Práctica 2

Completar el diseño de un cronómetro

Objetivos

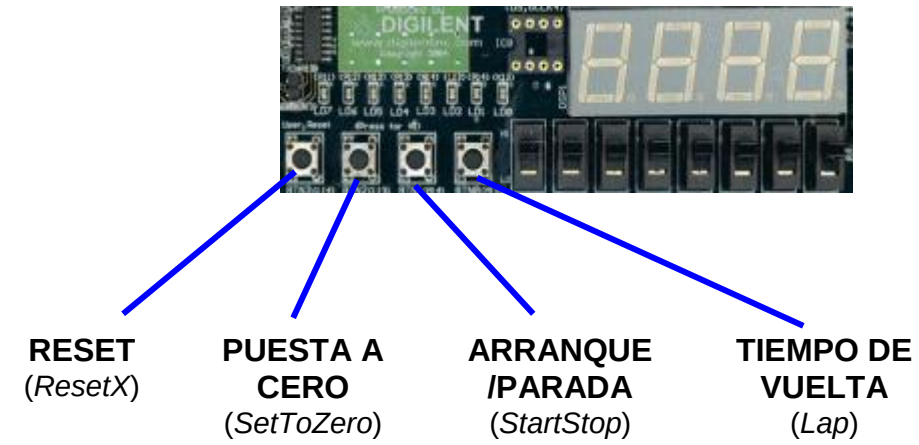
- Realizar el flujo de diseño con un ejemplo más complejo que el del Ejercicio 1.
- Practicar la codificación en lenguaje VHDL, desde estructuras simples hasta más avanzadas como funciones, *generics* y *packages*.
- Ver un mecanismo para acelerar las primeras simulaciones.

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.

