Práctica 2

Completar el diseño de un cronómetro

Objetivos

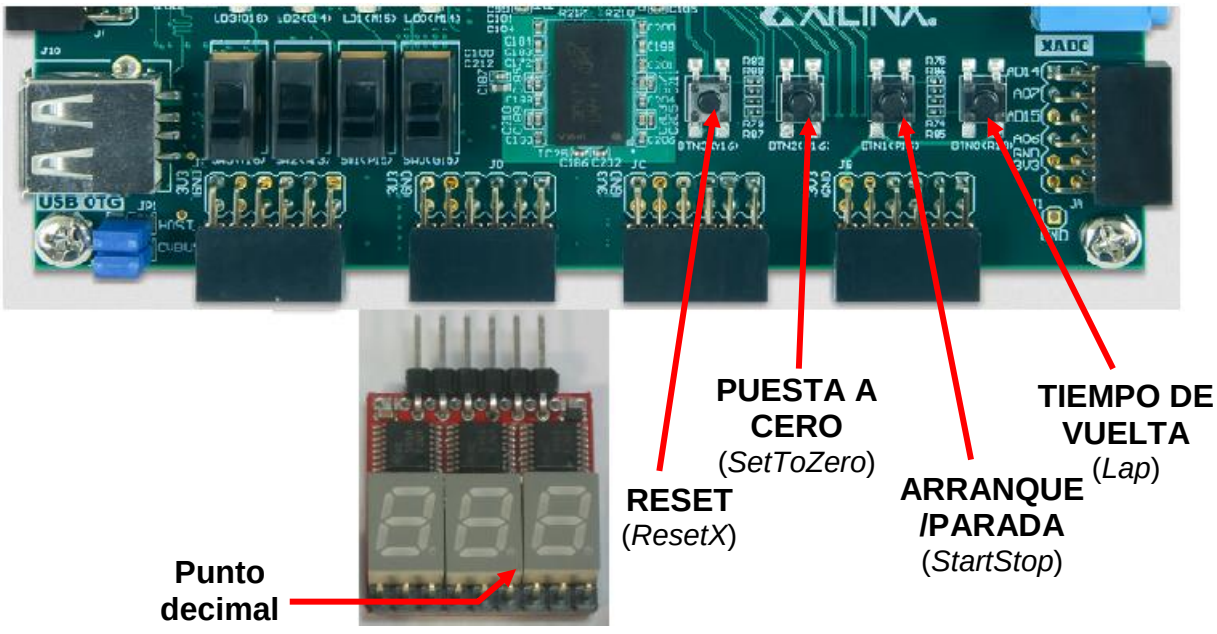
- Realizar el flujo de diseño con un ejemplo más complejo que el de la Práctica 1.
- Practicar la codificación en lenguaje VHDL, desde estructuras simples hasta más avanzadas como funciones, *generics* y *packages*.
- Ver un mecanismo para acelerar las primeras simulaciones.

NOTA IMPORTANTE: En esta práctica se trabaja sobre un diseño incompleto, entregándose a los alumnos porciones de código ya escritas. **Los alumnos deberán comprender <u>todo</u> el código**. A la hora de evaluar la práctica se podrán hacer preguntas sobre cualquier parte del diseño.

