Práctica 3. Gestión básica de interrupciones, excepciones y traps.

1. Objetivo

En esta práctica se pretende que el alumno comprenda los mecanismos hardware que dan soporte a interrupciones, excepciones y traps, de forma que sea capaz de definir y utilizar rutinas básicas para su gestión. El dominio de estás rutinas permitirá al alumno entender como, a partir de ellas, se pueden definir otros servicios de más alto nivel propios de los sistemas software empotrados (drivers de dispositivos o llamadas al sistema), y como tratar las situaciones excepcionales que se pueden producir durante su ejecución.

2. Introducción

Los mecanismos de atención a los eventos (interrupciones, excepciones y traps) que proporcionan los procesadores facilitan el acceso a los recursos del sistema de forma protegida, asegurando su integridad frente a los usos inadecuados.

Por una parte, mediante instrucciones tipo *TRAP* (también llamadas interrupciones software) el usuario puede solicitar servicios que fueron configurados durante la etapa de inicialización del sistema. Estos servicios permiten gestionar abstracciones tales como los sistemas de archivos, los procesos o el acceso a puertos de comunicación.

El mecanismo de las interrupciones, por su parte, permite a los dispositivos externos solicitar la atención del procesador, con el fin de que se ejecute una rutina como respuesta a su petición. Gracias a este mecanismo es posible evitar el control por sondeo de los trabajos asignados a los dispositivos.

Las excepciones, finalmente, permiten definir qué hacer cuando el procesador se encuentra en un estado no estable como cuando ejecuta una instrucción cuyo código no es válido, o se produce una división por cero o un overflow en las instrucciones de operación aritmética.

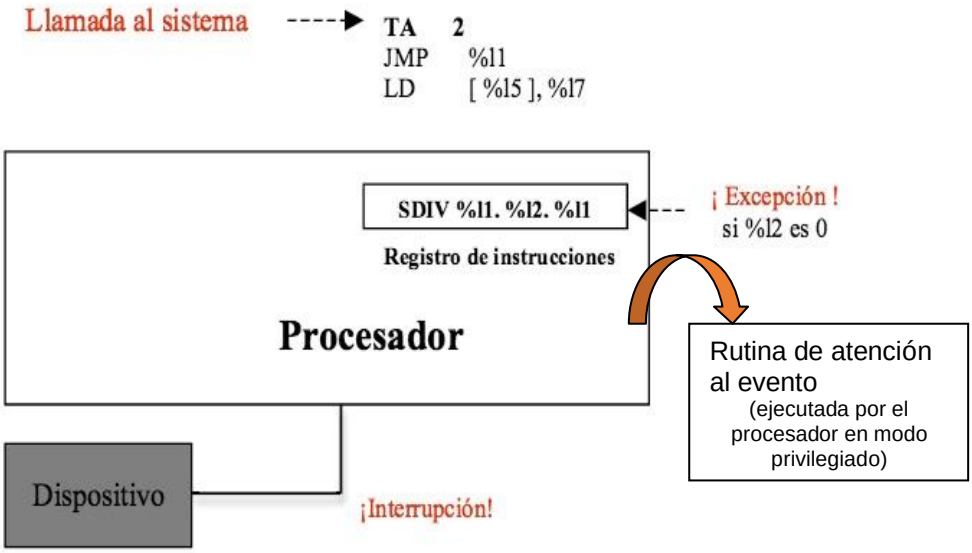

Figura 1. Mecanismos de atención a eventos proporcionados por el procesador.

3. Tabla de Vectores de Interrupción

La tabla de vectores de interrupción es una zona de memoria en la que el procesador localiza el código a ejecutar cuando se produce una interrupción, una excepción o se ejecuta una instrucción de tipo *TRAP*. La estructura de esta tabla depende de cada procesador, y puede tener tamaños muy diferentes en función de si se trata de un procesador de propósito general o de una pequeña CPU de 8 bits integrada en un microcontrolador. La figura 2 muestra una estructura de tabla de vectores de interrupción genérica donde cada evento es atendido por el procesador ejecutando el manejador cuya dirección se encuentra almacenada en el propio vector.

