Práctica 2

Completar el diseño de un cronómetro

Objetivos

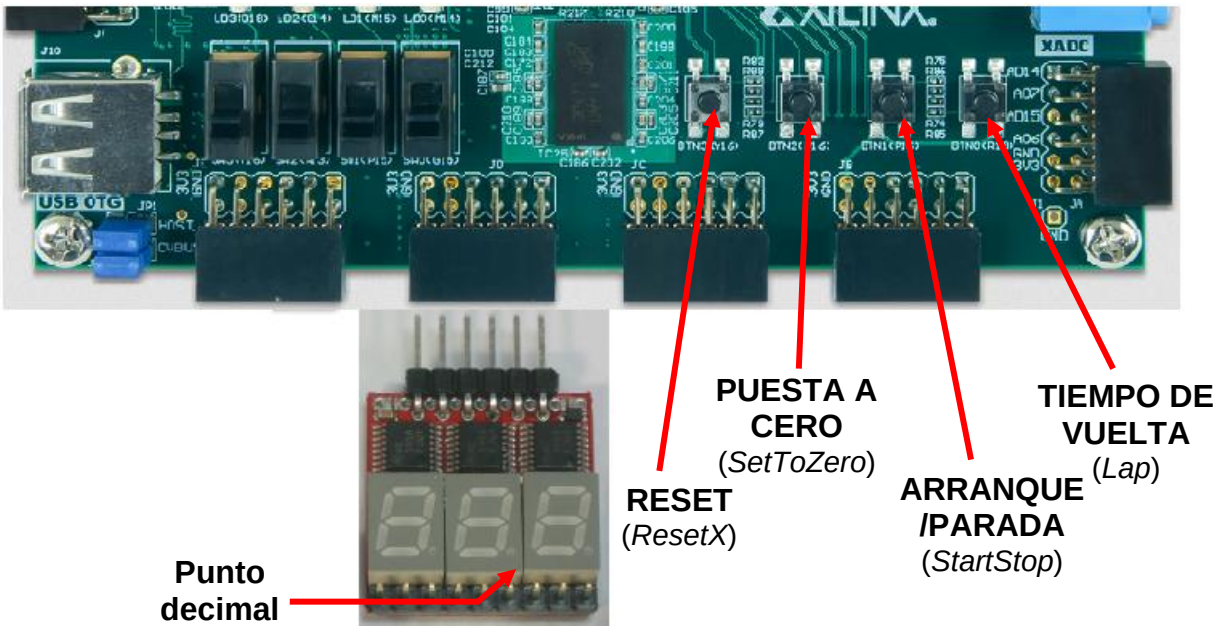
- Realizar el flujo de diseño con un ejemplo más complejo que el de la Práctica 1.
- Practicar la codificación en lenguaje VHDL, desde estructuras simples hasta más avanzadas como funciones, *generics* y *packages*.
- Ver un mecanismo para acelerar las primeras simulaciones.

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.