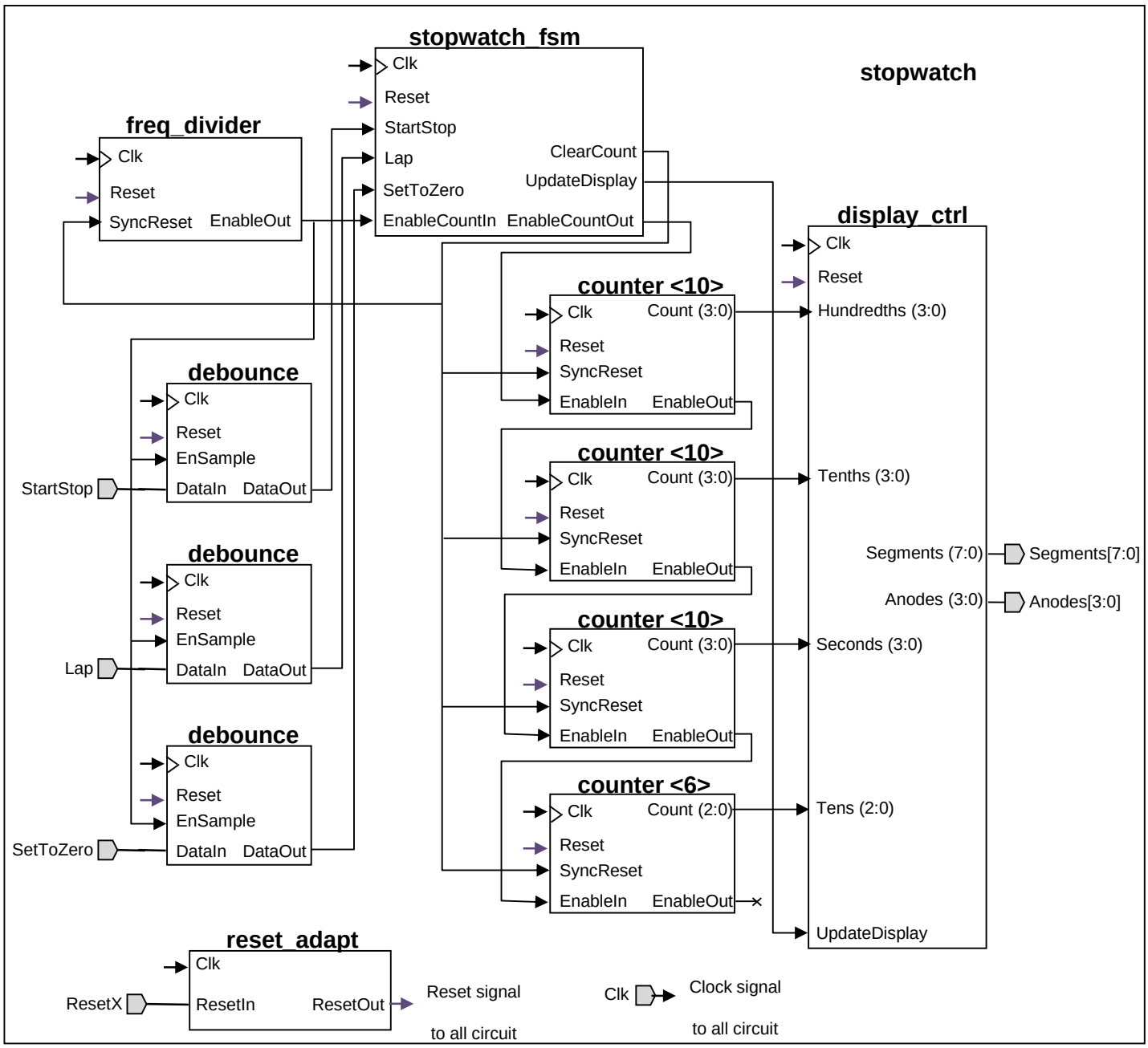
Circuito utilizado

En este ejercicio implementaremos sobre la placa *Spartan 3 Starter Kit Board* un cronómetro de cuatro dígitos, con segundos (0-59), décimas y centésimas. El cronómetro se visualizará sobre los cuatro displays 7-segmentos. Usaremos los pulsadores BTN3 a BTN0, por este orden, para lo siguiente:

- Reset asíncrono general del circuito.
- Puesta a cero del cronómetro.
- Parada y rearranque (Start/Stop) de la cuenta.
- 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:

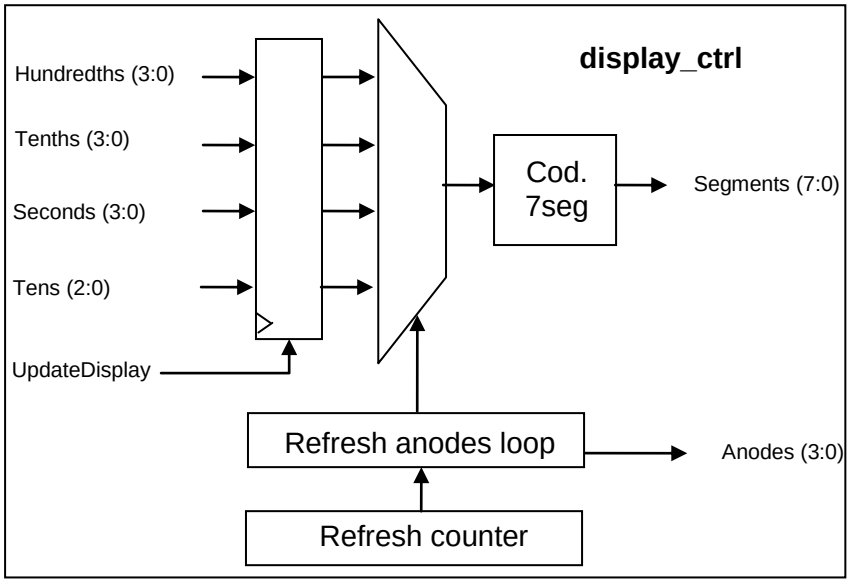


Un bloque de división de frecuencia (*freq_divider*) se encarga de dividir por 500000 el reloj de 50 MHz de la placa para dar un ciclo cada centésima de segundo. A partir de esta base de tiempos, cuatro contadores en cascada (*counter*) proveen la cifra de centésimas de segundo (*hundredths*), décimas 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).