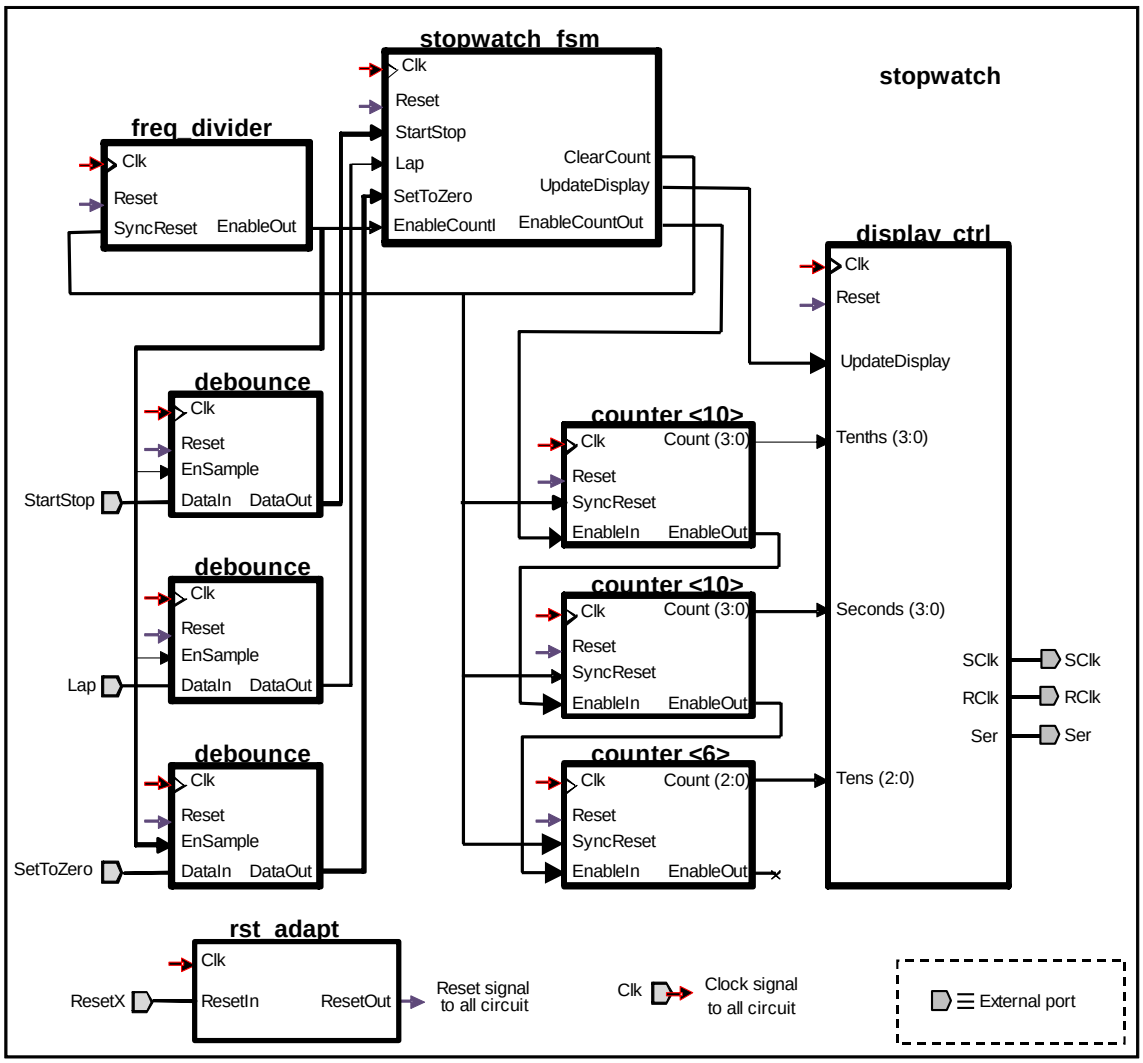
Circuito utilizado

En este ejercicio implementaremos sobre la placa *Zybo* un cronómetro de tres dígitos, con segundos (0-59) y décimas. El cronómetro se visualizará sobre los tres displays 7-segmentos de la placa de expansión tipo *pmod* OHO-DY1. Usaremos los pulsadores BTN3 a BTN0 para lo siguiente:

- BTN3: Reset asíncrono general del circuito.
- BTN2: Puesta a cero del cronómetro.
- BTN1: Parada y rearranque (Start/Stop) de la cuenta.
- BTN0: Función “Lap” para mostrar el tiempo de una vuelta sin que pare el cronómetro, que vuelve a mostrar el tiempo actual al pulsar el botón una segunda vez.



La siguiente figura muestra un diagrama de bloques del circuito:



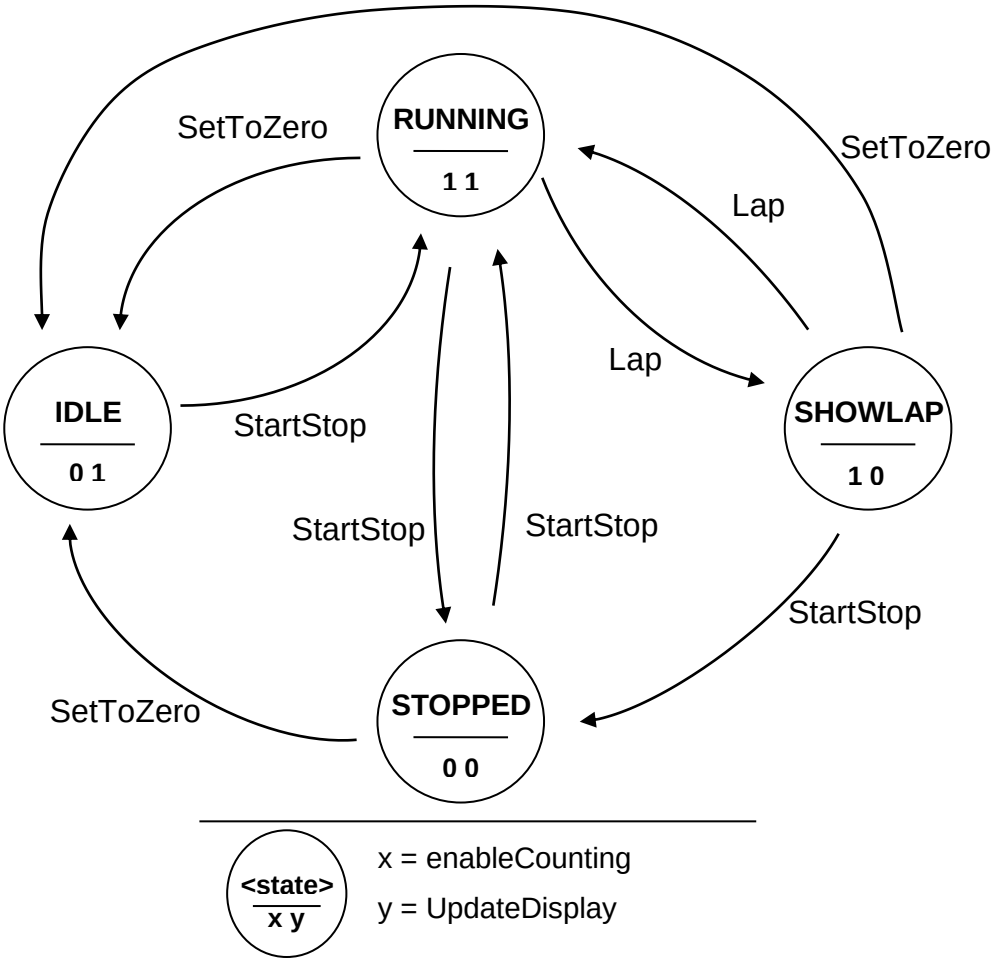
Un bloque de división de frecuencia (*freq_divider*) se encarga de dividir por 12.500.000 el reloj de 125 MHz de la placa para dar un pulso de un ciclo cada décima de segundo. A partir de esta base de tiempos, tres contadores en cascada (*counter*) proveen la cifra de décimas de segundo (*tenths*), unidades de segundo (*seconds*) y decenas de segundo (*tens*). Los contadores son instancias de un contador genérico que puede ser parametrizado para contar entre 0 y (N-1).

Unos bloques de anti-rebote (*debounce*) se encargan de adaptar la señal de los pulsadores de control. Otro bloque (*rst_adapt*) sincroniza el reset externo que viene del otro pulsador.

Una máquina de estados (*stopwatch_fsm*) recibe las señales de los pulsadores de control y permite que se muevan o no los contadores (dejando pasar o no los pulsos generados por *freq_divider*), seleccionando además que se congele o no la cifra visualizada, esto último para la funcionalidad “Lap”.

Finalmente, un bloque de control de los displays (*display_ctrl*) se encarga de convertir las cifras de cada dígito a unas salidas de control del display de tres dígitos 7-segmentos OHO-DY1. Adicionalmente este bloque tiene un registro a su entrada para permitir, cuando no se habilita su carga, “congelar” la visualización mientras el cronómetro corre. Esto último será lo que suceda al pulsar el botón “Lap”.

El diagrama de la máquina de estados es tal como se muestra en la siguiente figura:



Las señales de entrada de la FSM mostradas en las transiciones de este diagrama en realidad no son exactamente las del nombre mostrado, sino que han de corresponderse con señales internas a la máquina, la cuales sean '1' cuando las correspondientes entradas del bloque *stopwatch_fsm* (conectadas a las señales de botón filtradas anti-rebote) pasan de '0' a '1'. Por ejemplo, el *StartStop* mostrado en el diagrama será una señal interna que sea '1' cuando la entrada *StartStop* del bloque *stopwatch_fsm* pase de '0' a '1'.

