Apellidos, Nombre: _____

Diseño de circuitos digitales con dispositivos configurables FPGA

Estructura de Computadores
Ingeniería Informática. Tecnologías Informáticas
Dpto. de Tecnología Electrónica

1 Material

- Ordenador con entorno Xilinx ISE instalado.
- Placa de desarrollo Digilent Basys2 y documentación.
- Archivos iniciales de diseño (kit de laboratorio).

Los archivos y documentos necesarios pueden descargarse desde la web de la asignatura.

2 Descripción y objetivos

El objetivo general de esta práctica es aprender el proceso de diseño e implementación en chips configurables FPGA de circuitos digitales descritos mediante lenguajes de descripción de *hardware* (Verilog).

La práctica se desarrolla en dos fases:

- Fase 1: diseño e implementación de un contador binario módulo 16 con salida binaria y con entradas de cuenta (*up*) y puesta a cero (*clear*) (Figura 1a).
- Fase 2: diseño e implementación de un contador módulo 16 con salida de 7 segmentos. En esta fase se combina el contador anterior y un convertidor de código binario a 7 segmentos (Figura 1b).

El sistema se implementa sobre una placa de desarrollo que tiene un chip FPGA donde puede configurarse el diseño del circuito digital, y varios periféricos que permiten controlar las señales de entrada al sistema (mediante interruptores y botones) y visualizar el valor de las señales de salida (mediante LEDs y visores de 7 segmentos).

2.1 Resultados del aprendizaje

- Uso básico del lenguaje Verilog mediante la modificación y combinación de módulos previamente elaborados.

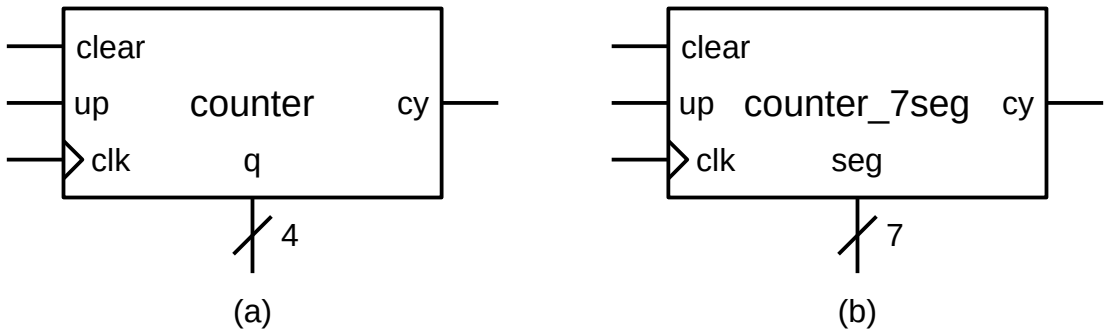

Figura 1: Contadores: (a) contador binario, (b) contador 7 segmentos.

- Simular y depurar diseños simples en Verilog.
- Configurar las conexiones de señales y periféricos en una placa de desarrollo FPGA mediante archivos de restricciones.
- Ejecutar los procesos de síntesis e implementación de circuitos digitales sobre FPGA y comprender a grandes rasgos en qué consisten estos procesos.
- Programar dispositivos FPGA y comprobar la operación del circuito implementado.

3 Trabajo previo

1. Familiarizarse con la placa de desarrollo Basys2. Leer la documentación del fabricante.

2. Completar el diseño del módulo **counter** y simularlo mediante el banco de pruebas suministrado hasta que su operación sea correcta (archivos `counter.v` y `counter_tb.v`).

3. Familiarizarse con el diseño del convertidor de código binario a 7 segmentos. Opcionalmente simular el convertidor con el banco de pruebas suministrado (archivos `sevenseg.v` y `sevenseg_tb.v`).

4. Completar el diseño del sistema completo consistente en un contador módulo 16 con salida en código 7 segmentos construido a partir de los módulos **counter** y **sevenseg** (archivo `counter_7seg.v`). Opcionalmente, hacer un banco de pruebas para el sistema completo y simularlo para comprobar su correcta operación. Se recomienda usar el banco de pruebas del contador en el archivo `counter_tb.v` como punto de partida.

4 Trabajo en laboratorio

El trabajo de laboratorio se realiza en dos fases:

- Implementación del contador de 16 bits. Control de la señal de reloj del contador mediante un botón y visualización del estado de cuenta y señal de fin de cuenta mediante LEDs.
- Construcción del sistema completo mediante la conexión del contador y el convertidor de código de binario a 7 segmentos; y visualización del estado de cuenta en visor de 7 segmentos.