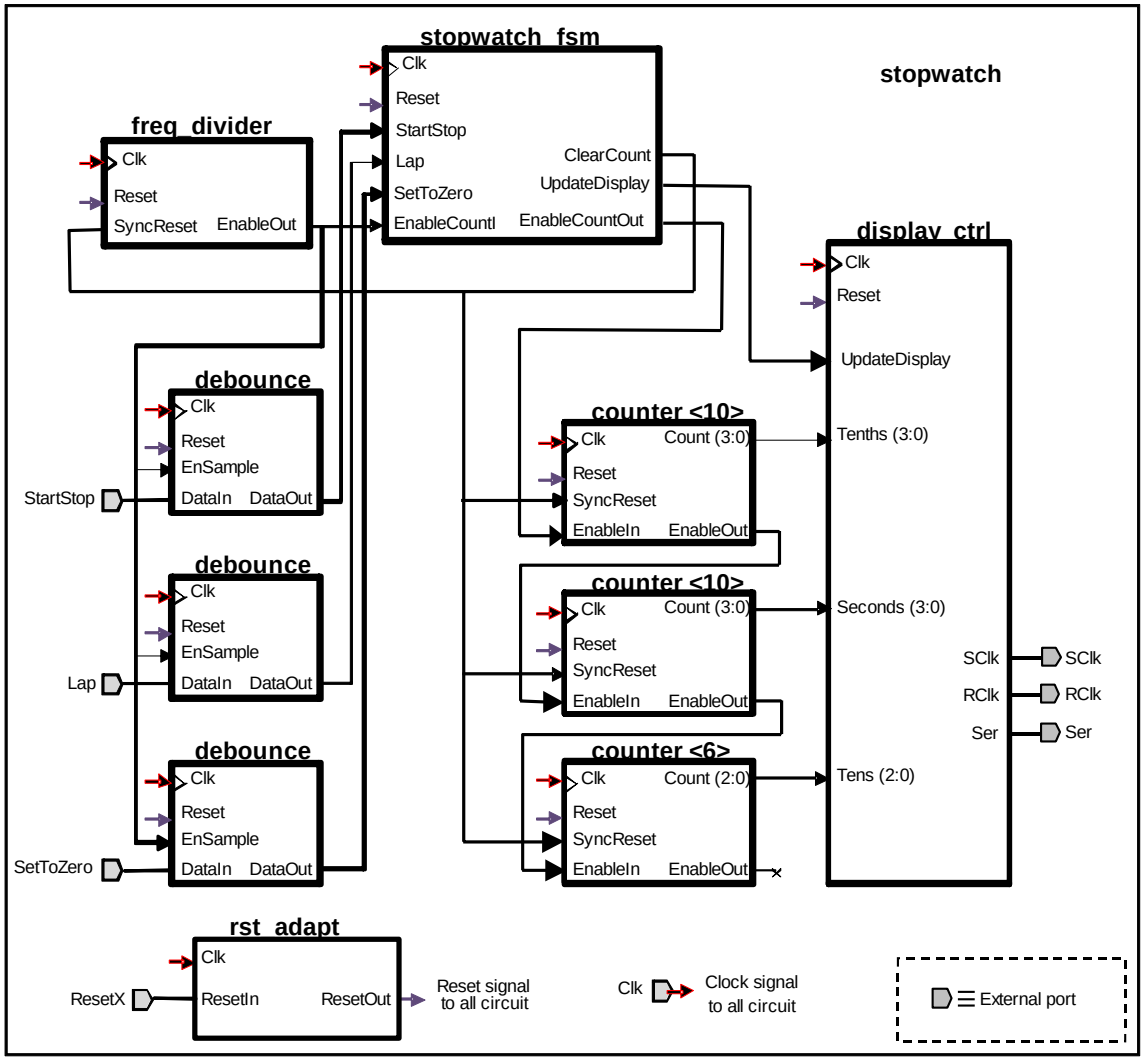
Circuito utilizado

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:



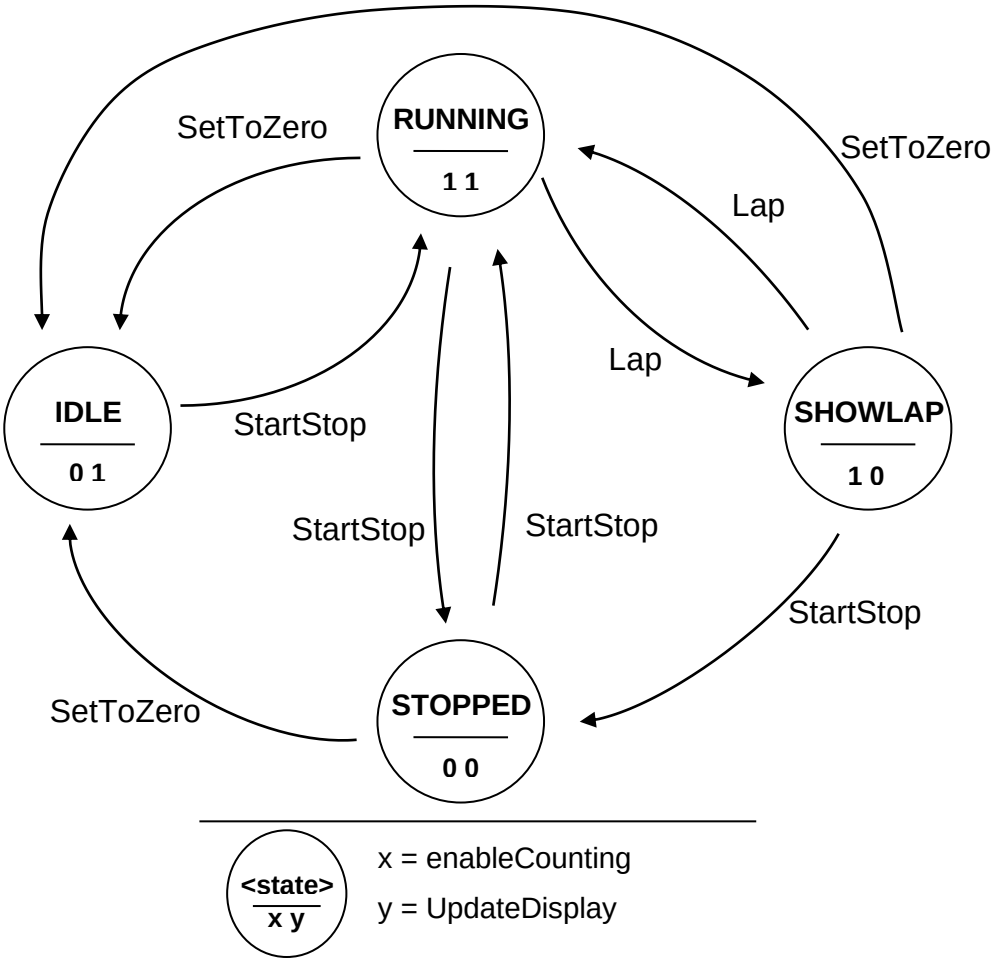
Un bloque de división de frecuencia (*freq_divider*) se encarga de dividir por 12.500.000 el reloj de 125 MHz de la placa para dar un pulso de un ciclo cada décima de segundo. A partir de esta base de tiempos, tres contadores en cascada (*counter*) proveen la cifra de décimas de segundo (*tenths*), unidades 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 simulación)
                        runsim_netlist.do -> completo (script simulación)