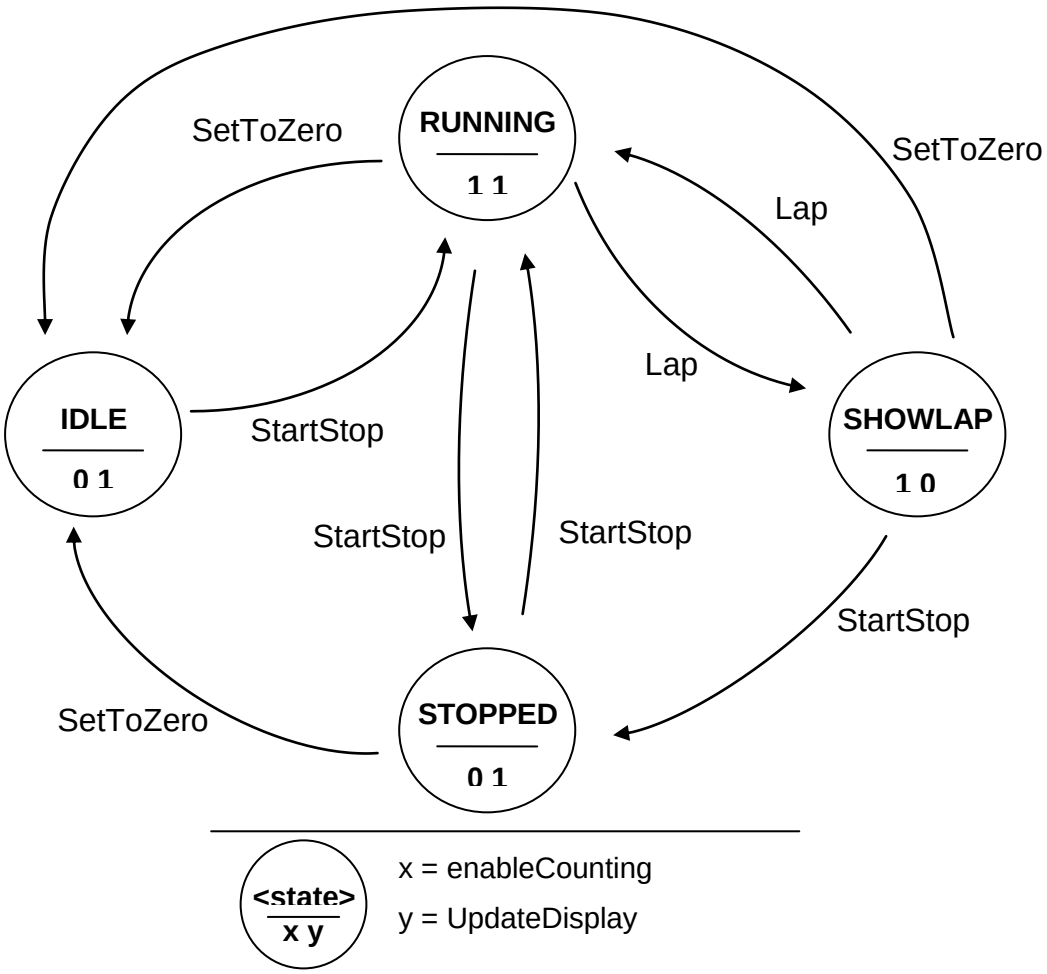
Unos bloques de anti-rebote (*debounce*) se encargan de adaptar los pulsadores de control, mientras que un bloque independiente (*reset_adapt*) sincroniza el reset externo que viene del otro pulsador.

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

Finalmente un bloque de control de los displays (*display_ctrl*) se encarga de convertir las cifras de cada dígito a código 7-segmentos y de rotar la activación de ánodo del display, que funciona multiplexando en el tiempo la señal de cada dígito. Adicionalmente este bloque tiene un registro a su entrada para permitir "congelar" la visualización mientras el cronómetro corre si se pulsa el botón "Lap". La siguiente figura detalla la estructura de este bloque, que se codifica en un único fichero VHDL (no es jerárquico):



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



Las señales de entrada mostradas en este diagrama en realidad han de corresponderse con señales internas a la máquina que sean '1' cuando las correspondientes entradas pasan de '0' a '1'. Por otra parte, la señal *enableCounting* también es interna, 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.