Por otra parte, la señal de salida de la FSM *enableCounting* también es interna al bloque, y se mezclará mediante un *and* con la señal *EnableCountIn* para generar *EnableCountOut*. Además, no se muestra en el diagrama la salida *ClearCount*, que se encargará de poner a 0 todos los contadores, y que será '1' siempre que llegue *SetToZero*, siendo por tanto independiente del estado y resultando equivalente a utilizar la señal de detección de cambio de '0' a '1' de ese pulsador.

El estado por defecto de la FSM debe ser "IDLE".

Guía para la realización del diseño

Se parte de la siguiente estructura de directorios y ficheros, que se entrega a los alumnos en un fichero zip y que habrá que situar bajo el directorio de trabajo:

```
P2_stopwatch /
                    rtl/
                          counter.vhd  -> incompleto
                          debounce.vhd -> completo
                          display_ctrl.vhd    -> incompleto (*)
                          freq_divider.vhd    -> no se entrega a los alumnos
                          rst_adapt.vhd       -> completo
                          stopwatch.vhd-> no se entrega a los alumnos
                          stopwatch_fsm.vhd   -> incompleto
                          stopwatch_pkg.vhd   -> completo

                    netlist/
                          display_ctrl.dcp -> completo (*)
                          display_ctrl.edn -> completo, no se usa (*)
                          display_ctrl.vhd -> completo (*)

                    sim/
                          stopwatch_tb.vhd    -> no se entrega a los alumnos
                          runsim.do           -> completo (script ModelSim)
                          runsim_netlist.do   -> completo (script ModelSim)

                    vivado/
                          stopwatch.xdc-> incompleto (constraints)
```

Esta estructura de directorios **deberá mantenerse**.
Para terminar el diseño será necesario **completar** los ficheros fuente **incompletos** y **crear los no entregados**.
(*) En el caso del bloque *display_ctrl*, este se entrega incompleto, pero también se entrega completo en formato de "netlist" (de forma efectiva es como si fuese una caja negra). Puede utilizarse en este formato para tener el diseño completo del cronómetro, siendo de entrega opcional (con mayor puntuación) el desarrollar este bloque en VHDL (directorio *rtl/*).

1- Crear un nuevo proyecto y el fichero del *top-level* del RTL

- Crear con Vivado un nuevo proyecto de nombre ***stopwatch***, creando su propia carpeta, **bajo la carpeta *P2_stopwatch/vivado***. Como en la Práctica 1, será un *RTL Project* para la FPGA XC7Z010CLG400-1. Durante la creación pueden añadirse ya los ficheros presentes en el directorio *rtl/*, **salvo *display_ctrl.vhd*, que no deberá añadirse al proyecto**. Por otra parte **NO se deberá utilizar la opción *Copy sources into Project***. De esta manera mantendremos nuestro código en *rtl/*.
- Bien con el editor de Vivado o con el editor de texto que uno prefiera, crear el fichero VHDL del "top-level" del diseño (*stopwatch.vhd*). Desde Vivado se puede hacer con clic derecho en el panel de fuentes y *Add Sources > Add or create design sources > Create File*. Si se crea externamente luego también se puede añadir al proyecto con *Add Sources*. **IMPORTANTE**: los ficheros de código fuente RTL utilizados por el proyecto Vivado **deberán situarse siempre en el directorio *rtl/***.
- En este fichero *stopwatch.vhd*, crear la entidad con sus puertos, a partir del esquema del circuito de más arriba (donde los puertos externos vienen identificados por un pentágono gris). Crear también inicialmente una arquitectura en blanco.

2- Crear un primer sub-bloque desde cero: el divisor de frecuencia

En este apartado haremos un primer ejercicio recordatorio de VHDL:

- Describiremos un contador, como ejemplo típico de proceso secuencial.
- Veremos en un mismo código la codificación del reset asíncrono y síncrono.
- Asignaremos varias señales en un mismo proceso.
- Al terminar el diseño de este sub-bloque, pasaremos a situarlo en el bloque "top-level" para recordar cómo se declaraba e instanciaba un componente.
- Lo que crearemos será un ejemplo de generación de "base de tiempos" para el funcionamiento de otros bloques.

- Bien con el editor de Vivado o con el editor de texto que uno prefiera, procederemos a crear el fichero VHDL del divisor de frecuencia (*freq_divider.vhd*). Desde Vivado se puede hacer con clic derecho en el panel de fuentes y *Add Sources > Add or create design sources > Create File*. Si se crea externamente luego también se puede añadir al proyecto con *Add Sources*. **IMPORTANTE**: **los ficheros de código fuente RTL deberán situarse siempre en el directorio *rtl/***.
- Para editar este fichero y los siguientes: si no se recuerda bien alguna estructura sintáctica de VHDL se puede consultar la documentación en Moodle y/o hacer uso de *Tools > Language Templates*, pero conviene memorizar las estructuras básicas del lenguaje.
- Para implementar el divisor de frecuencia crearemos un proceso que haga una cuenta de 0 a 12.499.999 (este valor deberá definirse como una *constant* VHDL de tipo *integer*). Para soportar la cuenta usaremos una *signal* "count" de tipo *std_logic_vector* que se deberá poner a 0: a) cuando llegue el reset asíncrono (activo a nivel alto); b) cuando al llegar el flanco de reloj veamos activo (a 1) el reset síncrono del bloque (*SyncReset*, señal que viene de la máquina de estados); y c) cuando al llegar el flanco de reloj la cuenta actual sea igual a la constante definida (12.499.999). Si no se cumple ninguna de estas condiciones, la cuenta deberá avanzar con la llegada del flanco de reloj. Para implementar este proceso partir de la descripción de un flip-flop con reset asíncrono, respetando su estructura y rellenando apropiadamente la parte del "if" en la que se ha detectado el reset asíncrono y la parte en que se ha detectado la llegada de un flanco activo del reloj.

