Práctica 4

Sistemas embebidos. Implementación en una FPGA de un sistema con el microcontrolador *PicoBlaze*.

Escuela Politécnica Superior - UAM

Objetivo

Esta práctica tiene por objetivo reforzar la comprensión del flujo de diseño introduciendo como novedad la implementación de un sistema con un microprocesador embebido (co-diseño hardware-software). Para ello se pide implementar y probar sobre una placa de desarrollo una aplicación con el microprocesador *PicoBlaze*.

PicoBlaze

PicoBlaze es un microcontrolador de 8 bits diseñado especialmente para ser implementado en una FPGA de las familias Virtex/Spartan-II ó Virtex-II/Spartan-3. El diseño ocupa unas 96 *slices* en Spartan 3, tiene un juego de instrucciones reducido, 16 registros, 256 puertos direccionables, 64 posiciones de memoria *scratch-pad*, interrupción enmascarable, y puede obtener un rendimiento en MIPS igual a la mitad de la frecuencia de reloj utilizada (ej.: 25 MIPS a 50 MHz). La memoria de programa que se puede conectar al *PicoBlaze* es de hasta 1024 instrucciones (está pensado para utilizar exactamente una *Block RAM* de la FPGA).

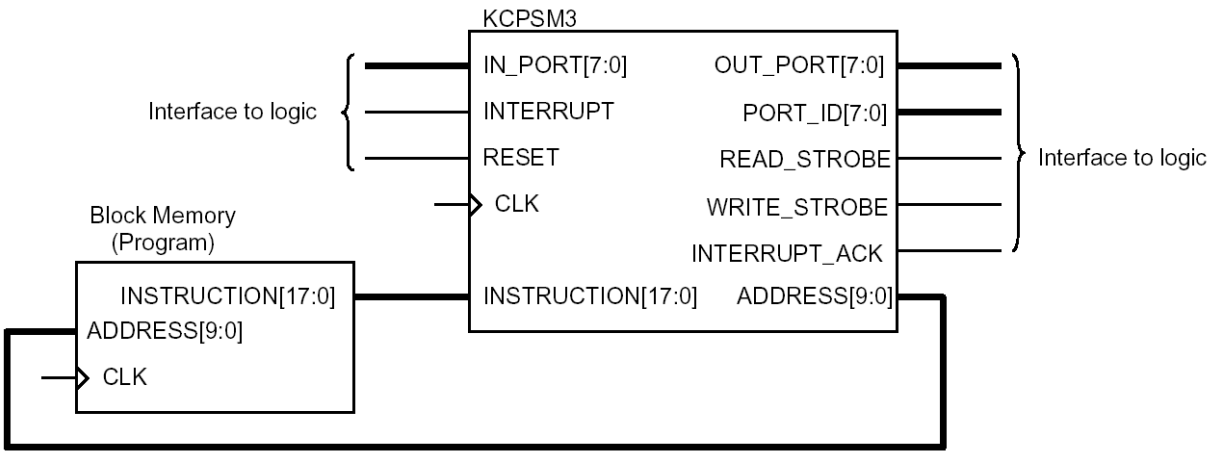

Figura 1.Diagrama de conexión del PicoBlaze con una memoria externa de 1024 palabras

En la página web del laboratorio está disponible un archivo con información sobre este microprocesador. Se recomienda comenzar consultando la presentación "KCPSM3_Manual.pdf".

Práctica Básica. Descripción del diseño.

La práctica consiste en implementar un cronómetro digital utilizando el microcontrolador *PicoBlaze*. A diferencia de la Práctica 2, el movimiento del cronómetro será controlado en buena parte por un programa corriendo en el *PicoBlaze*. Para ello se seguirá el siguiente esquema propuesto:

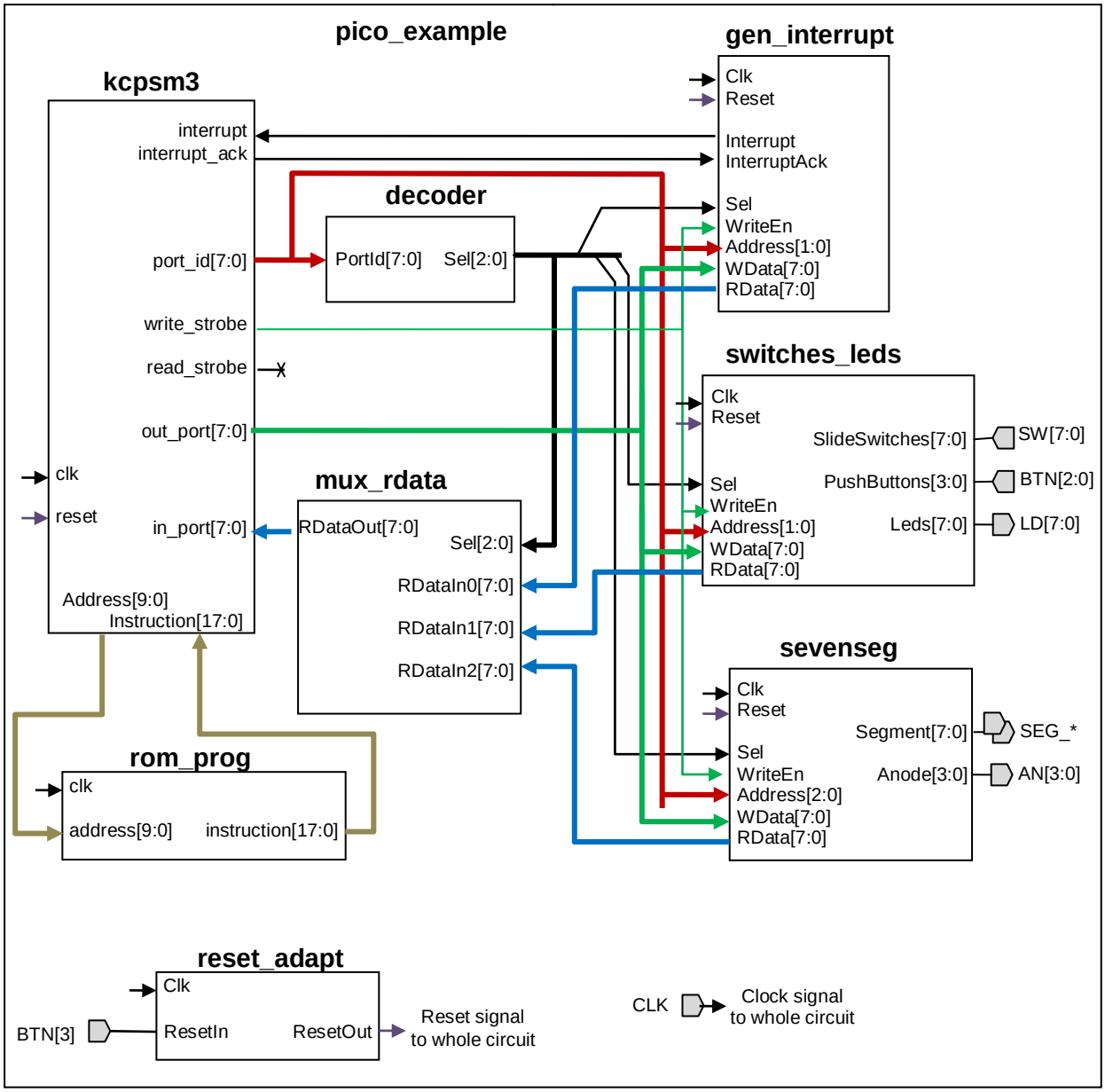

Figura 2. Esquema de bloques del cronómetro utilizando el microcontrolador PicoBlaze

Puertos del circuito:

Entradas

- CLK — Reloj del sistema, generado externamente en la placa.
- SW[7:0] – Puerto conectado a los "slide switches" de la placa S3BOARD (*Spartan-3 Starter Kit Board*). En el diseño básico no tendrán ninguna utilidad real, simplemente la FPGA los volcará sobre los LEDs de la placa mediante el software del *PicoBlaze*.
- BTN[3:0] – Puerto conectado a los "push button switches" (pulsadores) de la placa. Se usará el botón 1 (BTN1) para arrancar y parar el cronometro, el 2 para poner la cuenta a cero y el 3 como reset general del circuito. Un botón pulsado da un nivel alto en la entrada.