Guía para la realización del diseño

Se parte de la siguiente estructura de directorios y ficheros, que se puede encontrar en un fichero zip en la página web del laboratorio y que habrá que bajar en C:\DIE (ó C:\DCSE):

```
P2_stopwatch /
                  rtl/
                        counter.vhd       -> incompleto
                        debounce.vhd      -> completo
                        display_ctrl.vhd  -> incompleto
                        freq_divider.vhd  -> no se entrega
                        reset_adapt.vhd   -> completo
                        stopwatch.vhd     -> no se entrega
                        stopwatch_fsm.vhd -> incompleto
                        stopwatch_pkg.vhd -> completo

                  sim/
                        stopwatch_tb.vhd  -> no se entrega

                  ise/
                        stopwatch.ucf     -> incompleto
```

(El directorio P2_stopwatch está incluido en el zip, por lo que conviene pedir a la herramienta de descompresión que escriba directamente sobre C:\DIE)

Para terminar el diseño será necesario completar los ficheros fuente incompletos y crear los no entregados.

Comenzar ejecutando el programa *ISE Design Suite 14.1*.

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

- Usar *File / New Project*.
- Seleccionar el directorio donde se ubicará en *Location*, dando al botón "…". Elegir el directorio *ise/* (*P2_stopwatch/ise*).
- A continuación poner como nombre de proyecto "stopwatch". Según escribimos, la casilla *Location* se rellena con este nombre; borrarlo para que no se cree un nivel adicional de subdirectorio.

- Comprobar abajo que el *Top-level source type* es *HDL* y que los parámetros del siguiente cuadro de diálogo son los comentados en la primera práctica. Tras esto, dar a *Next* y a *Finish*.
- Ejecutar *Project / New Source*. Seleccionar VHDL module, seleccionar como *Location* el directorio *rtl* (*P2_stopwatch/rtl*) y como nombre poner "stopwatch".
- Dar a *Next* y a continuación introducir los puertos de esta entidad top-level de acuerdo con el diagrama de arriba: *Clk*, *ResetX*, *StartStop*, *Lap*, *SetToZero*, *Segments* (8 bits) y *Anodes* (4 bits). Cambiar el nombre de la arquitectura por "rtl". Después, *Next* y *Finish*.
- Situarse en la ventana del editor y comprobar que el fichero tiene el aspecto que esperamos.
- Sustituir la cabecera por defecto por una que contenga al menos: nombre del bloque descrito en el fichero, nombre del proyecto, descripción breve y nombres de los autores (se puede tomar como ejemplo las cabeceras de los otros ficheros entregados, añadiendo los autores).

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

- Bien con el editor de ISE o con el editor de texto que uno prefiera, procederemos a crear el fichero VHDL del divisor de frecuencia (*freq_divider*).

  En el caso de usar el editor de ISE, haremos *Project / New Source* y seleccionaremos *VHDL Module*, nombre "freq_divider" y *Location* nuestro directorio "rtl". Se recomienda esta vez no rellenar los puertos en la tabla que proporciona ISE, sino hacerlo "a mano" en el fichero fuente para practicar un poco más la escritura de VHDL. Es más, se recomienda que nada más abrir el fichero en el editor de ISE se borre por completo su contenido y se empiece de cero.

  En el caso de usar otro editor, podremos añadir posteriormente el fichero al proyecto con *Project / Add Source*. Los bloques del proyecto pueden verse en el panel (solapa) *Design / Hierarchy* de la izquierda, y una lista de los ficheros incluidos en el proyecto en la solapa *Files*. (Los ficheros VHDL aparecerán correctamente en la vista de la solapa *Design* solamente cuando tengan definida una arquitectura).
- Para implementar el divisor de frecuencia crearemos un proceso que haga una cuenta de 0 a 499999 (este valor deberá definirse como una constante VHDL de tipo *integer*). Para ello usaremos una *signal* "count" 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 el reset síncrono; y c) cuando al llegar el flanco de reloj la cuenta actual sea 499999. 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 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* de *std_logic_unsigned*.
- A continuación añadiremos en el mismo proceso la generación de la señal de salida *EnableOut* (un ciclo a '1' cada 500000), 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.