  A la hora de comparar nuestra cuenta con la constante, haremos uso de la función *conv_integer* del package *std_logic_unsigned*.
- A continuación, añadiremos en el mismo proceso la generación de la señal de salida *EnableOut*, la cual estará un ciclo a '1' cada 12.500.000. Es decir, generaremos esta señal de forma registrada, como un flip-flop adicional. No se debe olvidar que este flip-flop adicional también debe resetearse adecuadamente.
- Finalmente, añadir en el fichero *stopwatch.vhd* la declaración de este componente y su instanciación. En la instanciación conectar el puerto *Clk* al del "top-level" y crear señales internas para los demás puertos (por ejemplo: *reset*, *clearCount*, *enFromFreqDiv* u otros nombres con sentido). Declarar estas *signal*s en la parte declarativa de la arquitectura.

3- Terminar e instanciar el componente *counter*:

En este apartado volveremos a practicar creando un contador, pero esta vez:

- Haremos uso de una señal de "clock enable".
- Crearemos varias "asignaciones concurrentes" (formato VHDL abreviado de procesos) para señales combinacionales.
- Utilizaremos una señal combinacional interna generada con una de las asignaciones concurrentes para indicar al proceso secuencial contador cuándo ha llegado al final de cuenta.
- Veremos un ejemplo de uso de los *generic* VHDL.
- Veremos un ejemplo de uso de los *packages* VHDL.

- Si no se ha hecho ya, añadir al proyecto los ficheros "counter.vhd" y "stopwatch_pkg.vhd" (del directorio "rtl/"). Estudiar el contenido de estos ficheros. Se trata de un contador cuyo valor máximo de cuenta es parametrizable y de un *package* de utilidad. Observar el uso en el contador del *generic*, del *package* "stopwatch_pkg" y de la función "log2_ceil" definida en este package.
- Completar el fichero "counter.vhd". Tener en cuenta que el contador interno debe resetearse asíncronamente con *Reset* y síncronamente tanto con el final de cuenta como con la entrada *SyncReset* (señal que viene de la máquina de estados). Además, habrá que tener en cuenta que el contador sólo debe avanzar (o bien hacer "wrap-around", esto es, volver a 0 desde su valor máximo) cuando lo permita la señal de "enable" de entrada.
- Añadir la declaración del componente de este contador y sus tres instancias a "stopwatch.vhd".

  En cada instancia utilizar un *generic map* adecuado, para que cada contador cuente hasta donde debe.
- Conectar los puertos de las tres instancias de *counter*, añadiendo tanto las señales que van entre ellos como las que posteriormente conectaremos a las instancias de otros componentes. Declarar estas *signal*s que vamos creando, en la parte declarativa de la arquitectura. En el último contador su "enable" de salida queda al aire, por lo que usaremos, en vez de una *signal*, la palabra especial ***open***.
- Añadir en *stopwatch.vhd* la misma sentencia para uso del package *stopwatch_pkg* que se puede encontrar en *counter.vhd* (necesaria para utilizar la función *log2_ceil*).

4- Comprensión de los ficheros que se entregan completos:

En este apartado veremos dos ejemplos de mecanismos de sincronización:

- Cadena de anti-metaestabilidad.
- Sincronización de reset.

- Añadir al proyecto y examinar los ficheros "rst_adapt.vhd" y "debounce.vhd". Comprender su funcionamiento, preguntando al profesor en caso de dudas.

- Añadir sus declaraciones e instanciaciones a "stopwatch.vhd". Para el caso de "rst_adapt" puede obviarse el hacer un *generic map,* utilizando el valor por defecto del *generic*. Tener en cuenta que, como se muestra más arriba en el diagrama del diseño, el reset externo entrante es *ResetX* y la salida de *rst_adapt* será la que vaya, con el nombre que le hayamos dado, a todos los demás bloques.

5- Completar la máquina de estados:

En este apartado:

- Describiremos una máquina de estados (FSM), siguiendo el estilo de codificación que divide las FSM en un proceso combinacional y otro secuencial.
- Ejercitaremos la correcta escritura de un proceso combinacional.
- Veremos un ejemplo de detección de flancos en señales sin usar éstas como reloj.

- Si no se ha hecho ya, añadir al proyecto el fichero "stopwatch_fsm.vhd". Estudiarlo y completarlo, considerando la descripción de la máquina de estados dada más arriba. A la hora de describir la parte combinacional se puede tener en cuenta que *SetToZero* produce el mismo efecto en todos los estados, para simplificar el código.
- No olvidar dedicar el tiempo necesario para comprender la parte del código que ya viene escrita.
- Añadir la declaración e instanciación de este componente a "stopwatch.vhd".

6- Añadir el bloque de control de los *displays* 7-segmentos, como una *netlist*

En este apartado:

- Añadiremos a nuestro diseño un bloque del cuál no tenemos su código fuente (como una *IP -Intelectual Property-* que solo tuviéramos disponible en formato de *netlist*).

Los tres ficheros presentes en el directorio *netlist/* representan la misma netlist (lista de células + interconexión), en tres formatos diferentes: VHDL (que podremos usar como modelo de simulación del bloque), EDIF (formato de netlist estándar en la industria, incluido aquí simplemente como referencia) y DCP (formato propietario de lo que en Vivado se llama un "design checkpoint"). Echar un vistazo rápido a los dos primeros con un editor de texto. El tercero, fichero DCP, es realmente un archivo comprimido con varios ficheros en su interior.

Utilizar *Add Sources* para añadir al proyecto el fichero *display_ctrl.**dcp*** (ojo, no .vhd) del directorio *netlist*/, como si tratara de un fichero RTL más. Comprobar que anteriormente NO se ha añadido al proyecto por error el fichero *rtl/display_ctrl.vhd*.

Para poder simular esta "caja negra", añadir también su modelo de simulación, *netlist/display_ctrl.vhd.* **pero en este caso utilizar** *Add Sources > Add or create* ***simulation sources***.

Realizar la declaración e instanciación de este componente en *stopwatch*. Para esto, téngase en cuenta que la declaración de la entidad de *display_ctrl* se puede encontrar tanto en *netlist/display_ctrl.vhd* como en *rtl/display_ctrl.vhd*.

7- Finalizar el RTL del "top-level" *stopwatch.vhd*

En este apartado:

- Practicaremos la instanciación de bloques y su interconexión mediante *signals* internas declaradas por nosotros (ya habremos ido haciéndolo en los apartados anteriores).
- Practicaremos la aplicación tanto de *port map* para los puertos como de *generic map* para los *generics* que parametrizan sub-bloques (también habremos ido haciéndolo en los apartados anteriores).
- Adaptaremos nuestro diseño para que su funcionalidad pueda ser simulada (en cierta medida) en un tiempo razonable.