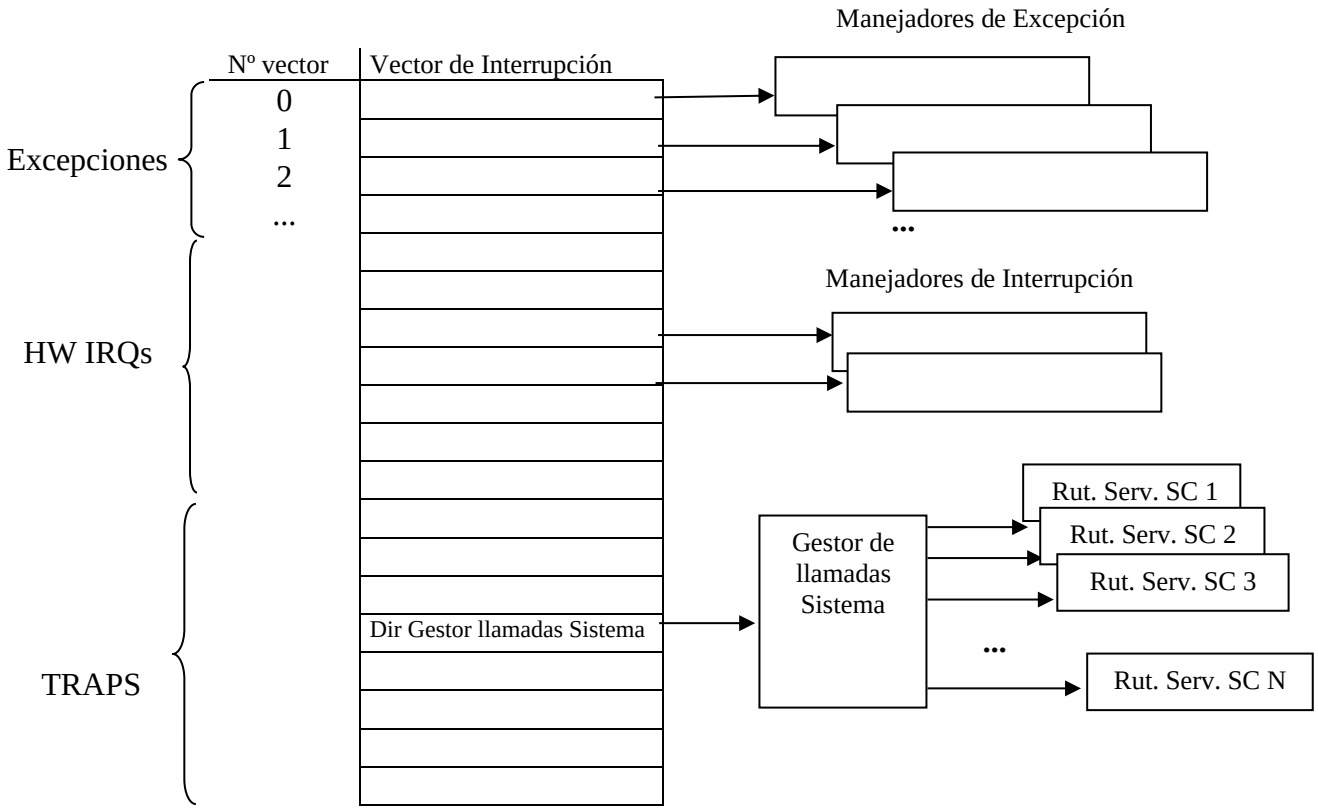

Figura 2. Tabla genérica de vectores de interrupción.

Esta tabla debe ser configurada durante la inicialiación del sistema (en el caso de contar con un Sistema Operativo, será éste el que se encarge de fijar la configuración inicial) . De esta forma, al ejecutarse posteriormente las aplicaciones, el sistema ya será estable, y **tendrá una respuesta controlada a cualquier evento que pueda ocurrir**.

Una configuración que los sistemas operativos emplean habitualmente es la de utilizar un único vector de la tabla para gestionar todas las llamadas al sistema. Una rutina, denominada *gestor de llamadas al sistema,* centraliza todas las peticiones, y en función de un parámetro recibido (bien a través de un registro predeterminado, bien a través de la pila) identifica la llamada al sistema solicitada, e invoca a su rutina de servicio.