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

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 *P2_stopwatch* situado bajo la carpeta *P2_stopwatch/vivado*. Como en la primera práctica, se tratará de un *RTL Project* configurado para la FPGA XC7Z010CLG400-1. Durante la creación pueden añadirse ya los ficheros presentes en el directorio *rtl/*, pero **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 **deberán situarse siempre en el directorio *rtl/***.
- En este fichero, 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*); 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*. Además, habrá que tener en cuenta que el contador sólo debe avanzar (o bien hacer "wrap-around") 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- Completar el bloque de control de los displays 7-segmentos

Este bloque de control (*display_ctrl*) es de desarrollo opcional, pudiéndose utilizar en su lugar los ficheros entregado en el directorio *netlist/*. Echar un vistazo a los tres ficheros presentes en este directorio; los tres representan la misma netlist (lista de células + interconexión) en tres formatos diferentes: VHDL, EDIF (formato de netlist estándar en la industria) y DCP (formato propietario de lo que en Vivado se llama un "design checkpoint").

La consideración de "opcional" de este desarrollo ha de entenderse como que no se exige este trabajo para aprobar la práctica. Sin embargo, **las entregas que no incluyan este desarrollo estarán limitadas a una nota máxima de 8 puntos**.

Si no se desarrolla el código fuente de este bloque o se prefiere completar inicialmente el diseño utilizando el fichero de netlist entregado (se recomienda hacer esto), bastará utilizar *Add Sources* para añadir al proyecto el fichero *display_ctrl.dcp* del directorio *netlist/* y realizar la instanciación de este bloque. Para esto último téngase en cuenta que la declaración de su entidad se puede encontrar tanto en *netlist/display_ctrl.vhd* como en *rtl/display_ctrl.vhd*.

La descripción que sigue en este apartado se refiere a la implementación del código fuente del bloque. Si de momento se va a usar la netlist, saltar al siguiente apartado.

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 periférico externo.

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

- Si antes se ha añadido al proyecto este bloque en formato netlist, deberá ser retirado del proyecto y añadir el fichero de código fuente.
- Se entrega un 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 según 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.

- Añadir la declaración e instanciación de “display_ctrl” a “stopwatch.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:

- Se puede utilizar el editor de Vivado o cualquier otro. Si se usa el editor de Vivado se puede crear el fichero directamente dentro del proyecto con *Add Sources* > *Add or create simulation sources*. Sin embargo, no es necesario tener el testbench dentro del proyecto porque utilizaremos otro simulador. Por otra parte, de incluirlo en el proyecto hay que asegurarse de que en sus *Source File Properties* (panel bajo el panel *Sources*) no esté su uso para *Synthesis*, pues de lo contrario Vivado intentará sintetizarlo.
- 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 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.
- 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 5 periodos de reloj.
    - Tras esto, hacer un "pulsado" del botón de Start/Stop de duración 1000 ns.
    - Tras bajar Start/Stop, esperar un tiempo definido en una constante SIM_TIME, que inicialmente declararemos con un valor de "70 us" (ojo, 70 microsegundos, no nanosegundos). Tras esto activar el flag *endSimulation* y matar el proceso con un *wait*.