Terminar e instanciar el componente *counter*:

- Añadir al proyecto los ficheros "counter.vhd" y "stopwatch_pkg" (del directorio "rtl/"). Tras ello podrán verse en la solapa *Files* de la parte izquierda. Hacer doble click sobre estos ficheros para poder ver su contenido en la ventana de edición. Se trata de un contador cuyo valor máximo de cuenta es parametrizable y de un *package* de utilidad. Observar el uso 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 hacer "wrap-around") cuando lo permita la señal de "enable" de entrada.
- Añadir la declaración del componente de este contador y sus cuatro instancias a "stopwatch.vhd". [Nota: ISE proporciona una ayuda para hacer instanciaciones, seleccionando el bloque en la solapa *Design* y ejecutando en *Processes* la funcionalidad *Design Utilities / View HDL Instantiation Template*. Sin embargo, esta utilidad no funciona bien en casos como éste en que tenemos un *generic*, una función en el tamaño de un puerto, etc., por lo que si la usamos habremos de corregir el código generado].
- Conectar los puertos de las cuatro 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 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*).

Comprensión de los ficheros que se entregan completos:

- Añadir al proyecto y examinar los ficheros "reset_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 "reset_adapt" puede obviarse el hacer un *generic map*, utilizando el valor por defecto del *generic*.

Completar la máquina de estados:

- Añadir al proyecto y completar el fichero "stopwatch_fsm.vhd", considerando la descripción del diseño de 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".

Completar el bloque de control de los displays 7-segmentos

- Añadir al proyecto el fichero "display_ctrl.vhd". Comprender el código entregado. Para completarlo basta hacer dos cosas:
  - Los ánodos de los cuatro displays son atacados por el puerto de 4 bits *Anodes*. Para mover este bus de salida se usa el bus interno *anodesInt*. Este bus contiene un único '0' que será el que active el display correspondiente en cada momento. Cada vez que se termina una "cuenta de refresco" el bus *anodesInt* debe rotar su cero, por ejemplo hacia la izquierda. Añadir el código que falta para hacer esta rotación del contenido de *anodesInt*.
  - Añadir la porción de código dedicada a la codificación 7-segmentos. Para ello, situar el cursor en la zona del fichero dedicada a esta funcionalidad y usar la función *Language Templates*, haciendo clic en el botón con un dibujo de una bombilla. El código que necesitamos está en *VHDL / Synthesis Constructs / Coding Examples / Misc*. Pinchando con el ratón en la línea correspondiente, podemos hacer con el botón derecho *Use in File*, que insertará la porción de código en el fichero donde estábamos trabajando, donde tuviéramos el cursor. Obsérvese que la codificación es activa a nivel bajo, la que necesitamos.
- Añadir la declaración e instanciación de "display_ctrl" a "stopwatch.vhd".

Finalizar el RTL

- Terminar de "cablear" el fichero "stopwatch.vhd" si quedaba algo pendiente.
- Comprobar que el chequeo sintáctico es correcto (obsérvese cómo compila toda la jerarquía del diseño seleccionando el top-level en la solapa *Design* y ejecutando el proceso *Synthesize – XST / Check Syntax*). 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" y de valor por defecto *FALSE*.
  - Usar una asignación condicional para sustituir el cable a la salida de *freq_divider* por un '1' cuando "FAST_SIMULATION" sea *TRUE*.
  - Declarar una constante "REFRESH_NCYCLES_SYN" que usaremos para asignar valor al generic de "display_ctrl" en condiciones normales (de *SYNthesis*). Inicializarla a 100000.
  - Describir, también en la parte declarativa de la arquitectura de "stopwatch.vhd", una función "set_refresh" que devuelva 4 si su único argumento, constante y de tipo boolean, es *TRUE* y que devuelva la constante anterior en caso contrario.
  - Declarar una constante "REFRESH_NCYCLES", inicializándola llamando a la función anterior con argumento "FAST_SIMULATION". Ésta será la constante que usemos como valor del generic de "display_ctrl".

Con esto conseguimos dos cosas: a) Eliminamos la división de frecuencia por 500000, haciendo que el contador de centésimas de segundo avance al ritmo del reloj. b) Aceleramos también la cuenta de refresco del display, haciendo que cada 4 ciclos se active un dígito diferente en vez de cada 100000 ciclos. Estas dos cosas nos permitirán, al principio del desarrollo, observar en simulación si el circuito está funcionando bien sin tener que esperar unos tiempos de simulación muy largos.