- Terminar de "cablear" el fichero "stopwatch.vhd" si quedaba algo pendiente. Comprobar que el chequeo sintáctico es correcto. Corregir los posibles errores y "warnings".
- Con el fin de "acelerar" el circuito durante las simulaciones de depuración de la funcionalidad, hacer lo siguiente:
  - Añadir un *generic* a "stopwatch" llamado "FAST_SIMULATION", de tipo *boolean* y valor por defecto *FALSE*. El objetivo es que este *generic* haga que el cronómetro se acelere cuando sea *TRUE*, para poder realizar más rápidamente las simulaciones. A continuación, veremos cómo hacerlo.
  - Usar una asignación concurrente condicional para sustituir el cable a la salida de *freq_divider* por un '1' cuando "FAST_SIMULATION" sea *TRUE*. El multiplexor introducido deberá mantener en su salida todas las conexiones anteriormente existentes para la salida de *freq_divider*, es decir, también deberán quedar conectados a esta salida del multiplexor los bloques *debounce*.

  De esta forma eliminamos la división de frecuencia por 12.500.000, haciendo que el contador de centésimas de segundo avance al ritmo del reloj. Esto nos permitirá, al principio del desarrollo, observar en simulación si el circuito está funcionando bien sin tener que esperar unos tiempos de simulación muy largos.

8- Preparar el testbench para la simulación:

En este apartado:

- Haremos un ejemplo sencillo de testbench, instanciando el bloque a verificar y creando dos procesos: uno para la generación del reloj y otro para el control principal de la simulación, generando otros estímulos. Al contrario que en un caso real, para simplificar, este testbench *no* comprueba por sí mismo la corrección del diseño, sólo permite la comprobación mediante ondas.
- Gracias al generic incorporado al bloque "top-level" de nuestro diseño, ahora lo configuraremos en "modo simulación" para que ésta se ejecute en un tiempo razonable.

- Crear **en el directorio *sim/*** un nuevo fichero fuente "stopwatch_tb.vhd". Editar el código fuente del testbench:

- Como siempre, se puede utilizar el editor de Vivado o cualquier otro. Eso sí, al crear el fichero, o al añadirlo al proyecto si lo editamos exteriormente, esta vez hay que utilizar *Add Sources > Add or create **simulation sources***. De lo contrario, esto es, si añadiésemos el fichero como cualquier VHDL de tipo RTL, Vivado intentaría sintetizarlo, lo cual no tiene sentido. (Nota: en caso de añadirlo incorrectamente, siempre podemos desactivar su uso para síntesis en el panel *Source File Properties*, bajo *Sources*).
- Declarar el componente *stopwatch*. Nota: el valor del generic en la declaración del componente ha de ser el mismo que en su entidad.
- Escribir la instancia de este componente, usando un valor *TRUE* en el *generic map* (este valor, que acelera la simulación, se podría usar en un proyecto real hasta haber terminado de depurar el código). Asegurarse de tener declarada una señal por cada puerto de *stopwatch* (por ejemplo, se puede usar para estas señales el mismo nombre que tienen los puertos, pero con la letra inicial en minúscula). Conectar estas señales a los puertos.
- Declarar una constante "CLK_PERIOD" de tipo *time* para nuestro periodo de reloj, igual a 8 ns (dado que el reloj de nuestra placa es de 125 MHz).
- Declarar una señal "endSimulation" de tipo *boolean*, inicializada a FALSE. La pondremos a TRUE en el proceso principal para avisar a otros procesos de que queremos terminar la simulación.
- Crear un proceso que genere en la señal *clk* un reloj de periodo CLK_PERIOD hasta que *endSimulation* se haga TRUE, momento en que el proceso deberá quedar suspendido para siempre.
- Crear otro proceso que implemente la siguiente secuencia principal de test:
  - Inicializar al principio (t=0) todas las entradas a "stopwatch"
  - Crear una fase de reset inicial de duración 12 periodos de reloj.
  - Tras esto, hacer un "pulsado" del botón de Start/Stop de duración 1050 ns.
  - Tras bajar Start/Stop, esperar un tiempo definido en una constante SIM_TIME, que inicialmente declararemos con un valor de "85 us" (ojo, 85 microsegundos, no nanosegundos). Tras esto activar el flag *endSimulation* y matar el proceso con un *wait*.

9- Simular el diseño funcionalmente:

Lanzar la simulación, como se hizo en el tutorial de la primera práctica: en el panel izquierdo (*Flow Navigator*) seleccionar en el bloque *SIMULATION* la acción *Run Simulation*. Elegir la única opción que estará inicialmente disponible, *Run Behavioral Simulation*.

Si tenemos algún error en el código, el lanzamiento de la simulación fallará. En tal caso, habrá que buscar la causa del problema en el *log* que aparece en la pestaña *Tcl Console* (en la parte de abajo de la herramienta). Lo último que mostrará este *log* aparecerá en rojo y normalmente no será muy útil para diagnosticar el problema, sino que tendremos que irnos a algún mensaje de error que haya aparecido más arriba (haciendo *scroll* en esta consola, que podemos maximizar temporalmente si nos resulta más cómodo). Por ejemplo, si por error hemos escrito "std_lugic" en vez de "std_logic" nos podemos encontrar algo como:

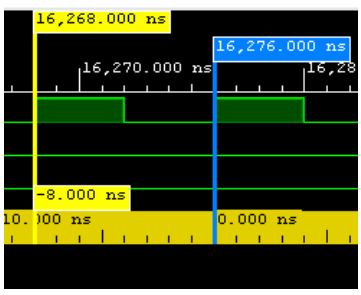
ERROR: [VRFC 10-2989] 'std_lugic' is not declared […../P2_stopwatch/sim/stopwatch_tb.vhd:64] ← Error en la línea 64

Una vez hayamos corregido los errores en el código, nos aparecerá la configuración de ventanas de SIMULATION. Podremos ver que la simulación ha corrido el tiempo que tiene el proyecto configurado por defecto, normalmente 1000 ns. Para que realice la simulación completa, deberemos hacer clic en el botón del triángulo de la barra principal (*Run All*).

Comprobar en la ventana de ondas del simulador que el circuito hace lo esperado. Para ello, pensar bien qué mirar, añadiendo a la ventana de ondas las señales **internas** que puedan resultar de interés. Es cierto que ver en las ondas que todos los detalles del diseño del cronómetro funcionan bien es muy difícil, pero está claro que podemos al menos visualizar rápidamente ciertas ondas que nos permitan ver si nuestro diseño sí que "parece estar comportándose" como el cronómetro que pretendemos que sea.

