo el campo de prioridad correspondiente de los registros *meiplS*.
9. Habilite las interrupciones para las fuentes de interrupción externas apropiadas, mediante la activación del bit *inten* de los registros de *meieS* para cada fuente de interrupción *S*.

10. Active el bit *mei* del CSR *mstatus*.
11. Habilite todas las interrupciones externas activando el bit *miep* del CSR *mie*.

Estos son los pasos generales para las interfaces *S*. Sin embargo, en RVfpga sólo utilizamos 2 fuentes de interrupción (IRQ3 e IRQ4), cada una de las cuales tiene su propia interfaz. Además, hay que señalar que el orden no es totalmente estricto, ya que algunas acciones son intercambiables (por ejemplo, el paso 4 puede completarse antes que el paso 2). Además, debido a que cada función llama a pspInterruptsDisable al entrar, el paso 1 no es estrictamente necesario.

B. Modo de funcionamiento de las interrupciones externas

En esta subsección se describe cómo funciona el PIC una vez que se dispara una interrupción externa. Una vez que el evento deseado ocurre en la línea de interrupción externa (cable), tienen lugar las siguientes acciones:

1. El PIC decide qué interrupción pendiente tiene la mayor prioridad.
2. Cuando el hart objetivo (hilo hardware) toma la interrupción externa, deshabilita todas las interrupciones (es decir, pone a cero el bit *mie* en el registro *mstatus* de RISC-V) y salta al gestor de interrupciones externas.
3. El gestor de la interrupción externa escribe en el registro *meicpct* para activar la captura del identificador de la fuente de interrupción de la interrupción externa de mayor prioridad que está pendiente (en el registro *meihap*) y su correspondiente prioridad (en el registro *meicidpl)*.
4. El gestor lee entonces el registro *meihap* para obtener el identificador de la fuente de interrupción proporcionado en el campo *claimid*. Basándose en el contenido del registro *meihap*, el gestor de interrupciones externas salta al gestor específico de esta fuente de interrupción externa. Esto se puede observar en la Figura 18.
5. El gestor de interrupciones específico de la fuente (ISR) atiende la interrupción externa, y después:
    a. En el caso de las fuentes de interrupción disparadas por nivel, el gestor de interrupciones borra el estado en el IP del SoC que inició la solicitud de interrupción.
    b. Para las fuentes de interrupción disparadas por flanco, el gestor de interrupciones borra el bit IP en la interfaz de la fuente de interrupción escribiendo en el registro *meigwclrS*.

    Esto desestima la solicitud de interrupción de la fuente.
6. Mientras tanto, en segundo plano, el PIC sigue evaluando las interrupciones pendientes.

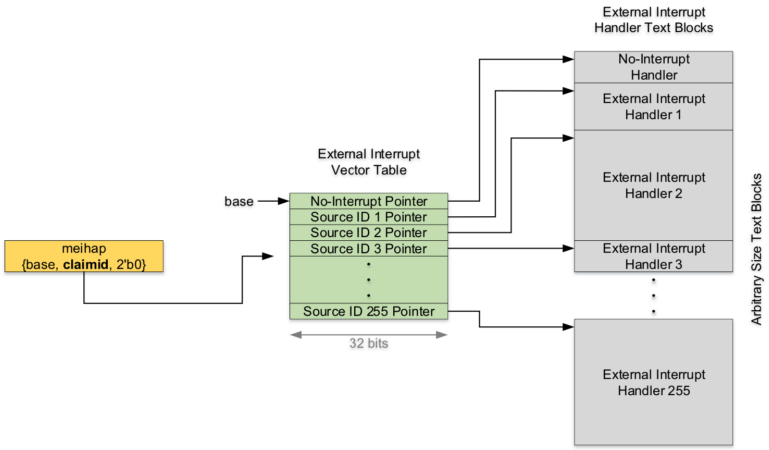

Figura 18. Interrupciones externas vectorizadas (tomado de [PRM v1.7])

Hay que señalar que éste es el modo de funcionamiento habitual. Las interrupciones anidadas (un máximo de 15) también se soportan en el core SweRV EH1. Para más información, por favor consulte el [PRM].