4.1 Implementación y prueba de un contador módulo 16

A continuación se describe en detalle el proceso de implementación del contador módulo 16 en la placa Basys2. Se parte del diseño en Verilog del contador y se realiza la implementación en la placa de forma que pueda observarse la operación del contador, mediante la conexión de las entradas y salidas del módulo contador a elementos adecuados de la placa: botones, LEDs, etc.

4.1.1 Creación de un proyecto en Xilinx ISE

- Descargue el archivo `counter_kit.zip` de la web y extraiga su contenido en una carpeta.
- Inicie el entorno Xilinx ISE. La ventana del entorno se divide en tres secciones: la zona izquierda es el panel del proyecto y sirve para gestionar los archivos y módulos del proyecto y las acciones que se pueden realizar sobre los mismos; la zona derecha es el panel principal que da acceso a la edición de los elementos del proyecto, como los archivos Verilog; la zona inferior es la consola, donde podemos ver los mensajes de error y avisos de las herramientas del entorno conforme se realizan las operaciones sobre el diseño.
- Es posible que al iniciar ISE se abra automáticamente el último proyecto editado. Si es así, cierre el proyecto desde *File → Close Project*.
- Cree un nuevo proyecto desde *File → New Project*. Se abrirá el asistente para creación de proyectos. Escriba un nombre para el proyecto como **counter**. Puede elegir la carpeta para el nuevo proyecto si lo desea, por ejemplo, la carpeta donde ha descargado los archivos iniciales de la práctica. Pulse *NEXT*.
- Ahora introduzca los detalles del tipo de chip FPGA en que se sintetizará el proyecto. Para la placa Basys2 los detalles son:
  - General Purpose
  - Family: Spartan3E
  - Device: XC3S100E
  - Package: CP132
  - Speed grade: -5

El resto de opciones del proyecto no deben modificarse: XST synthesis, ISIM simulator, Verilog preferred language, etc.). Pulse *NEXT*. Revise el resumen de opciones del proyecto y si no hay errores pulse *FINISH*.

4.1.2 Añadir archivos al proyecto

- Para añadir los archivos previamente elaborados al proyecto haga click derecho sobre el panel de la jerarquía del diseño (*Hierarchy*) y elige *Add Source*.
- Seleccione los archivos Verilog `counter.v` y `counter_tb.v` desde la carpeta donde fueron descargados. Puede elegir varios archivos a la vez manteniendo la tecla **Ctrl**. ISE usará estos archivos para implementar y/o simular el diseño. Puede elegir la finalidad de cada archivo manualmente, pero normalmente ISE detectará la función de cada archivo automáticamente y solo tendremos que confirmar.

- Ahora, el panel de diseño muestra los archivos que forman el proyecto. Hay dos vistas: implementación (*Implementation*) y simulación (*Simulation*). La vista de implementación muestra los archivos con el diseño que se configurará en la FPGA. La vista de simulación muestra, además, los archivos con los bancos de prueba que solo se emplean en la simulación.
- Haciendo doble click sobre cualquier archivo, este se abre en el panel central y puede ser editado. Si no lo ha hecho previamente, abra el archivo `counter.v` y complete el diseño.

4.1.3 Simulación del banco de pruebas

Para simular el módulo **counter** con el banco de pruebas suministrado haga lo siguiente:

- Seleccione la vista de simulación y seleccione el módulo **test** que se encuentra en el archivo `counter_tb.v`. Observe como el panel jerárquico organiza el diseño en función de los módulos que contiene, independientemente de en qué archivo se definen (aunque indica el nombre del archivo entre paréntesis).
- En el panel de procesos (*Processes*) aparecen las acciones que se pueden ejecutar sobre el módulo seleccionado en el panel de vistas del diseño. Al seleccionar el módulo **test** aparecen las acciones asociadas al simulador ISim. Aquí puede seleccionar comprobar la sintaxis del código Verilog (*Behavioral Check Syntax*) o directamente simular el módulo del banco de pruebas (*Simulate Behavioral Model*). Haga doble click sobre esta última opción. Si hay errores en el diseño tendrá que editar el código y volver a ejecutar la simulación.
- Si el código es correcto, tras unos segundos se abrirá la ventana del simulador ISim. En el panel principal verá el código Verilog simulado y una marca donde se ha detenido el simulador. Elija la pestaña con las formas de onda, busque los botones de selección de ampliación (zoom) y use el botón que muestra todo el tiempo simulado.Observe las señales y compruebe que el resultado es correcto.
- Por defecto, ISim simula un tiempo de 1000ns o bien hasta que encuentre la directiva Verilog *$finish*. Desde el menú *Simulation* o mediante los botones de control de la simulación sobre el panel principal puede continuar la simulación, detenerla o reiniciarla, pero no es necesario para esta práctica.
- El formato en que se muestran las señales puede modificarse desde el menú contextual que aparece al hacer click derecho sobre cualquier señal. Por ejemplo, pruebe a mostrar la señal de 4 bits *q* en formato decimal sin signo: *Radix → Unsigned Decimal*.
- Es muy útil guardar la configuración de la visualización de las señales para futuras simulaciones. Elija en el menú *File → Save As…* y guarde la configuración con un nombre significativo como `counter_tb.wcfg`.
- Salga del simulador. De vuelta en la ventana de ISE, se va a configurar el proceso de simulación para que cargue automáticamente la configuración de ondas que se ha salvado. Para ello, haga click derecho sobre el proceso *Simulate Behavioral Model* y elija *Process Properties…*. En la ventana que aparece, busque la opción *Custom Waveform Configuration File* y busque el archivo de configuración de ondas `counter_tb.wcfg`. Esto facilitará mucho ver los nuevos resultados de simulación cuando se vuelva a ejecutar el simulador.

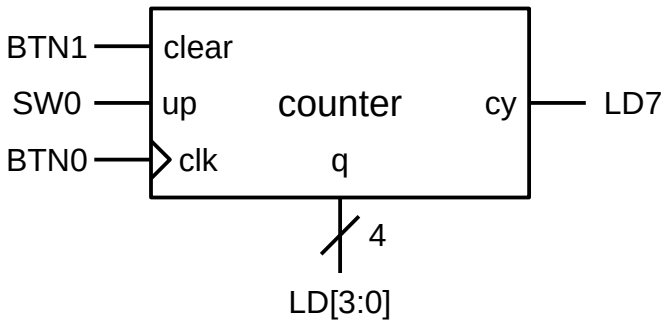

Figura 2: Conexión del contador binario con las señales de la placa Basys2.

4.1.4 Añadir un archivo de restricciones

El chip FPGA incluido en la placa Basys2 tiene sus pines conectados a los diversos conectores y periféricos presentes en la placa: interruptores, botones, LEDs, visores de 7 segmentos, etc. Para que el contador diseñado pueda se implementado y probado en la placa, las señales de entrada y salida del contador deben asignarse a pines concretos de la FPGA de modo que las señales del contador acaben conectadas a los periféricos adecuados. Por ejemplo, queremos poder controlar la señal de cuenta del contador mediante uno de los interruptores de la placa, y observar la señal de fin de cuenta conectándola a uno de los LEDs, que se encenderá cuando la señal valga 1.

Esto se consigue mediante un archivo de restricciones (UCF –*User Constrains File*–) para proceso de síntesis que fija algunos parámetros de la implementación, como qué señal acaba en qué pin de la FPGA. Todo proyecto debe contener un archivo de restricciones, de lo contrario, la herramienta de síntesis hará una asignación arbitraria de los pines y no podremos controlar las entradas ni ver las salidas (a menos que tengamos mucha suerte).

En nuestro contador queremos poder controlar la señal de reloj (clk) y de puesta a cero (clear) mediante botones, la señal de cuenta (up) mediante un interruptor; y observar la salida binaria del contador (q) y la señal de fin de cuenta (cy) mediante LEDs. Establecer estas restricciones es sencillo ya que el fabricante de la placa proporciona un archivo UCF plantilla con la lista de todos los pines de la FPGA y los periféricos a los que están asignados. Los periféricos están identificados mediante nombres de la conexiones físicas de en la placa fáciles de reconocer. Por ejemplo, los LEDs se conectan a las señales LD0, LD1, etc.; los interruptores a las señales SW0, SW1, etc. y así sucesivamente. Los nombres de estas señales también se encuentran escritos sobre la propia placa. El fabricante de la placa da información detallada sobre los periféricos que contiene y su operación.