Para manejar el simulador, tener en cuenta lo siguiente:

- Recordar que los mensajes de la simulación aparecen en la pestaña *Tcl Console*.
- Cuando añadamos señales internas, éstas típicamente aparecerán sin dibujar, porque el simulador no habrá guardado su evolución al correr la simulación. Para que estas ondas se rellenen, podemos relanzar la simulación desde t=0, sin recompilar, haciendo clic sucesivamente en los botones *Restart* (triángulo con línea a la izquierda) y *Run all* (triángulo) de la barra de botones principal.
- Recordar que podemos configurar las ondas para que se muestren con distinto formato, haciendo clic con el botón derecho sobre los nombres de una o más ondas seleccionadas y ajustando cosas como *Radix* o *Signal Color*. Podemos guardar en disco la configuración que tengamos para la visualización de las ondas con *Ctrl+S* (o *File | Simulation Waveform | Save Configuration*). Se nos preguntará por un nombre de fichero (con extensión *.wcfg*) donde guardar esta configuración, proponiéndosenos uno por defecto. También se nos preguntará si queremos añadir este fichero de configuración de ondas al proyecto, de modo que siempre que lancemos la simulación nos aparezca ya la ventana de ondas tal como la hemos configurado.
- Para movernos por la simulación, podemos usar los botones de *Zoom*, etc. de la propia ventana de simulación. Es muy cómodo también hacer sobre las propias ondas un arrastre ("drag") con el botón izquierdo del ratón, de forma que, según hacia dónde arrastremos el ratón, podremos hacer *Zoom Fit*, *Zoom In*, *Zoom Out* o *Zoom Range*. Esto último (arrastrando hacia abajo a la derecha) nos permite seleccionar cómodamente la zona de ondas que queremos ver.
- Para realizar medidas de tiempo entre flancos de señales podemos por ejemplo hacer lo siguiente. Primero situar el cursor amarillo sobre la ventana de ondas usando el botón izquierdo del ratón en cualquier punto de ésta, y arrastrarlo sobre un flanco hasta que veamos que aparece un círculo relleno. A continuación, añadir un *marker*, por ejemplo, con el botón *Add Marker* ( ), o con botón derecho, *Markers|Add Markers*. Arrastrar el marcador azul hasta el segundo flanco. En la parte de arriba de la ventana de ondas podremos ver la posición temporal de los marcadores y del cursor y en la parte de abajo la separación entre ellos:



- Para cerrar la simulación podemos hacer clic en el botón de cierre situado a la derecha en la barra azul "SIMULATION".
- Si cambiamos el código, deberemos cerrar la simulación y lanzarla otra vez, no basta un *Restart*.

Comprobar el funcionamiento y en caso de encontrar cualquier problema, buscar su causa y corregir los posibles errores. Si una señal no se comporta como esperamos, pensar de qué otras señales depende ésta, hasta encontrar la causa raíz del problema.

También se puede continuar con los siguientes dos puntos, intentando realizar la implementación del circuito. Esto puede hacer aparecer errores de síntesis que también nos den información sobre dónde tenemos problemas, que podemos corregir y tras ello volver a este punto de comprobación de la simulación. Por supuesto, habrá que repetir la implementación una vez corregidos los posibles errores.

NOTA: en el directorio *sim/* se incluyen *scripts* de simulación de *ModelSim*, por si alguien desea probar a utilizar esta herramienta (esto es, no se requiere el uso de estos ficheros *.do*).

10- Asignar los pines de la FPGA:

- Añadir al proyecto el fichero "stopwatch.xdc", del directorio "vivado".
- Examinar en él las constraints de posicionado de pines. Falta la asignación de pin para la señal *StartStop* (pulsador BTN1). **Usando el manual de la placa, completar el fichero de constraints con la asignación de este pin** (ver *Figure 14* en la sección *Basic I/O*).

11- Realizar la implementación

- Lanzar la síntesis del circuito. Corregir posibles errores hasta que esta termine sin problemas.
- A continuación, ejecutar *Run Implementation*. Al terminar, abrir el "Implemented design".
- En la consola TCL de Vivado ejecutar "*report_io*" (teclear este comando). Comprobar que las localizaciones de los pines son las especificadas en el fichero de *constraints*. En particular comprobar el pin que originalmente faltaba en este fichero.
- Proceder con *Generate Bitstream* (cuando salga el cuadro de diálogo avisándonos de que se ha terminado con éxito este proceso, se puede seleccionar *View Reports* o simplemente *Cancel*). Tras esto tendremos un diseño listo para bajar a la placa.

Tras realizar generar el *bitstream*, dedicar unos pocos minutos a un pequeño análisis de los resultados de la implementación. Para ello, ver en el *Project Summary* los siguientes datos (**NO se pide** incluir estas respuestas en una memoria):

- ¿Cuántas LUTs ha usado el diseño? ¿Porcentualmente respecto al total disponible?
- ¿Cuántos Flip-Flops? ¿Porcentualmente respecto al total disponible?
- ¿Cuántas IOs se han usado?
- ¿Cuántos buffers de reloj (BUFGs) tiene nuestra FPGA y cuántos hemos usado?

Las LUTs son los elementos básicos combinacionales (aunque a veces se usan como memorias o shift registers). Los Flip Flops son los elementos básicos secuenciales. Aproximadamente podemos considerar la ocupación de nuestra FPGA en términos porcentuales, para cada una de estas facetas, lógica y registros,

como el uso de LUTs y Flip Flops que aparece en el reporte. Respecto a las I/O, su número viene dado por el conjunto de entradas y salidas de nuestro diseño. Los buffers de reloj permiten distribuir un reloj a una gran cantidad de flip-flops destino con un mínimo "skew" (máxima diferencia entre el tiempo de llegada a unos y otros flip-flops).

Tras realizar lo anterior, se recomienda, si es posible, saltar al punto 13 de esta guía para comprobar en la placa el diseño realizado, volviendo después a las tareas adicionales del punto 12. Si no se tiene la placa a disposición, se puede ir avanzando con estas tareas adicionales.

12. Tareas adicionales para poder sacar la mejor nota

Se proponen a continuación dos posibles tareas diferentes (12.1 y 12.2). Si no se hace ninguna de las dos, la nota máxima en esta práctica estará limitada a **8 puntos** sobre 10. La realización de cualquiera de ellas permitirá puntuar sobre el total de 10 puntos (se pueden realizar también las dos tareas y ambas serán tenidas en cuenta para la nota).