Salidas

- LD[7:0] – Puerto conectado a los LEDs de la placa (líneas activas a nivel alto)
- SEG_A, SEG_B,...,SEG_G, SEG_DP: Señales de activación de los segmentos y el punto decimal para los cuatro displays 7-segmentos de la placa. Las señales son activas a nivel bajo.
- AN[3:0] – Bus de 4 bits que permite determinar cuál de los cuatro dígitos 7-segmentos está activo en cada momento. Señales activas a nivel bajo.

Bloques:

- **pico_example** : Éste es el top-level del circuito. Los alumnos deberán desarrollar este bloque completamente, instanciando y conectando los demás bloques.
- **kcpsm3** : Este bloque es el microprocesador *PicoBlaze* ("(K)Constant Coded Programmable State Machine 3"). Consultar la documentación del mismo para más detalles.
- **rom_prog** : Memoria de programa. El código VHDL de este bloque lo generan automáticamente las herramientas de *PicoBlaze* (p.ej. *pBlaze IDE*) a partir del código ensamblador.
- **decoder** : Este bloque definirá el mapa de memoria de los distintos periféricos que va a utilizar el micro. En función del valor de la salida del micro *port_id* activará una sola señal de selección (un bit de su puerto de salida *Sel[2:0]*).
  El bloque deberá ser desarrollado enteramente por los alumnos, implementando el siguiente mapa de memoria:

	DIRECCIÓN BASE (en hexadecimal), según el número de Grupo de Prácticas:							
PERIFÉRICO	**Gr. 1**	**Gr. 2**	**Gr. 3**	**Gr. 4**	**Gr. 5**	**Gr. 6**	**Gr. 7**	**Gr. 8**
gen_interrupt	**00**	**20**	**40**	**60**	**80**	**A0**	**C0**	**E0**
switches_leds	**08**	**28**	**48**	**68**	**88**	**A8**	**C8**	**E8**
sevenseg	**10**	**30**	**50**	**70**	**90**	**B0**	**D0**	**F0**

Si el número de grupo de prácticas es mayor que 8 se rotará a lo largo de la tabla, es decir, el grupo 9 utilizará las direcciones del grupo 1, etc (la fórmula sería *((num_grupo-1) módulo 8)+1*).

A cada periférico se le asignará un espacio de 8 direcciones. Así por ejemplo, la señal de selección para el bloque *switches_leds* del grupo 3 de prácticas (dirección base = 48h) se activará si *port_id* = 01001xxx , donde 'x' significa que cualquier valor es válido.

- **mux_rdata** : Multiplexor para la lectura de datos de puertos de entrada por parte del micro. En función de la señal de selección provista por el *decoder*, deja pasar uno entre varios buses. Este bloque se entrega ya listo para ser integrado.
- **switches_leds** : Periférico que facilita el acceso del programa a los *slide switches*, los pulsadores y los LEDs. Los dos primeros se corresponden con dos puertos de entrada en diferentes direcciones y el último con un puerto de entrada/salida en otra dirección (el programa puede tanto escribir los LEDs como leer qué ha escrito previamente). Nótese que en este diseño el pulsador 3 se usará como reset general, por lo que se conectará un cero a la correspondiente entrada de este bloque en lugar del pulsador. Este bloque se entrega ya listo para ser integrado.

	Bit							
Dirección relativa	7	6	5	4	3	2	1	0
0	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
1	0	0	0	0	0	BTN2	BTN1	BTN0
2	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0

- **gen_interrupt** : Periférico que genera una interrupción cada cierto número de ciclos de reloj. Este número de ciclos es un valor de 32 bits que puede ser configurado por el microprocesador a través de cuatro puertos de 8 bits. Configurando el número adecuado se podrá obtener una interrupción cada centésima de segundo partiendo del reloj del sistema, de 50 MHz. El periférico también permite leer el valor configurado. La señal de interrupción se borra automáticamente al activar el *PicoBlaze* su señal de *interrupt acknowledgement*. Consúltese la documentación de *PicoBlaze* para adquirir una mejor comprensión del mecanismo de interrupciones. Este bloque se entrega ya listo para ser integrado.
- **sevenseg** : Periférico que registra los valores a visualizar en los displays de 7 segmentos y realiza la activación rotatoria de los cuatro dígitos. Los valores se escriben ya en formato 7-segmentos (la conversión de "hexadecimal" a 7 segmentos la realiza el microprocesador). Hay 4 registros, 1 para cada dígito de 7 segmentos, en direcciones relativas 0,..,3 para las posiciones de los dígitos de derecha a izquierda (esto es, en la dirección relativa 0 se escribirá el valor para visualizar las centésimas de segundo). Cada registro de usuario es accesible tanto en escritura como en lectura y tiene el siguiente formato:

Bit	7	6	5	4	3	2	1	0
Segmento	dp	g	f	e	d	c	b	a

Adicionalmente, el bloque tendrá otros 4 registros, en las direcciones relativas 4 a 7, que servirán para poder definir la velocidad de rotación en la estimulación de los diferentes dígitos 7-segmentos. Estos 4 registros configurarán un valor de

32 bits que será utilizado para definir una cuenta de ciclos de idéntica manera a como lo hace el bloque *gen_interrupt*.

El bloque deberá ser desarrollado enteramente por los alumnos (nótese que puede usarse por ejemplo el bloque *gen_interrupt* como referencia de la implementación del registro de configuración de la cuenta).

Nota 1: Ejemplo de direccionamiento: el registro con el código 7-segmentos para las décimas de segundo, para los alumnos del grupo 3, tendrá la dirección absoluta 51h (50h + 1), según se muestra a continuación:

	port_id							
Bit	7	6	5	4	3	2	1	0
Valor	0	1	0	1	0	0	0	1
Campo	Selección (dir. base) Decodificación en el bloque *decoder*					Offset (dir. relativa) Decodificación en el propio bloque *sevenseg*		

Nota 2: En un diseño algo más realista este bloque probablemente incluiría la decodificación de "hexadecimal" a 7 segmentos para facilitar la tarea al software, pero aquí esta parte la pasaremos al programa corriendo en el *PicoBlaze*, para simplificar el diseño de este bloque y hacer un poco más completo el ejemplo de programa.

- **reset_adapt** : Bloque de adaptación de la señal externa de reset. Este bloque se encarga de sincronizar adecuadamente esta señal con el reloj del sistema. Se entrega ya listo para ser integrado.

Programa:

El programa que correrá el *PicoBlaze* constará en primer lugar de un bucle principal que se ejecutará de forma infinita. En cada iteración del bucle se escribirá en los puertos de I/O de *sevenseg* los valores numéricos adecuados para representar los contadores del software (centésimas de segundo, décimas, unidades de segundo y decenas), y se leerá de los puertos de *switches_leds* si las señales de puesta a cero de cuenta o de arranque/parada están activas. Si la señal de arranque/parada está activa se cambiará de estado una variable booleana que permitirá o no avanzar la cuenta. Si la señal de puesta a cero está activa se pondrán a cero los cuatro contadores internos y también se detendrá la cuenta.

Mediante la adecuada configuración, por parte del programa, del bloque de generación de interrupciones, cada 1/100 s se producirá una interrupción en el microcontrolador que llamará a una rutina de interrupción que actualizará los contadores internos. Esta

actualización se hará siempre y cuando la variable booleana con el estado del cronómetro indique que éste está contando.

Con el fin de hacer un cierto *debouncing* de las señales que vienen de los pulsadores, el programa se asegurará de que una vez detectado un pulsador no se le vuelva a hacer caso en un cierto tiempo, para lo cual tras procesar un nivel activo en un pulsador se esperará a la aparición de varias interrupciones con el pulsador a nivel inactivo antes de volver a tomarlo en consideración.

Antes de comenzar esta operativa continuada, el programa deberá configurar el bloque de generación de interrupciones de modo que genere una interrupción cada centésima de segundo, y el bloque de control de los displays de forma que se produzca un refresco de aproximadamente entre 100 y 500 veces por segundo, velocidad adecuada para ver encendidos todos los displays a la vez.

La configuración inicial comentada anteriormente puede ser modificada en el programa de forma que todo se ejecute mucho más rápidamente (ej.: una interrupción cada 200 ciclos de reloj y cambio de un dígito del display 7-segmentos a otro cada 4 ciclos). Esto permitirá realizar simulaciones con una duración más adecuada.

El programa se entrega pero los alumnos deberán hacer dos tipos de modificaciones:

- ⇨ Corregir algunos errores funcionales simples presentes en el código entregado.
- ⇨ Incluir las instrucciones necesarias para el volcado de los "slide switches" sobre los LEDs.