Para las interrupciones hardware, una configuración habitual también es aquella en la que el software de sistema proporciona un manejador de interrupción estándar, con un prólogo y un epílogo (escrito en ensamblador) dadas las características específicas de la arquitectura. Éste, a su vez, se encarga de invocar al manejador de interrupción de usuario asociado a la interrupción hardware generada. Es este manejador el que el usuario puede instalar, y se puede utilizar un simple array para almacenarlo.

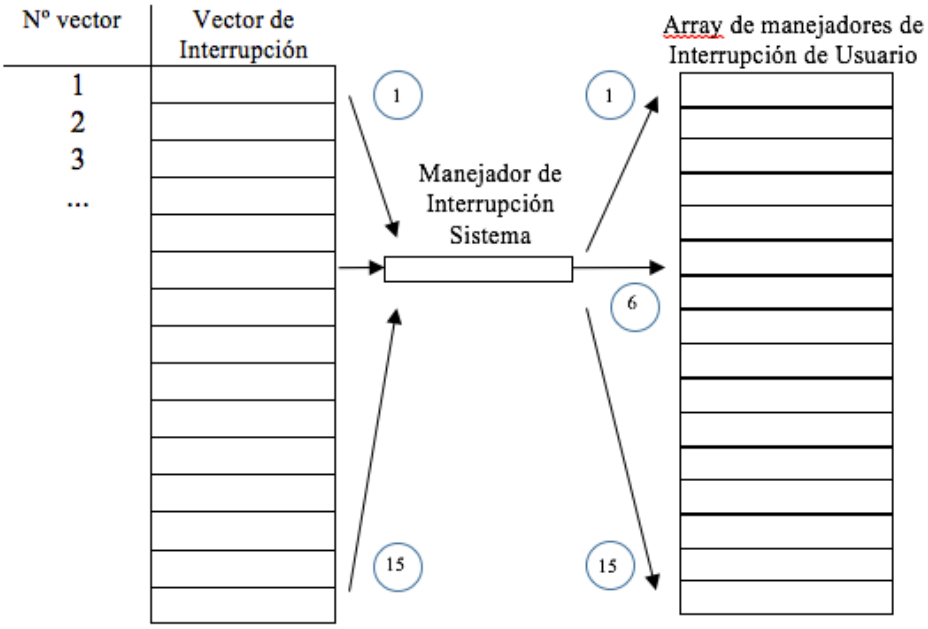

Figura 3. Configuración que utiliza un Manejador de Interrupción de Sistema (común para todas las interrupciones) y un Manejador de Interrupción de Usuario.

4. Tabla de Vectores de Interrupción del procesador LEON3

El procesador LEON3 presenta una estructura de tabla de vectores de interrupción en la que cada elemento de la tabla no es una dirección a la que saltar, sino un conjunto de 4 instrucciones que el procesador ejecuta diréctamente cada vez que se produce el evento.

Nº vector	Vector de Interrupción			
X	Instrucción-1	Instrucción-2	Instrucción-3	Instrucción-4

Esta estructura permite definir una respuesta rápida al evento. Sin embargo, cuando la respuesta es compleja, es necesario saltar a una rutina para poder definir la respuesta. La siguiente configuración de un vector, define cuatro instrucciones que permiten saltar a una función denominada `handler`, guardando en el registro `%l0` el registro de estado `%psr`, y en el registro `%l3` el número del vector. Ambos registros podrán ser utilizados por la rutina `handler` para realizar un tratamiento adecuado al evento.

Vector de Interrupción			
`rd %psr,%l0`	`sethi %hi(handler), %l4`	`jmp %l4 + %lo(handler)`	`mov vector,%l3`

La siguiente figura muestra la tabla de vectores de interrupción del procesador LEON3, donde **TT** indica el número de vector, **Trap** el evento al que está asociado el vector (puede ser una excepción, una interrupción hardware o un trap provocado por una instrucción) y **Pri** la prioridad con la que se atiende al evento. Las primeras 15 entradas corresponden a excepciones del procesador, las 15 siguientes son interrupciones externas, mientras que el rango definido por 0x80-0xFF está reservado para ser invocado a través de la instrucción *TRAPS*