9- Simular el diseño funcionalmente:

En clase se utilizará el simulador *Questa* de Mentor, muy similar a *ModelSim*, que puede conocerse de otras asignaturas. Utilizaremos este simulador como herramienta independiente de Vivado, utilizando **scripts** para la compilación y ejecución de la simulación. Se entregan a tal efecto los scripts *runsim.do* y *runsim_netlist.do*.; estos ficheros están completos, pero conviene entender su funcionamiento pues en prácticas posteriores no se entregarán completos. Se utilizará un script u otro en función de si para el bloque *display_ctrl* se desea compilar el código fuente desarrollado (*runsim.do*) o bien el fichero de netlist entregado a los alumnos (*runsim_netlist.do*).

Ejecutar **vsim –gui &** para arrancar la interfaz gráfica del simulador Questa.

En la consola del simulador, situarse en el directorio *sim/* utilizando los comandos habituales de Linux.

Lanzar la simulación completa ejecutando **do runsim.do**, o bien **do runsim_netlist.do**. Durante la depuración de errores sintácticos también se puede pedir a la herramienta que compile un único fichero utilizando el mismo comando **vcom** presente en el script.

> Nota: si se pasa de simular en casa con Windows a en el laboratorio con Linux o viceversa, puede ser necesario borrar la carpeta *sim/work*.

La primera vez que la compilación de los ficheros sea correcta y el simulador continúe sin problemas con la elaboración (letras azules en la consola) aparecerá un error ("** *Error: Cannot open macro file: wave.do*" o algo similar). Esto es porque el script principal invoca a un script *wave.do* donde almacenar las preferencias de visualización de las ondas. Añadir las señales del testbench a las ondas y, con la ventana de ondas seleccionada, ejecutar *File > Save Format…* Por defecto se grabará un fichero *wave.do*. Volver a lanzar el script principal. En adelante se podrá guardar la configuración de las ondas (nuevas señales, cambio de base en buses, etc.) cuantas veces se quiera sobrescribiendo *wave.do*.

Si el simulador reporta algún error o warning, corregirlo y repetir el proceso. Los errores se pueden corregir en el editor del propio simulador, haciendo doble click sobre cada error, o cambiando de tarea al editor utilizado. Tras corregir el fichero se puede reintentar sin salir del simulador, ejecutando el último comando en la consola (clicando en ella y pulsando la flecha hacia arriba del teclado y <*Enter*>).

Comprobar en la ventana de ondas del simulador que el circuito hace lo esperado (pensar bien qué mirar).

> Nota: al contrario que el funcionamiento por defecto del simulador de Vivado, que corre la simulación por un tiempo predefinido en la configuración, el script utiliza el comando "run -all", que hace que el simulador corra hasta que no tenga más que hacer.

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*. **Usando el manual o la serigrafía de la placa, completar el fichero de constraints con la asignación de este pin.**

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". 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*.

12- Bajar el diseño a la placa:

- Conectar el módulo *pmod* con los dígitos 7-segmentos en la fila superior del conector *JD* (ver imagen al principio de este enunciado).
- Conectar la placa al PC y cargar en la FPGA el diseño tal como se hizo anteriormente en la Práctica 1.
- **Comprobar el funcionamiento de la placa**. ATENCIÓN: hay que tener en cuenta que los botones no se pueden pulsar con demasiada rapidez.

Análisis de resultados de la implementación

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).

Para pensar

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 vemos moverse el cronómetro a la velocidad normal?
- ⇨ 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).

Entrega de la práctica

El correcto funcionamiento del diseño, en simulación y sobre la placa "ZYBO", deberá mostrarse al profesor de prácticas en clase, antes de la fecha límite especificada en el calendario del laboratorio.

Por otra parte, **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. Si el archivo comprimido ocupara más de 5 MB, eliminar tan sólo los ficheros de ondas de simulación, que son los que a veces pueden ocupar más espacio ("wlf*", "*.wlf"…), y que estarán localizados en el subdirectorio "sim".
- 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.

Memoria: NO es necesario entregar ninguna memoria.

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

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

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

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