Proyecto de Ingeniería de Computadores (PEC)

Objetivo del curso

El objetivo de esta asignatura es que el alumno aprenda a desarrollar un prototipo de un computador o un SoC (System on Chip) en un chip programable sobre una placa base para crear un mini-ordenador. Se pondrán en práctica algunos de los conocimientos adquiridos en asignaturas anteriores sobre el diseño de la microarquitectura de un procesador, sobre el diseño e implementación de software de sistema, y sobre el diseño de sistemas digitales.

Fases principales del proyecto.

1) Aprendizaje de las herramientas de desarrollo para los chips programables (FPGA) y práctica del lenguaje de descripción del hardware VHDL.
2) Implementar pequeños componentes o dispositivos en el chip programable de la placa base.
3) Implementar una primera versión simplificada del procesador en una FPGA (sin memoria externa, ni soporte para el sistema operativo o dispositivos externos)
4) Implementar una versión completa del procesador.
5) Programar un sistema de arranque (BIOS) para el Sistema Operativo en el procesador.
6) Evaluar el rendimiento de varias aplicaciones sobre la plataforma que se ha diseñado.

Objetivo de las sesiones

En estas sesiones vamos a implementar físicamente un procesador sencillo en la FPGA. Lo haremos por etapas. En cada etapa cogeremos el trabajo realizado en la anterior y le añadiremos o modificaremos algún componente.

Etapa 5: Entrada y Salida.

En esta etapa daremos soporte para las instrucciones que permiten la comunicación con los dispositivos de entrada y salida. Además, desarrollaremos unos controladores de dispositivo que nos permitirán encender unos LED (un grupo de 8 *leds* rojos y otro grupo de 8 *leds* verdes), usar una agrupación de 4 visores *7-segmentos* para permitir la visualización de un valor numérico, leer el valor de 8 *switchs* y controlar 4 pulsadores que tiene la placa de desarrollo.

Estos dispositivos no funcionarán con interrupciones ya que el procesador aún no las soporta, pero se le añadirá más adelante. Los cambios al sistema que haremos seguirán este orden:

1. Añadiremos soporte para las instrucciones IN y OUT. Estas instrucciones escribirán y leerán de un registro de 16 bits que se encuentra fuera del procesador. Además, haremos que los 8 bits de menor peso de este registro estén conectados a los 8 *leds* de la placa. Tendremos un registro para los 8 *leds* rojos y otro para los 8 *leds* verdes. Por tanto, tras este paso podremos encender y apagar los *leds* de la placa de desarrollo conectados a la FPGA.
2. Con estas modificaciones y sin tocar el procesador, añadiremos 3 dispositivos adicionales al espacio de entrada y salida. Los pulsadores, los *switchs* y el grupo de 4 visores *7-segmentos*.

Soporte básico

Primero desarrollaremos un módulo que será muy parecido a una memoria RAM de *words*. Esta memoria corresponderá al espacio direccionable de entrada y salida, diferente del espacio normal y tendrá un bus de datos de 16 bits (*word*) y un espacio direccionable de 256 posiciones. Es como si dispusiésemos de 256 registros de 16 bits cada uno.

Para cada dispositivo que queramos conectar al procesador deberemos escoger, al menos, una posición de esta memoria (o puerto) donde estará mapeado y describir su comportamiento. Imaginemos que deseamos conectar los 8 *leds* rojos al puerto 6. Este dispositivo podrá ser leído (IN rd, 6) y escrito (OUT 6, rb) como un registro normal. El espacio de

direccionamiento para las instrucciones IN y OUT es compartido (cuando lo estudiasteis en IC era separado). Sin embargo, este módulo no tendrá de momento los efectos colaterales en las lecturas vistos en la asignatura de IC ni implementaremos ningún protocolo de *handshaking*.

Al módulo que contiene todos los registros (la memoria de E/S) y las conexiones le llamaremos **controladores_IO**. Este módulo, como hemos dicho antes, se comporta como una memoria: tiene un bus de direcciones de 8 bits (*addr_io*), un bus de 16 bits de escritura (*wr_io*) y otro de lectura (*rd_io*). También tiene una entrada (*wr_out*) que es el *write enable* y otra entrada (*rd_in*) que indica si estamos leyendo un puerto (que se usaba para los efectos colaterales de la lectura que por ahora no implementaremos).

Es habitual que haya que inicializar algunos de los controladores que hay en el espacio de entrada y salida, así como disponer de una señal de reloj para uso de otros controladores que implementaremos en siguientes sesiones del proyecto. Así que a este módulo también le podremos las entradas *boot* y CLOCK_50 aunque por ahora alguna no se use.