12.1 (tarea adicional 1) - Completar el bloque de control de los *displays* 7-segmentos

En este apartado:

- Practicaremos la implementación de código de una manera más libre.
- Realizaremos una comunicación de datos con conversión paralelo a serie.
- Generaremos relojes de salida de la FPGA, que se utilizarán para registrar datos en un circuito externo.

Esta tarea consiste en sustituir la "caja negra" *display_ctrl* por nuestro propio código VHDL. Para ello, se entrega a los alumnos en el directorio *rtl/* un fichero *display_ctrl.vhd* incompleto, como punto de partida. Proceder como se describe a continuación:

Hacer una copia del directorio completo "P2_stopwatch" de nivel superior (el que contiene *rtl*, *sim*, etc.). Renombrar esta copia como "P2_stopwatch_dc".

Abrir el proyecto Vivado contenido en esta copia, ejecutando en la ventana principal de Vivado *Open Project* y seleccionando el fichero *stopwatch.xpr* bajo *P2_stopwatch_dc/vivado/stopwatch.*

Sacar del proyecto los ficheros correspondientes a la *netlist* del bloque *display_ctrl*. Para ello, hacer clic derecho sobre el bloque *display_ctrl* en la jerarquía que se muestra bajo *Design Sources* en el panel *Sources* y ejecutar *Remove File from Project…* Repetir la misma operación con el *display_ctrl* bajo la jerarquía *Simulation Sources*. Con esto habremos sacado del proyecto tanto el fichero *display_ctrl.dcp* como *display_ctrl.vhd*.

Añadir al proyecto, como fichero fuente, el fichero *display_ctrl.vhd* situado en el directorio *rtl/*. Veremos que este bloque vuelve a aparecer en ambas jerarquías.

A la hora de implementar este bloque ha de tenerse en consideración lo siguiente:

- El fichero *display_ctrl.vhd*, incompleto, contiene la declaración de la entidad y una arquitectura con dos funciones que serán útiles para implementar este bloque:
  - *dec_to_7seg* : convierte un número de un dígito, dado por los 4 bits que lo representan, en un bus con formato {dp,g,f,e,d,c,b,a}, siendo *a..g* la nomenclatura estándar de los leds de un dígito 7-segmentos y *dp* el punto decimal, que se introduce como un segundo argumento a la función. **Ver la página 11 del manual del módulo display (sección 3, "OHO_DY1 Block Diagram")**.
  - *map_segments* : convierte un std_logic_vector con formato {dp,g,f,e,d,c,b,a} en otro std_logic_vector con los bits reordenados según han de quedar en el *shift register* que se muestra para cada dígito en la mencionada página del manual. El bit 7 es el que queda a la izquierda en esta figura, y por tanto el primer bit a introducir

en la entrada "SER" del display será el bit 7 del valor de las decenas de segundos, expresado en este formato.

- A la hora de utilizar *dec_to_7seg* habrá que considerar cuándo hay que pasarle como argumento un '1' en el punto decimal, teniendo en cuenta que este punto está físicamente en la esquina inferior derecha de cada dígito.
- Lo esencial del diseño a realizar es una conversión de paralelo a serie. Hay que capturar en paralelo sobre un *shift register* la información entrante e irla sacando bit a bit hacia el display.
- Lo anterior hay que hacerlo en varias etapas:
  a) Capturar en un shift register la representación del valor de cuenta del cronómetro obtenido por las funciones entregadas.
  b) Mover las veces necesarias el reloj SCLK del display, según se va desplazando el contenido del shift register, que va saliendo de la FPGA y se va introduciendo por la pata SER del display.

     Hay que tener en cuenta que SER no debe cambiar a la vez que el flanco de subida de SCLK, pues el display podría capturar un valor incorrecto (se podría asegurar que no hay problemas, pero requeriría establecer un conjunto de *timing constraints* que nos complicaría bastante el diseño, de forma innecesaria).

     Por otra parte, el manual indica un valor máximo de frecuencia para SCLK (ver más abajo en la misma página del manual). Será necesario asegurarse de no mover SCLK demasiado rápidamente.
  c) Mover RCLK para que su flanco de subida haga al display volcar sus shift registers sobre sus registros paralelos conectados a los leds.
- La presencia de distintas etapas sugiere la utilización de una máquina de estados, aunque no es la única forma posible de implementar este hardware. En el caso de utilizar una máquina de estados, esta NO tiene por qué tener el formato de dos procesos (combinacional + secuencial), de hecho, se sugiere utilizar un único proceso secuencial para practicar esta otra forma de codificar FSMs.

Realizar la implementación del bloque y utilizar el simulador para comprobar su correcto funcionamiento.

12.2 (tarea adicional 2) – Añadir un modo diferente de funcionamiento

En este apartado:

- Practicaremos más codificación VHDL.
- Veremos qué tareas implica añadir una nueva entrada al circuito.

El objetivo en este caso es el siguiente: se utilizará el conmutador SW0 para seleccionar entre dos modos de funcionamiento del circuito:

- Si SW0 está a '0' (abajo), tendremos el comportamiento habitual de cronómetro.
- Si SW0 está a '1' (arriba), la cuenta solo se moverá mientras mantengamos pulsado el botón *StartStop*. La cuenta parará inmediatamente si levantamos el dedo de este botón.

Hacer una copia del directorio completo "P2_stopwatch" de nivel superior (el que contiene *rtl, sim,* etc.). Renombrar esta copia como "P2_stopwatch_switch".