Trap	TT	Pri	Description
reset	0x00	1	Power-on reset
write error	0x2b	2	write buffer error during data store
instruction_access_error	0x01	3	Error during instruction fetch
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction
privileged_instruction	0x03	4	Execution of privileged instruction in user mode
fp_disabled	0x04	6	FP instruction while FPU disabled
cp_disabled	0x24	6	CP instruction while Co-processor disabled
watchpoint_detected	0x0B	7	Hardware breakpoint match
window_overflow	0x05	8	SAVE into invalid window
window_underflow	0x06	8	RESTORE into invalid window
register_hadrware_error	0x20	9	register file EDAC error (LEON-FT only)
mem_address_not_aligned	0x07	10	Memory access to un-aligned address
fp_exception	0x08	11	FPU exception
cp_exception	0x28	11	Co-processor exception
data_access_exception	0x09	13	Access error during data load, MMU page fault
tag_overflow	0x0A	14	Tagged arithmetic overflow
divide_exception	0x2A	15	Divide by zero
interrupt_level_1	0x11	31	Asynchronous interrupt 1
interrupt_level_2	0x12	30	Asynchronous interrupt 2
interrupt_level_3	0x13	29	Asynchronous interrupt 3
interrupt_level_4	0x14	28	Asynchronous interrupt 4
interrupt_level_5	0x15	27	Asynchronous interrupt 5
interrupt_level_6	0x16	26	Asynchronous interrupt 6
interrupt_level_7	0x17	25	Asynchronous interrupt 7
interrupt_level_8	0x18	24	Asynchronous interrupt 8
interrupt_level_9	0x19	23	Asynchronous interrupt 9
interrupt_level_10	0x1A	22	Asynchronous interrupt 10
interrupt_level_11	0x1B	21	Asynchronous interrupt 11
interrupt_level_12	0x1C	20	Asynchronous interrupt 12
interrupt_level_13	0x1D	19	Asynchronous interrupt 13
interrupt_level_14	0x1E	18	Asynchronous interrupt 14
interrupt_level_15	0x1F	17	Asynchronous interrupt 15
trap_instruction	0x80 - 0xFF	16	Software trap instruction (TA)

Figure 4 LEON3 Interrupt Vector Table

Para poder manejar mejor la gestión de esta tabla de vectores de interrupción, es de gran utilidad definir funciones que, a partir del número del vector, y de un puntero a la rutina, gestione la forma en la que esta tabla debe completarse. Un prototipo de función con estas características sería el siguiente:

```
uint8_t leon3_set_event_handler( uint32_t vector_num ,
                                 void (* handler) (void));
```

El parámetro `vector_num` corresponde al número de vector, mientras que `handler` es el puntero a la función que queremos que se invoque para atender al evento.

5. Tarea a realizar. Creación de proyecto para LEON3

Creación de un nuevo proyecto denominado prac3 cuyo ejecutable sea para la plataforma Sparc Bare C. En ese proyecto crear dos subdirectorios **include** y **src**. En el directorio **src** añadir los archivos *leon3_bprint.c y leon_uart.c* de la práctica anterior e igualmente los archivos *leon3_ev_handling.S leon3_ev_handling.c y leon3_hw_irqs.c* que encontrarás en el archivo enlazado como *prac3_fuentes* en la página web. El contenido de cada uno de estos archivos se determina a continuación:

leon3_ev_handling.c

Implementa las siguientes rutinas de sistema para la gestión de los eventos:

- `leon3_install_hwirq_handler`. Función que instala una rutina de usuario para la atención a una interrupción externa siguiendo la estructura de la Figura 3. El prototipo de la función es el que se muestra a continuación, donde `hwirq_number` es el número de vector y `handler` es un puntero a la rutina a instalar:

```
uint8_t leon3_install_hwirq_handler(     uint8_t hwirq_number,
                                 void (* handler) (void));
```

- `leon3_set_trap_handler`. Función para la gestión de eventos que instala directamente la rutina de atención a un trap, indicando el número de vector del trap (`trap_vector_number`), y el `handler` que es un puntero a la rutina a instalar. El prototipo de esta función es el siguiente:

```
uint8_t leon3_set_trap_handler(    uint8_t trap_vector_number,
                                 void (* handler) (void));
```