La asignación de señales del contador a las conexiones de la placa se muestra en la Figura 2. Para trasladar esta información al diseño mediante un archivo UCF hay que hacer lo siguiente:

- Añada el archivo UCF plantilla al diseño como hizo con los archivos Verilog. En este caso use la opción *Add Copy of Source…*. Esto creará una nueva copia del archivo en la carpeta del proyecto y dejará intacto el archivo original, que podrá ser usado para otros proyectos. En el panel del diseño, el archivo UCF solo aparece en la vista de implementación, ya que este archivo no contiene información relevante para la simulación.
- Abra el archivo UCF haciendo doble click sobre él en el panel de diseño. Verá que el archivo tiene líneas para configurar cada periférico, pero están todas comentadas. Para

asignar una señal a un periférico, descomentaremos la línea correspondiente y sustituiremos la señal indicada tras la palabra “NET” por la señal que queremos conectar de nuestro módulo. Por ejemplo, para conectar la señal *clear* del contador al botón 1 (BTN1) localizamos la línea correspondiente al botón 1:

```
#NET "btn<1>" LOC = "C11"; # Bank = 2, Signal name = BTN1
```

y cambiamos la línea por:

```
NET "clear" LOC = "C11"; # Bank = 2, Signal name = BTN1
```

- Repita lo mismo para el resto de señales tomando la Figura 2 como referencia.
- Salve el archivo UCF.

4.1.5 Síntesis e implementación

“Síntesis” es el proceso que convierte el código Verilog en un circuito digital que realiza la operación descrita por dicho código. Es un proceso complejo similar el proceso manual consistente en obtener la tabla de verdad de las funciones, reducir las funciones (K-mapa), seleccionar las puertas lógicas adecuadas, etc. La “implementación” consiste en varios pasos en los que la funcionalidad del circuito obtenido por la síntesis se “mapea” en los bloques lógico configurables (CLBs) que componen la FPGA y se establecen las rutas de conexión de estos bloques para obtener un circuito funcional. El circuito generado por la síntesis es un circuito compuesto de componentes estándar (puertas lógicas y biestable) independiente del tipo de FPGA usado. El proceso de implementación depende del chip de FPGA disponible y de los elementos que contenga. Es durante el proceso de implementación cuando se tienen en cuenta las restricciones para la conexión de señales y pines descrita anteriormente.

El proceso de implementación finaliza generando el conjunto de bits (*bitstream*) de configuración de los CLBs y matrices de interconexión de la FPGA. Cuando estos bits sean cargados en los bloques correspondientes, la FPGA se comportará como el circuito diseñado.

Por suerte, las herramientas son capaces de realizar los procesos de síntesis e implementación de forma completamente automática, aunque es posible modificar parámetros para obtener diferentes resultados: optimizar velocidad, consumo de potencia, etc.

- Seleccione el módulo a implementar en el panel de diseño (módulo **counter**). En el panel de procesos aparecen las tres acciones principales del proceso completo de síntesis: *Synthesize-XST*, *Implement Design* y *Generate Programming File*.
- Haga click derecho sobre *Generate Programming File* y seleccione *Run*. Esto generará el archivo de configuración de la FPGA, pero también ejecutará todos los procesos anteriores que sean necesarios. Observe el icono animado que indica la ejecución de cada proceso.

Si todo va bien, aparecerá una marca verde o amarilla junto a cada proceso. Si algún proceso falla, aparecerá una marca roja y se detendrá la síntesis. En este caso, puede consultar los errores en la consola inferior, en la pestaña *Errors*. Mire si hay errores, lea los mensajes de error e intente interpretarlos. Debe aparecer al menos un error que empieza como:

```
ERROR:Place:1018 - A clock IOB / clock component pair have been
found that are not placed at an optimal clock IOB / clock site
pair.
[...]
These examples can be used directly in the .ucf file to override
this clock rule. < NET "clk" CLOCK_DEDICATED_ROUTE = FALSE; >
```

Este error se debe al uso que hemos hecho de la señal de reloj *clk* en nuestro diseño. En general, las señales de reloj deben suministrarse al sistema por pines (IOB) especiales para señales de reloj, pero nosotros hemos conectado esta señal a un botón de la placa (BTN0) que está asignado a un IOB convencional. Como esto es en general un error de diseño, la herramienta se detiene en la fase de implementación. Este diseño es muy simple y no hay problema en tener el reloj en un pin convencional. Se puede indicar a la herramienta que ignore esta restricción en el archivo UCF, tal como indica el propio mensaje de error.