Preparar el testbench para la simulación:

- Crear un nuevo fichero fuente seleccionando *Project / New Source*.
- Esta vez seleccionar a la izquierda "VHDL Test Bench" y a la derecha en *Location* el directorio "sim". Como nombre para el fichero usar "stopwatch_tb".
- Al dar a *Next* nos permite seleccionar para qué diseño es el testbench. Seleccionar "stopwatch".
- Editar el código fuente del testbench:
  - Sustituir la declaración de componente que se genera automáticamente por una declaración correcta del componente *stopwatch*.
  - Preparar la instancia de nombre "uut" de este componente, usando un valor *TRUE* en el *generic map* (este valor que acelera la simulación lo usaremos hasta que hayamos depurado satisfactoriamente el código). Asegurarse de tener declarada una señal por cada puerto de *stopwatch* (nota: si uno no tiene muy clara la sintaxis y el funcionamiento de los "port maps" puede ser una buena idea por ejemplo usar para la señal el mismo nombre que el puerto 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 20 ns.
  - Declarar una señal "finSimulacion" de tipo *boolean*, inicializada a FALSE. La pondremos a TRUE para terminar la simulación.
  - Crear un proceso que genere en la señal *clk* un reloj de periodo CLK_PERIOD hasta que *finSimulacion* se haga TRUE, momento en que el proceso deberá quedar suspendido para siempre.
  - Sustituir el proceso "Stimulus process" que genera ISE por otro que haga lo siguiente:
    - Inicializar al principio todas las entradas a "stopwatch"
    - Crear una fase de reset inicial de duración 5 periodos de reloj.
    - Hacer un "pulsado" del botón de Start/Stop de duración 1047 ns.
    - Esperar un tiempo definido en una constante SIM_TIME, que inicialmente declararemos con un valor de "100 us" (ojo, 100 microsegundos, no nanosegundos). Tras esto activar el flag *finSimulacion* y matar el proceso con un *wait*.

Simular el diseño funcionalmente:

- En general se recomienda usar *ModelSim* como herramienta independiente, simulando mediante el uso de un proyecto y/o scripts en esta herramienta. A continuación se indican los pasos si se prefiere de momento lanzar la simulación desde ISE:
- Cambiar la vista de la solapa *Design* (*View*, arriba) de *Implementation* a *Simulation*.
- Seleccionar el testbench en la parte de arriba y simular haciendo doble clic en el panel de procesos sobre *ModelSim Simulator / Simulate Behavioral Model*.
- Si *ModelSim* reporta algún error o warning, corregirlo y repetir el proceso. Los errores se pueden corregir en el editor del propio *ModelSim*, haciendo doble click sobre cada error, o cambiando de tarea al editor utilizado (por ejemplo el del ISE). Tras cambiar el fichero se puede reintentar sin salir de *ModelSim* ejecutando el último comando en la consola *Transcript*, clicando en ella y pulsando la flecha hacia arriba del teclado y <*Enter*>).
- Comprobar en la ventana de ondas de ModelSim que el circuito hace lo esperado. Por defecto ISE encargará a ModelSim que simule 1000 ns, por lo que nos convendrá decirle que corra más tiempo, mediante el botón correspondiente o escribiendo en la consola "run –all" (simular hasta que el simulador no tenga nada que hacer; también se puede usar "run <tiempo>", con <tiempo> incluyendo las unidades -ej.: "run 100us"-).

Asignar los pines de la FPGA:

- Añadir al proyecto el fichero "stopwatch.ucf", del directorio "ise".
- En la solapa *Design* volver a cambiar la vista a *Implementation*. Podremos ver ahora, bajo el diseño "stopwatch", una línea con el fichero "stopwatch.ucf". Pinchar en ella con un solo clic del ratón (un doble clic dispara todo el proceso de implementación).
- Desplegar en la ventana de *Processes* la línea *User Constraints* y hacer doble clic sobre *Edit Constraints (Text)*. Aparecerá en el editor el fichero "stopwatch.ucf". Examinar las constraints de posicionado de pines. Falta la asignación de pin para la señal *Segments<2>* (segmento "c" del display). **Usando el manual o la serigrafía de la placa, completar el fichero de constraints con la asignación de este pin.**

Realizar la implementación

- Seleccionar en la jerarquía del diseño mostrada en la solapa *Design* la línea del bloque "stopwatch", inmediatamente debajo de la que pone el identificador del modelo de FPGA ("xc3s200-4ft256").
- Mostrar los subprocesos de implementación haciendo clic sobre el "+" a la izquierda de *Implement Design*.
- Dispararemos la implementación completa haciendo doble clic en el proceso *Generate Programming File*.
- Observar cómo evoluciona la implementación, en los iconos de la ventana *Processes* y en la consola. Al finalizar comprobar las solapas *Errors* y *Warnings* de la consola, haciendo las correcciones oportunas si es necesario.