Método a seguir

Estructura de directorios:

Se entrega una base de datos del diseño con la siguiente estructura, que habrá de mantenerse:

rtl/ - Código fuente VHDL para el diseño, con excepción del fichero VHDL describiendo la ROM de programa.

sim/ - Directorio de simulación, contendrá el/los ficheros de test-bench y ficheros relacionados con *ModelSim*.

soft/ - Directorio de software de *PicoBlaze*, contendrá el código fuente del programa utilizado (*.psm*) y el correspondiente fichero VHDL. También contendrá el ejecutable *pBlazeIDE.exe* y la plantilla para la generación del VHDL (*ROM_form.vhd*).

ise/ - Directorio para el proyecto ISE. Inicialmente contendrá un fichero UCF incompleto.

Flujo de trabajo:

La primera tarea será completar el diseño y simularlo comprobando su corrección. Para ello, se recomienda utilizar exclusivamente la herramienta *ModelSim* y el ensamblador de *PicoBlaze*, y opcionalmente un editor de texto diferente si así se desea. Es decir, hasta que no se compruebe al menos mínimamente el funcionamiento por simulación, no debería abrirse el entorno *ISE* (salvo si se desea utilizar su editor de texto). En ModelSim puede crearse si se desea un nuevo proyecto, añadiendo (sin copiar) los ficheros fuente VHDL. Sin embargo, utilizar un "proyecto ModelSim" no es estrictamente necesario cuando se trabaja con scripts como vamos a hacer en esta práctica. En cualquier caso, para todo lo relativo a la simulación se deberá trabajar siempre bajo el directorio *sim/*.

[Nota: si se desea trabajar en casa consúltense las notas al respecto en la página web del laboratorio (en esta práctica, al contrario que en las anteriores, el simulador necesita utilizar las librerías de primitivas de Xilinx, que deben estar correctamente instaladas y referenciadas)].

Para realizar las sucesivas iteraciones de simulación con *ModelSim*, se hará uso de un fichero *.do*, que permite ejecutar en forma de "script" diversos comandos: de compilación, simulación, selección de ondas a mostrar, etc. Se entrega un fichero *runsim.do* incompleto, que deberá completarse al menos con los comandos necesarios para compilar los ficheros creados por los alumnos. Como parte de este fichero de comandos para el simulador, se llama a otro script, *wave.do*, con la configuración de visualización de ondas. Este fichero puede generarse desde el propio visor de ondas (tras añadir las que queramos); para ello usar el comando de guardado del menú *File* mientras se tiene seleccionada la ventana de ondas. Los ficheros *.do* se ejecutan desde la consola de *ModelSim* con el comando *do*, esto es, por ejemplo: "*do runsim.do*".

Se entrega en *sim/* un testbench muy simple (*pico_example_tb.vhd*), que no comprueba nada, simplemente inicia el funcionamiento del cronómetro.

Nótese que es posible mostrar en la ventana de ondas del simulador los registros del microprocesador, la instrucción en curso, etc. Para ello, en la solapa *sim* (donde se muestra la jerarquía del diseño) buscar bajo la instancia del procesador el proceso *simulation* (hacia el final de los objetos que cuelgan del procesador, con un icono en forma de círculo) y seleccionarlo. A continuación, usar el menú *View* | *Locals*. Aparecerá una nueva ventana donde podremos seleccionar estas informaciones y mandarlas al visor de ondas.

Para ensamblar el programa y convertirlo a un VHDL de la ROM, cópiese el ejecutable *pBlazIDE.exe* en el directorio *soft/* y ejecútese desde este directorio. (El entorno *pBlaze IDE* permite además simular el comportamiento del micro, haciendo uso de directivas especiales en el ensamblador para crear controles en el interfaz gráfico dedicados a los diferentes puertos de entrada / salida). Importante: configurar este programa seleccionando en el menú *"Settings*" la opción "*PicoBlaze 3*".

Estúdiese el programa ensamblador, añadiendo y/o corrigiendo las líneas que se estime necesario.

Una vez satisfecha la simulación funcional, se abrirá el entorno ISE, creando un proyecto en el directorio *ise/*:

- Usar *New Project*, nombrar el proyecto "**pico_example**" y tras usar el botón "…" de *Location* borrar *pico_example* del final del path para que éste termine en ...*ise*).
- Comprobar que se selecciona como "top-level source type" *HDL*, que la FPGA seleccionada es la adecuada (Spartan 3 XC3S200 FT256 -4) y que las herramientas seleccionadas son las correctas.
- Añadir al proyecto todos los ficheros *.vhd* del directorio *rtl/*, el fichero *rom_prog.vhd* del directorio *soft/*, y el fichero *pico_example.ucf* del directorio *ise/* (utilizar siempre *Add Source*, nunca *Add Copy of Source*).