De momento, las salidas del módulo sólo serán las señales de control de los 8 *leds* verdes y los 8 *leds* rojos. Mapearemos los *leds* verdes directamente a los 8 bits de menor peso del puerto 5 y los 8 *leds* rojos a los bits de menor peso del puerto 6.

Al inicializar estos dispositivos, todos los *leds* deberán permanecer apagados.

Así pues, una posible cabecera del diseño en vhld es como sigue:

```
ENTITY controladores_IO IS
  PORT (boot        : IN  STD_LOGIC;
        CLOCK_50    : IN  std_logic;
        addr_io     : IN  std_logic_vector(7 downto 0);
        wr_io       : in  std_logic_vector(15 downto 0);
        rd_io       : out std_logic_vector(15 downto 0);
        wr_out      : in  std_logic;
        rd_in       : in  std_logic;
        led_verdes  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        led_rojos   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END controladores_IO;

ARCHITECTURE Structure OF controladores_IO IS
BEGIN
        .
        .
        .
END Structure;
```

Antes de implementar un diseño en vhdl para este módulo, veamos las modificaciones que debemos hacer en el procesador.

Implementación

Para implementar las instrucciones de entrada y salida: IN y OUT hay que hacer algunas pequeñas modificaciones en el procesador. Básicamente el módulo de control debe saber decodificar correctamente las nuevas instrucciones y generar las señales de control. A parte hay que añadir los buses de datos, con sus multiplexores correspondientes, que lleven los datos desde el Banco de registros al módulo de entrada/salida y viceversa.

Módulo del datapath

Debemos hacer una pequeña modificación en el módulo de ejecución para la ejecución de las instrucciones IN y OUT. Deberemos añadir al *datapath* el bus de entrada de los datos procedentes del módulo de entrada/salida (*rd_io*) y llevarlos hasta la entrada del banco de registros. Para ello necesitaremos añadir un multiplexor y su señal de control. También

debemos añadir el bus de salida (*wr_io*) para enviar los datos a los dispositivos. Según la codificación de las instrucciones, estos valores son directamente los que salen por el puerto *b* del Banco de registros.

Módulo de control

Ahora tenemos que añadir la lógica de control de estas nuevas señales que tenemos y modificar ligeramente la de otras que ya usábamos. El módulo de control deberá generar los valores para las señales nuevas: *addr_io*, *rd_in*, *wr_out* y la señal del nuevo multiplexor del *datapath*.

Para poder decodificar e implementar correctamente las instrucciones es necesario saber cómo se codifican. La siguiente tabla muestra un resumen la codificación de las nuevas instrucciones.

15 14 13 12	11 10 9	8	7 6 5 4 3 2 1 0		
0 1 1 1	Rd	0	n n n n n n n n	Input	IN
	Rb	1		Output	OUT

Podemos ver que el código de operación de las instrucciones de entrada/salida es el 7 y las diferenciamos con el bit 8 del formato de instrucción. En ambos casos, el puerto de entrada/salida está codificado directamente en los 8 bits de menor peso de la instrucción.

Módulo proc

En el módulo **proc** simplemente debemos añadir las nuevas señales de entrada y salida (*addr_io*, *rd_io*, *wr_io*, *rd_in*, *wr_out*) y la señal del multiplexor, y conectarlas adecuadamente al módulo de control y al *datapath*.

Módulo SISA SoC

Este es el módulo de nivel superior de la FPGA (el módulo que es el *System on Chip*) y es el que se conecta con los dispositivos externos como pueden ser los chips de memoria o los *leds*. Ahora, el módulo **sisa** contendrá, aparte del procesador y del controlador de memoria, el nuevo módulo de entrada/salida **controladores_IO** que hemos implementado. Además, tendrá las nuevas señales que van directamente a los *leds* de la placa de desarrollo.

La cabecera en VHDL de la entidad se parecería a la siguiente (en negrita lo se ha añadido):

```
ENTITY sisa IS
  PORT (CLOCK_50    : IN  STD_LOGIC;
        SRAM_ADDR   : out std_logic_vector(19 downto 0);
        SRAM_DQ     : inout std_logic_vector(15 downto 0);
        SRAM_UB_N   : out std_logic;
        SRAM_LB_N   : out std_logic;
        SRAM_CE_N   : out std_logic := '1';
        SRAM_OE_N   : out std_logic := '1';
        SRAM_WE_N   : out std_logic := '1';
        LEDG        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        LEDR        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        SW          : in  std_logic_vector(9 downto 9)); -- para boot!
END sisa;

ARCHITECTURE Structure OF sisa IS
BEGIN
        .
        .
        .
END Structure;
```

Con esto ya tenemos toda la lógica necesaria para que las instrucciones IN y OUT puedan funcionar.