- En la ventana principal seleccionar la solapa *Design Summary* y ya dentro de ella *Design Overview / Pinout Report*. Comprobar que en la columna *Signal Name* aparecen las señales que hemos usado, en los pines apropiados (en particular comprobar el pin que faltaba en el fichero UCF original).

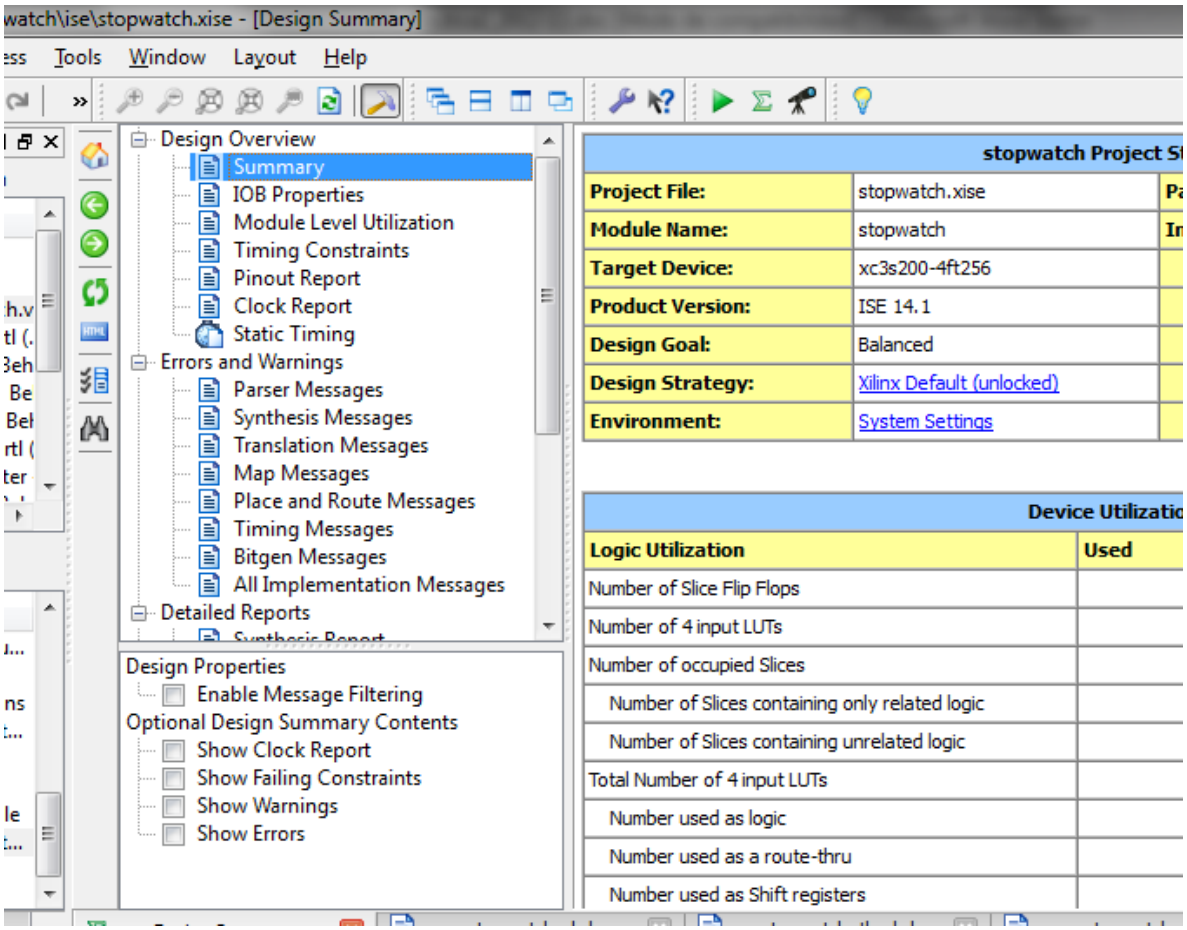
Bajar el diseño a la placa:

- Preparar la placa en el siguiente orden: 1) Conectar las tres piezas del adaptador JTAG 2) Conectar con suma atención el adaptador JTAG a la placa. 3) Alimentar la placa. 4) Conectar el adaptador JTAG al ordenador.
- En la ventana *Processes*, hacer doble clic sobre *Configurate Target Device / Manage Configuration Project (iMPACT)*. Proceder a configurar la FPGA como en la primera práctica.
- Comprobar el funcionamiento de la placa.