En esta práctica utilizaremos ambas funciones para controlar la respuesta a diferentes eventos del sistema.

leon3_hw_irqs.c

Este módulo implementa rutinas de usuario para la gestión de los **registros de control de las interrupciones externas** (no para excepciones, ni traps). Implementa parcialmente las siguientes funciones:

- `leon3_mask_all_irqs`. Función que permite enmascarar los 15 niveles de interrupción externa (del 1 al 15) poniendo a 0 el registro IMASK (ubicado en la dirección 0x80000240). Su prototipo es el siguiente:

```
void leon3_maks_all_irqs();
```

- `leon3_unmask_all_irqs`. Función que permite desenmascarar los 15 niveles de interrupción externa (del 1 al 15) poniendo a 0xFE el registro IMASK (ubicado en la dirección 0x80000240). Su prototipo es el siguiente:

```
void leon3_unmaks_all_irqs();
```

- `leon3_mask_irq`. Función que permite enmascarar **uno de los 15 niveles** de interrupción externa poniendo a 0 en el registro IMASK el bit correspondiente al nivel (**sólo ese bit, el resto deben permanecer con el valor que tienen**). El número de nivel debe suministrarse mediante el parámetro `irq_level`. Su prototipo es el siguiente:

```
uint8_t leon3_mask_irq(uint8_t irq_level);
```

- `leon3_unmask_irq`. Función que permite desenmascarar **uno de los 15 niveles** de interrupción externa poniendo a 1 en el registro registro IMASK el bit correspondiente al nivel nivel (**sólo ese bit, el resto deben permanecer con el valor que tienen**). El número de nivel debe suministrarse mediante el parámetro `irq_level`. Su prototipo es el siguiente:

```
uint8_t leon3_unmask_irq(uint8_t irq_level);
```

- `leon3_force_irq`. Función que permite forzar el disparo de uno de los 15 niveles de interrupción externa poniendo a 1 en el registro IFORCE (registro ubicado en la dirección 0x80000208) el bit correspondiente al nivel (**sólo ese bit, el resto deben permanecer con el valor que tienen**). El número de nivel debe suministrarse mediante el parámetro `irq_level`. Su prototipo es el siguiente:

```
uint8_t leon3_force_irq(uint8_t irq_level);
```

- `leon3_clear_irq`. Función que permite borrar el bit de *pending* de uno de los 15 niveles de interrupción externa, poniendo a 1 en el registro ICLEAR (registro ubicado en la dirección 0x8000020C). La función cambia solamente el bit correspondiente al nivel (**el resto de bits se fijan a cero para no cambiar el status de su bit de *pending***). El número de nivel debe suministrarse mediante el parámetro `irq_level`. Su prototipo es el siguiente:

```
uint8_t leon3_clear_irq(uint8_t irq_level);
```

leon3_ev_handling_asm.S

Archivo en ensamblador que implementa las siguientes funciones de gestión de eventos:

- `leon3_trap_handler_enable_irqs` **rutina de atención a un trap** que permite habilitar todas las interrupciones que no estén enmascaradas en el registro IMASK. Para ello fija el Priority Interrupt Level (PIL) del registro de estado del procesador (PSR) al valor 0.

- `leon3_trap_handler_disable_irqs` **rutina de atención a un trap** que permite deshabilitar todas las interrupciones, independientemente de cómo esté configurada su máscara en el registro IMASK. Para ello fija el Priority Interrupt Level (PIL) del registro de estado del procesador (PSR) al valor 15.

- `leon3_sys_call_enable_irqs` **llamada al sistema**, efectuada a través de un *TRAP*, que permite ejecutar en modo supervisor la rutina de atención `leon3_trap_handler_enable_irqs`.

- `leon3_sys_call_disable_irqs(void)` **llamada al sistema**, efectuada a través de un *TRAP*, que permite ejecutar en modo supervisor a la rutina de atención `leon3_trap_handler_disable_irqs`.