Implementad en vhdl un nuevo diseño con estas modificaciones.

Para probar si funciona, simplemente podemos ejecutar el siguiente código como juego de pruebas.

```
movi r0, 0x55
out  5, r0   ; enciende los led verdes de posiciones pares
in   r1, 5
add  r1, r1, r1
out  6, r1   ; enciende los led rojos de posiciones impares
halt
```

Este programa simplemente enciende los 4 *leds* verdes que ocupan las posiciones pares (LEDG[6], LEDG[4], LEDG[2] y LEDG[0])y los 4 *leds* rojos que ocupan las posiciones impares (LEDR[7], LEDR[5], LEDR[3] y LEDR[1]).

Otros dispositivos

Añadamos el resto de los dispositivos que nos faltan. Para esta etapa, no hay que modificar ni el procesador ni la lógica de este. Sólo hay que modificar el sistema de entrada/salida (controladores_IO) y el módulo *SoC* de más alto nivel (sisa) donde se mapean físicamente las conexiones.

Interruptores y pulsadores

Hasta ahora sólo hemos hecho dispositivos de salida, pero para comunicarnos con el procesador es vital poder desarrollar unos dispositivos de entrada. Los pulsadores (*key*) y los interruptores (*switch*) son seguramente los dispositivos de entrada más sencillos que existen. Los pulsadores y los interruptores no tendrán de momento ningún controlador especial ya que no deben enviar ninguna interrupción al procesador al ser activados. El procesador detectará que han sido pulsados por encuesta, irá comprobando el registro que almacene el estado hasta que este cambie. Por tanto, no debemos desarrollar ninguna lógica para implementarlos. Sólo los conectaremos al bus de lectura de dispositivos y la CPU los podrá leer en cualquier momento.

El valor real de los 4 pulsadores (no importa que cuando se oprime el valor recibido es un 0 y cuando no está pulsado un 1) de que dispone la placa de desarrollo estarán constantemente mapeados en los 4 bits de menor peso del registro del puerto 7. Al ser un registro, actualizaremos su valor cada vez que llegue el flanco ascendente de la señal de reloj. Como es un puerto de entrada que se actualiza constantemente, no tiene sentido permitir las escrituras mediante la instrucción OUT. Aun así, el procesador lo tratará como cualquier otro puerto (permitiendo la ejecución de instrucciones OUT) y deberá ser el controlador de Entrada/Salida el que inhiba esa escritura en el registro.

También añadiremos los 8 primeros interruptores que hay en la placa de desarrollo (SW[7] … SW[0]). Estos interruptores los mapearemos en los 8 bits de menor peso del registro del puerto 8 de forma similar a los pulsadores.

Seguiremos usando el SW[9] como señal de *boot* del sistema y el SW[8] nos queda libre para usarlo como queramos.

Visores 7-segmentos

Añadiremos los 4 visores 7-segmentos de la placa de desarrollo. Este componente electrónico permite representar números de cuatro cifras de forma digital tal y como vimos en una de las tareas de principio de este curso. De esta manera, podemos hacer un circuito que coja un número de 16 bits en binario y lo represente en hexadecimal en estos visores (un visor para cada dígito).

Para ello, tenemos que implementar una especie de *driver* que traduzca el número codificado en hexadecimal en 16 bits a señales de estos *leds* tal y como hicimos en las tareas de principio del curso.

El valor que se mostrará por los visores será en que esté almacenado en el registro del puerto 10 de entrada/salida.

Recordad que según la implementación de la placa, los *leds* que forman cada visor están conectados usando lógica negada, por tanto, el valor lógico 0 encenderá el *led*.

Además, podremos apagar y encender indistintamente cada uno de los visores. Los 4 bits de menor peso del registro mapeado en el puerto 9 de entrada/salida indica si el visor correspondiente debe estar encendido o apagado. Por ejemplo, si el bit 0 del registro del puerto 9 vale 0 el visor HEX0 estará apagado y si vale 1 estará mostrando en carácter hexadecimal correspondiente al valor que representan los 4 bits de menor peso de registro mapeado en el puerto 10.

Al inicializar estos dispositivos, todos los *leds* de los visores deberán permanecer apagados.

Ahora, a la entidad **sisa** hay que añadirle las entradas y salidas que faltan:

```
HEX0        : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
HEX1        : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
HEX2        : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
HEX3        : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
SW          : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
KEY         : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
```

Implementad en vhdl todas las modificaciones para añadir estos nuevos dispositivos de entrada/salida y probad que funcionan correctamente.