Análisis de resultados de la implementación

Una vez comprobado el funcionamiento de nuestro circuito haremos un pequeño análisis de resultados de la implementación en la FPGA.

Seleccionar la solapa *Design Summary* (si la hemos cerrado anteriormente usar *Project* | *Design Summary/Reports*) y dentro de esta *Design Overview / Summary*.



Echar un vistazo a las cifras del *Device Utilization Summary* y apuntar las respuestas a las siguientes preguntas:

- ⇨ Cuántas LUTs ha usado el diseño? ¿Porcentualmente respecto al total disponible?
- ⇨ ¿Cuántos Flip-Flops? ¿Porcentualmente respecto al total disponible?
- ⇨ ¿Cuántas LUTs se han utilizado exclusivamente como recurso de rutado?
- ⇨ ¿Cuántos IOBs se han usado?
- ⇨ ¿Cuántos drivers de reloj (BUFGMUXs) 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 (considerando las LUTs realmente usadas para lógica, no las usadas para rutado). Los IOBs son los bloques de I/O; podemos comprobar que hay uno por cada entrada o salida de nuestro diseño.

En la lista de la izquierda del *Design Summary*, seleccionar que se visualice el reporte de síntesis haciendo clic en *Detailed Reports* / *Synthesis Reports*. Buscar la sección "Timing Summary" (hacia el final, se puede buscar con *Ctrl-F*).

- ⇨ ¿Qué frecuencia máxima estima el sintetizador que podrá soportar nuestro diseño?

Abrir el reporte de *Place and Route*, también en la sección *Detailed Reports*. Buscar la sección de información sobre el reloj y el timing ("Generating Clock Report").

- ⇨ ¿Qué skew tiene la *net* de reloj?
- ⇨ ¿Cuál es el Best Case Achievable para el periodo de reloj? ¿Y la frecuencia correspondiente?

Comprobar que estos mismos datos aparecen también en las secciones del *Summary* llamadas "*Design Overview / Clock Report"* y "*Design Overview / Timing Constraints*", ya que la mejor estimación es la obtenida tras el Place and Route.

Razonar una respuesta para las siguientes preguntas:

- ⇨ ¿Qué se observa en la placa al pulsar el botón de reset asíncrono? ¿Y al pulsar el botón "SetToZero"? ¿Por qué esta diferencia?
- ⇨ Si hemos visto la simulación "acelerada", ¿por qué en la placa vemos moverse el cronómetro a la velocidad normal?
- ⇨ ¿Qué "path" combinacional observas en el diseño que recorra varios sub-bloques (no hace falta buscar en reportes de la herramienta)?

Entrega de la práctica

El correcto funcionamiento del diseño sobre la placa "S3BOARD" deberá mostrarse al profesor de prácticas en clase antes de la fecha límite especificada en la web del laboratorio.

Por otra parte, **deberá entregarse por web un archivo zip con el diseño realizado**, antes de la fecha límite especificada en la web del laboratorio. Entregar el directorio "P2_stopwatch" 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 "ise" si se ha seguido el flujo ISE habitual (o en el directorio "sim" si se ha usado *ModelSim* directamente sobre este directorio).

Todos los ficheros del código fuente deberán entregarse con una cabecera adecuada, incluyendo siempre el nombre de los autores (modificar con este fin la cabecera de los ficheros entregados).

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>_P2.zip
```

y la siguiente para los alumnos de DCSE:

```
<{DCSE_L | DCSE_XA | DCSE_XB}><numero_pareja_2digitos>_P2.zip
```

(Ejs.: `DCSE_XB01_P2.zip` para pareja 1 de DCSE-b miércoles, `DIE_J02_P2.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.