Para completar la configuración de archivos, en el directorio **include** se deben añadir los archivos *leon3_bprint.h y leon_uart.h* de la práctica anterior. Añadir también los archivos *leon3_asm.h leon3_ev_handling.h y leon3_hw_irqs.h* que igualmente encontrarás en el archivo enlazado como *prac3_fuentes* en la página web. El contenido de estos tres últimos archivos es el siguiente:

- *leon3_asm.h* Macros en ensamblador de utilidad para la definición de las funciones implementadas en *leon3_ev_handling_asm.S*

- *leon3_hw_irqs.h* Archivo para la declaración de las funciones definidas en *leon3_hw_irqs.c*

- *leon3_ev_handling.h* Archivo para la declaración de las funciones definidas en *leon3_ev_handling.c* y *leon3_ev_handling_asm.S*

6. Prac3. Tarea a realizar.

1. Completar las funciones `leon3_mask_irq`, `leon3_unmask_irq` y `leon3_force_irq` para que se gestionen de forma adecuada las máscaras y el registro que fuerza el disparo de interrupciones.

2. Implementar en el archivo *leon3_bprint.c* la función `leon3_print_uint32` con el siguiente prototipo. Esta función implementa una función análoga a la función `leon3_print_uint8` que se realizó en la práctica anterior, pero trabajando con un entero de 32 bits. (Añade esta declaración a *leon3_bprint.h* para que se pueda usar la función)

```
 int8_t leon3_print_uint32(   uint32_t i);
```

3. Comprobar la validez de la implementación con el siguiente programa principal completando las partes que faltan:

```
#include "leon3_uart.h"
#include "leon3_bprint.h"
#include "leon3_hw_irqs.h"
#include "leon3_ev_handling.h"

void device_hw_irq_level_1_handler(void)

      leon3_print_string("Device HW IRQ user handler \n");
}

int main()
{

  //Instalar como manejador del trap 0x83 la rutina
  // que habilita las interrupciones
  leon3_set_trap_handler(0x83,leon3_trap_handler_enable_irqs);

  //Instalar el manejador del trap que 0x83 la rutina
  // que deshabilita las interrupciones
  leon3_set_trap_ handler(0x84,leon3_trap_handler_disable_irqs);


  //Llamada al sistema para deshabilitar las interrupciones
  leon3_sys_call_disable_irqs();

   //COMPLETAR
   //
   //

      //Enmascarar todas las interrupciones


     //Instalar la función device_hw_irq_level_1_handler como
     // manejador de usuario de la interrupción de nivel 1


      //Desenmascarar la interrupción de nivel 1


      //Llamada al sistema para habilitar las interrupciones


     //Fuerza la interrupción


   //FIN COMPLETAR

    return 0;

}
```

Comprobar que la salida que se da por pantalla es la siguiente:

```
System HW IRQ handler
Irq level = 1;
Device HW IRQ user handler
```

4. Repetir la ejecución poniendo un breakpoint en la primera instrucción de la función `leon3_trap_handler_enable_irqs` definida en *leon3_ev_handling_asm.S* y otro en la **llamada** a la función `leon3_sys_call_enable_irqs` **que se encuentra en el main**, y que se ha definido como *wrapper* del trap 3 (vector 0x83). Ejecutar **paso a paso** (Step-into, tecla F5) para comprobar cual es el comportamiento. ¿A qué función salta el programa tras la instrucción en ensamblador **ta** ?

5. Repetir la ejecución enmascarando la interrupción de nivel 1 (usa `leon3_mask_irq`) antes de `leon3_force_irq(1)` y comprobar que no se genera ningún mensaje por pantalla.

6. Repetir comentando la llamada a `leon3_sys_call_enable_irqs()`, comprobando que tampoco se genera ningún mensaje.

7. Ejecutar el siguiente código al final del programa. ¿Qué ocurre? Sabiendo que cuando se produce una división por 0 la rutina de la biblioteca que implementa la división llama al trap 0x82, ¿cómo utilizarías la función `leon3_set_trap_handler` para conseguir que el programa no se cuelgue y en su lugar imprima un mensaje que diga “error, división por cero”?

```
uint8_t i;
uint8_t j;
for(i=10; i>0; i--)
  j=j/(i-9);
```

7. Prac3b. Tarea a realizar.

En esta parte de la práctica se va a modificar el driver de la UART-A de la práctica 2 para poder disparar interrupciones ante la recepción de datos. Estas interrupciones están asociadas al nivel de interrupción externa 2. Además, se va a aprender a configurar la UART-A en modo LOOP-BACK para forzar que los datos que se intenten transmitir, sean redirigidos de nuevo hacia la entrada, de forma que se pueda simular la recepción de datos a través de la UART-A y su gestión mediante interrupciones.