El fichero UCF entregado está incompleto. Antes de conectar la placa al ordenador deberá completarse, repasarse su corrección y posteriormente comprobarse que el reporte de *pads* se corresponde con las constraints de localización especificadas.

Proceder a la síntesis e implementación del circuito tal como se ha aprendido en prácticas anteriores.

Una vez completado el proceso de implementación, comprobar el resultado en *timing* y área y **responder a las siguientes preguntas** (anotar en papel o en un fichero las respuestas).

- ⇨ ¿De dónde a dónde va el peor *path*?
- ⇨ ¿Cuánto tarda?
- ⇨ ¿Cuál es el periodo mínimo de reloj? ¿Y la frecuencia máxima?
- ⇨ ¿Cumple el circuito las restricciones de tiempo especificadas?
- ⇨ ¿Cuál es la ocupación de la FPGA?

Comprobar el correcto funcionamiento en la placa y si es necesario (porque no se haya hecho antes) corregir los errores en el software a partir de lo observado en la placa.

- ⇨ **Hacer una lista de los cambios hechos en el software**.

Lanzar una simulación y **observar lo siguiente**:

- ⇨ La ejecución de una interrupción del microprocesador.
- ⇨ ¿Cuánto tarda en ejecutarse cada instrucción?

Pensar en lo siguiente:

- ⇨ ¿Cuántos "mapas de direcciones" diferentes tiene este micro? ¿Cómo es el mapa de direcciones de puertos de I/O (qué hay en qué dirección)?
- ⇨ ¿Es mucho el tiempo que pasa el micro atendiendo la interrupción respecto al tiempo que dedica a otras tareas (bucle principal)?

Mejoras opcionales

Si se desea se podrán realizar mejoras sobre esta práctica, las cuales serán tenidas en consideración. Por ejemplo se podrá entregar un diseño **original** distinto del cronómetro básico. En este caso la calificación dependerá de la complejidad del diseño elegido, de la calidad de su implementación y verificación y del éxito en su comprobación en la placa S3BOARD. También puede añadirse funcionalidad al cronómetro o completar el testbench entregado de manera que compruebe automáticamente la corrección del diseño, de una manera cuanto más completa mejor. Si se realiza cualquier mejora deberá explicarse con claridad al profesor en la entrega de la práctica.

Entrega

El **correcto funcionamiento sobre la placa S3BOARD deberá mostrarse al profesor** en clase como muy tarde el día publicado en la web del laboratorio.

Por otra parte, **deberá entregarse por web un archivo zip con el diseño realizado**, **incluido el software**, antes de la fecha límite especificada en la web del laboratorio. Entregar el directorio "P4_pico_example" completo, eliminando tan sólo los ficheros de ondas de simulación, que son los que suelen ocupar más espacio ("wlf*", "*.wlf"…), y que estarán localizados en el subdirectorio "sim/" si se ha seguido el flujo de trabajo con *ModelSim* indicado en este enunciado.

Importante: Todos los ficheros fuente que se entreguen (.vhd, .ucf, etc.), que hayan sido creados o modificados por los alumnos, deberán llevar una **cabecera** adecuada, **incluyendo siempre el nombre de los autores**. Además todo el código fuente deberá llevar comentarios apropiados, tanto en contenido como en cantidad.

El archivo zip deberá tener la siguiente nomenclatura para los alumnos de DIE:

```
<{DIE_L | DIE_X | DIE_J}><numero_pareja_2digitos>_P4.zip
```

y la siguiente para los alumnos de DCSE:

```
<{DCSE_L | DCSE_XA | DCSE_XB}><numero_pareja_2digitos>_P4.zip
```

(Ejs.: `DCSE_XB01_P4.zip` para pareja 1 de DCSE-b miércoles, `DIE_J02_P4.zip` para pareja 2 de DIE jueves)

No será necesario presentar ninguna memoria.

Se recuerda que los alumnos deberán comprender el diseño completo y familiarizarse con él.