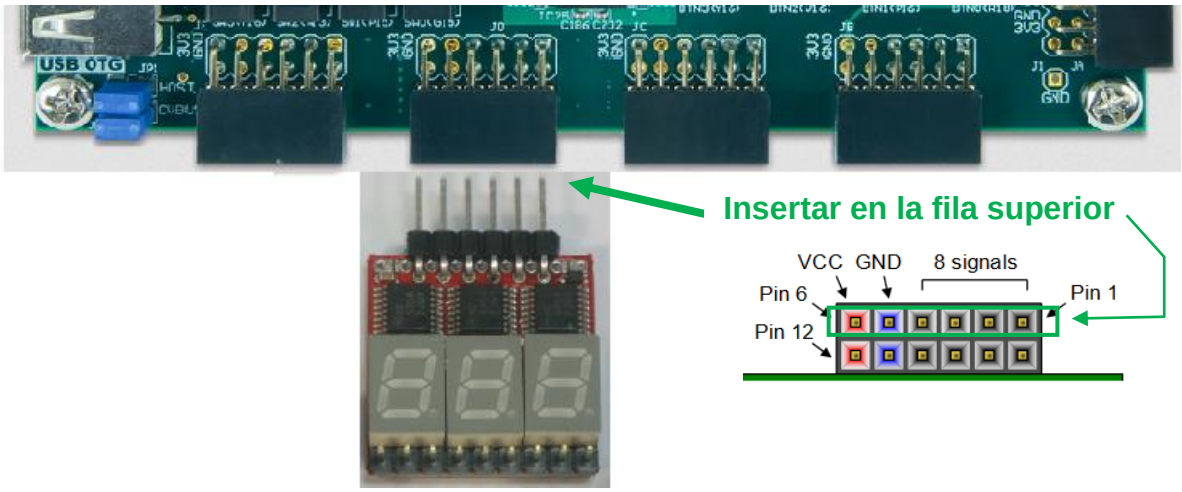
Abrir el proyecto Vivado contenido en esta copia, ejecutando en la ventana principal de Vivado *Open Project* y seleccionando el fichero *stopwatch.xpr* bajo *P2_stopwatch_switch/vivado/stopwatch.*

Hacer en este proyecto todas las modificaciones necesarias para lograr la funcionalidad deseada y comprobarla en simulación. No olvidar actualizar el fichero de *constraints* con la localización de la nueva entrada (consultar la sección *Basic I/O* del manual de la placa).

13- Bajar el diseño a la placa:

A continuación se comprobará el funcionamiento del diseño en la placa. Si se ha realizado alguno de los dos diseños alternativos del punto anterior, probar la placa con cada uno de los proyectos realizados.

- Asegurarse de que se ha generado el *bitstream* tras cualquier cambio que se haya podido realizar al código (podría estar la simulación funcionando correctamente pero el *bitstream* corresponderse con una versión anterior del código, con errores funcionales).
- Conectar el módulo *pmod* con los dígitos 7-segmentos en la **fila superior** del conector ***JD***:



- Conectar la placa al PC con el cable USB. **PRESTAR MUCHA ATENCIÓN PARA INTRODUCIR EL CONECTOR MICRO-USB DE FORMA CORRECTA** (este conector se puede forzar con facilidad para ser introducido al revés, dañando la placa).

- Comprobar que el *jumper* JP7 está en la posición *USB* (para recibir la alimentación por este conector).
- Encender la placa (*power switch SW4* en posición *ON*). Se encenderá un led rojo.

- Cargar en la FPGA el diseño. Para ello, ejecutar *Open Hardware Manager*, en la sección *PROGRAM AND DEBUG* del *Flow Navigator*.
    - Bajo la cabecera del HARDWARE MANAGER, hacer clic en *Open Target* y después en *Auto Connect*.
    - El panel *Hardware* se rellenará con información, apareciendo una línea con el modelo de nuestra FPGA y a la derecha su estado: *Not programmed*. Hacer clic con el botón derecho en esta línea y ejecutar *Program Device…*.
    - En el cuadro de diálogo que se abre, comprobar que en la casilla *Bitstream file* aparece el fichero *.bit* correspondiente al proyecto que queremos descargar en la FPGA. Si no es así (puede apuntar a un fichero que hayamos utilizado anteriormente), seleccionar el fichero adecuado haciendo clic en los puntos suspensivos de la derecha.
    - Hacer clic en *Program* y comprobar cómo se enciende el led verde *DONE* de la placa.
- **Comprobar el funcionamiento de la placa**. ATENCIÓN: hay que tener en cuenta que para que este cronómetro funcione bien los botones no se pueden pulsar con demasiada rapidez.
- Cerrar el *HARDWARE MANAGER* y apagar la placa.

Para pensar

Dedicar 10 minutos a intentar razonar una respuesta para las siguientes preguntas (**NO se pide** incluir estas respuestas en una memoria):

- ⇨ Si hemos visto la simulación "acelerada", ¿por qué en la placa veremos moverse el cronómetro a la velocidad normal (si el código es correcto)?
- ⇨ Mirando el esquema general del diseño mostrado al comienzo de este enunciado y sabiendo ya cómo están hechos los sub-bloques, ¿qué camino ("path") de propagación combinacional destaca en el diseño por recorrer varios sub-bloques entre un flip-flop origen y un flip-flop destino? (no hace falta buscar en reportes de la herramienta).
- ⇨ ¿Qué diferencia hay entre lo que ocurre en la placa al pulsar el botón de reset asíncrono (BTN3) y lo que ocurre al pulsar el botón de puesta a cero (BTN2)? ¿Por qué?

Entrega de la práctica

Deberá entregarse en Moodle un archivo comprimido (zip o rar) con el diseño realizado, antes de la fecha límite especificada en el calendario del laboratorio:

- Entregar el directorio "P2_stopwatch" completo, así como los directorios completos "P2_stopwatch_dc" y/o "P2_stopwatch_switch" en el caso de que se haya hecho alguna de las tareas adicionales del apartado 12. Entregar todo ello en el mismo archivo comprimido.
- Todos los ficheros del código fuente escritos completa o parcialmente por los alumnos deberán entregarse con una cabecera adecuada, incluyendo siempre el **nombre de los autores** y una descripción del fichero (modificar con este fin la cabecera de los ficheros entregados).
- Todo el código fuente deberá llevar comentarios apropiados, tanto en contenido como en cantidad.

Nota: si el zip generado para la entrega es demasiado grande se puede borrar el subdirectorio *P2_stopwatch/vivado/stopwatch/stopwatch.sim*.

Memoria: NO es necesario entregar ninguna memoria.

Demostración de funcionamiento: El profesor podrá requerir a los alumnos mostrar la placa funcionando y/o una simulación que demuestre que el circuito aparenta funcionar como se espera.

El archivo comprimido deberá tener **obligatoriamente** la siguiente **nomenclatura**:

```
<<n° de grupo>>_<<n° de alumno (dos dígitos)>>_<<n° de práctica>>.[zip|rar]
```

(Ej.: `3311_05_2.zip` para el alumno 5 del grupo de los lunes)

En Moodle estará disponible una lista con los números de alumno a utilizar.

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