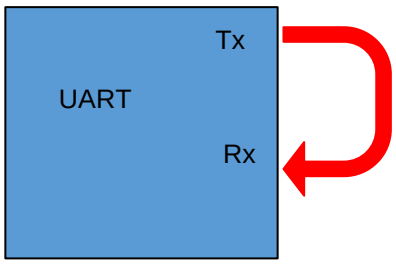

Figura 5. UART-A en modo LOOP-BACK

1. Crear un nuevo proyecto denominado Prac3b, en el que copiaremos todos los archivos del proyecto Prac3, siguiendo la misma estructura de directorios que en el proyecto Prac3 (si tenéis el proyecto configurado en SVN, cuidado con copiar directorios porque tendréis problemas al subir Prac3b)

2. Comentar la línea que define la macro VERBOSE_TRAP_HANDLER del archivo *leon3_ev_handling.h* (Así se evitarán envíos de caracteres por la UART por parte del manejador de interrupciones)

```
#ifndef TRAPS_H_
#define TRAPS_H_

//#define VERBOSE_TRAP_HANDLER
```

3. Añadir al archivo *leon3_uart.h* la declaración de las funciones `leon3_getchar`, `leon3_uart_ctrl_rx_enable`, `leon3_uart_ctrl_rx_irq_enable` and `leon3_uart_ctrl_config_rxtx_loop` tal como aparece a continuación:

```
char leon3_getchar();

void leon3_uart_ctrl_rx_enable();

void leon3_uart_ctrl_rx_irq_enable();

void leon3_uart_ctrl_config_rxtx_loop(uint8_t set_rxtxloop);
```

4. Añadir al archivo *leon3_uart.c* el código de las funciones `leon3_getchar`, `leon3_uart_ctrl_rx_enable`, `leon3_uart_ctrl_rx_irq_enable` y `leon3_uart_ctrl_config_rxtx_loop`, según la siguiente definición:

   - `leon3_getchar` devuelve el valor (convertido a uint8_t) que se encuentra en el registro Data de la UART-A (ver descripción de este registro en la práctica 2)

   - `leon3_uart_ctrl_rx_enable:` pone a 1 el campo `Receiver_enable` del registro de control de la UART-A **sin modificar el resto de campos de ese registro** habilitando la recepción de datos a través de la UART. (Ver la figura siguiente para ubicar la posición del campo).

   - `leon3_uart_ctrl_rx_irq_enable` pone a 1 el campo `Receiver_interrupt_enable` del registro de control de la UART-A **sin modificar el resto de campos de ese registro** habilitando las interrupciones tras la recepción de datos a través de la UART. (Ver la figura siguiente para ubicar la posición del campo).

   - `leon3_uart_ctrl_config_rxtx_loop (uint8_t set_rxtxloop)` función que recibe como parámetro el valor que se quiere fijar en el campo `loop_back` del registro de control de la UART-A (ver la figura 5), de

forma que si `set_rxtxloop` vale 1 se habilita el modo LOOP_BACK de la UART y si vale 0 se deshabilita.

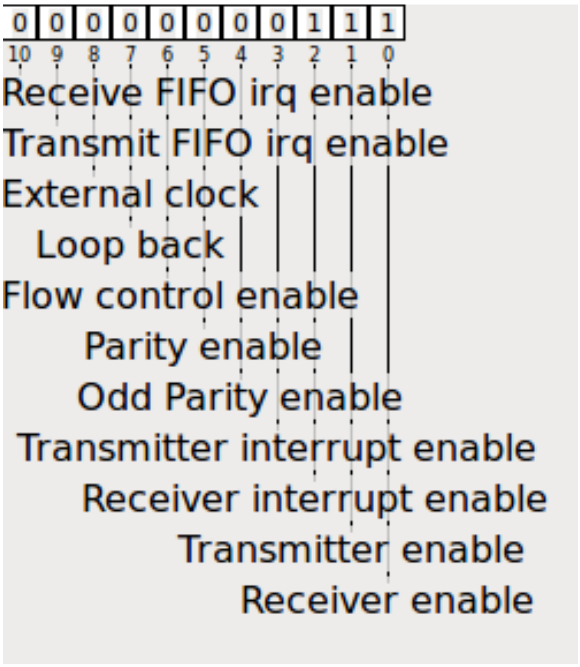

Figura 6. Descripción del registro Control de la UART-A

5. Completar el siguiente programa principal, de forma que para la interrupción externa de nivel 2, asociada a la recepción de un caracter por la UART-A, se instale el manejador `uart_rx_irq_handler`. Además, una vez instalado el manejador, y configurada la UART con las funciones `leon3_uart_ctrl_rx_enable`, `leon3_uart_ctrl_rx_irq_enable` y `leon3_uart_ctrl_config_rxtx_loop`, se deberá proceder a habilitar las interrupciones, y a desenmascarar la interrupción de nivel 2.

```
#include "leon3_uart.h"
#include "leon3_bprint.h"
#include "leon3_hw_irqs.h"
#include "leon3_ev_handling.h"

uint8_t irq_counter=0;
char RxChars[8];

/*Esta función se utilizará como manejador de la interrupción
que genera la recepción de un dato por el puerto serie.
La función almacena en RxChars los primeros 8 carateres que se
reciben, reenviando, además el sucesor por el mismo canal Tx*/

void uart_rx_irq_handler(void){

      char aux;
      aux=leon3_getchar();

      if(irq_counter<7){
            RxChars[irq_counter]=aux;
            aux++;
```

```
            leon3_putchar(aux);
      }
      irq_counter++;
}

int main()
{
      uint8_t i;

  //Instalar como manejador del trap 0x83 la rutina
  // que habilita las interrupciones
  leon3_set_trap_handler(0x83,leon3_trap_handler_enable_irqs);

  //Instalar el manejador del trap que 0x83 la rutina
  // que deshabilita las interrupciones
  leon3_set_trap_ handler(0x84,leon3_trap_handler_disable_irqs);


  //Llamada al sistema para deshabilitar las interrupciones
  leon3_sys_call_disable_irqs();

      //COMPLETAR instalando como manejador de la interrupción de
      //nivel 2 la rutina uart_rx_irq_handler siguiendo el mismo
      //patrón de la prac3a









      //FIN COMPLETAR

   //Habilito loop-back
   leon3_uart_ctrl_config_rxtx_loop(1);

   //Habilito interrupción de recepción por la UART
   leon3_uart_ctrl_rx_irq_enable();

   //Habilito la recepción por la UART
   leon3_uart_ctrl_rx_enable ();


   //COMPLETAR habilitando las interrupciones y
   //desenmascarando la interrupción de nivel 2


   //FIN COMPLETAR
   leon3_putchar('A');

   //Sondeo si los 8 caracteres se han recibido
   while(irq_counter < 7)
            ;

   //Tras recibir los 8 caracteres, configuro la UART
    //sin loop-back
   leon3_uart_ctrl_config_rxtx_loop(0);
```

```
    //Envío de nuevo los 8 caracteres que se recibieron,
    //pero sin loop-back, por lo que aparecerán por pantalla
    for(i=0;i<8;i++)
             leon3_putchar(RxChars[i]);

    leon3_putchar('\n');

    //Espero a que todos los caracteres se hayan enviado.
    while(!leon3_uart_tx_fifo_is_empty())
             ;
    return 0;
}
```

6. Comprobar que una vez hechos todos los cambios indicados en los puntos anteriores, la salida por pantalla muestra la cadena ABCDEFGH tal como aparece a continuación.

```
allocated 2048 KiB ROM memory
icache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
dcache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
gdb interface: using port 1234
Starting GDB server. Use Ctrl-C to stop waiting for connection.
connected
ABCDEFGH
gdb: disconnected
gdb interface: using port 1234
Starting GDB server. Use Ctrl-C to stop waiting for connection.
```

7. ¿Qué línea tendrías que cambiar para que la salida por pantalla fuera 12345678, en vez de ABCDEFGH?.