- Edite el archivo UCF y añada una regla que indique que la señal *clk* no tiene que usar una ruta de reloj especial. Para ello copie la siguiente línea tras la línea de asignación de la señal *clk*:

```
NET "clk" LOC = "G12"; # Bank = 0, Signal name = BTN0 +
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
```

Cuando se ejecuta algún proceso en ISE que depende de procesos anteriores, ISE comprueba si los procesos anteriores están actualizados y solo ejecutará aquellos procesos necesarios hasta ejecutar la acción solicitada. Por ejemplo, si ya se ha generado un archivo de programación (*bitstream*), habrá que cambiar el archivo UCF y volver a solicitar la generación de un nuevo archivo de programación, se ejecutará el proceso de implementación pero no el de síntesis, ya que el archivo UCF no afecta a la síntesis sino a la implementación. Si se cambia alguno de los archivos Verilog del proyecto, se volverá a sintetizar e implementar aquellos módulos que hayan sido modificados, etc.

Si en algún momento se desea ejecutar el proceso completo descartando resultados anteriores habrá que elegir la opción *Rerun All* en el menú contextual del proceso a ejecutar. Si se quiere eliminar todos los datos y archivos temporales generados por el entorno, habrá que hacerlo desde el menú *Project→Cleanup Project Files…* Muy útil cuando se vaya a dejar de trabajar en un proyecto para ahorrar espacio.

4.1.6 Programar el diseño en la FPGA

Como resultados de los procesos anteriores, se ha generado un archivo con la configuración (bitstream) que hay que cargar en el dispositivo FPGA. Esta configuración está en el archivo `counter.bit` guardado en la carpeta del proyecto. El proceso de programación consiste simplemente en cargar este paquete de bits en la FPGA. El proceso es sencillo, pero existen varias alternativas para hacerlo, dependiendo del sistema operativo donde estemos usando ISE. En MS-Windows se dispone de la herramienta independiente *Adept* suministrada por el fabricante de la placa Basys2.

Con *Adept* se obtendrá el diseño programado en la placa y se podrá probar. En caso de que la operación no sea la correcta hay que comprobar todo el proceso de diseño:

- Revisar si las herramientas han dado errores o avisos en algún momento del proceso.
- Comprobar que la asignación de señales a pines de la placa son correctos en el archivo UCF.
- Comprobar si los resultados de la simulación son correctos. Si no se ha hecho un banco de pruebas y simulado previamente el diseño, quizá sea hora de hacerlo.

A continuación de describe el procedimientos de programación.

Programación empleando la herramienta gráfica Adept

Esta herramienta está disponible para sistemas MS-Windows. El procedimiento es el siguiente:

- Asegúrese de que la placa está conectada a un puerto USB y está encendida.
- Ejecute la herramienta *Digilent Adept*. Debe aparecer una ventana con el nombre de la placa (Basys2) y se deben mostrar dos entradas para cargar archivos de programación para los dos dispositivos que tiene la placa: FPGA y PROM.
- En la pestaña *Config* introduzca el nombre del archivo a configurar en la entrada que corresponde al dispositivo FPGA, buscando el archivo en el sistema de archivos.
- Pulse en *Program* junto a la entrada de la FPGA. Es posible que aparezca una ventana de aviso advirtiendo que el *bitstream* está configurado para usar CCLK en vez de JTAG CLK como reloj de inicio de la FPGA. Puede continuar e ignorar el aviso, y consultar la nota de diseño 1 al final de este manual para una explicación sobre este aviso.
- Uno de los LEDs de la FPGA parpadeará unos segundos. Luego el dispositivo estará configurado.
- Cierre la ventana de *Digilent Adept*.

Comprobación del diseño tras la programación

- Pruebe el diseño: conecte el interruptor SW0 para activar la señal de cuenta *up* y pulsa el botón BTN0. Debe ver que se encienden varios LEDs mostrando la representación en binario del número de cuenta del contador. Al llegar a 15 (1111) se debe activar el LED LD7, que indica el fin de cuenta. ¡Enhorabuena!
- Pruebe también la señal de *clear* conectada al botón 1 (BTN1). Al pulsar el botón el contador debe volver al valor 0. ¿Es así? ¿El clear es síncrono o asíncrono? Mire el diseño Verilog si no se acuerda.

4.2 Contador módulo 16 con salida 7 segmentos

Para diseñar el sistema contador con salida en código de 7 segmentos se emplea el contador binario y un convertidor binario a 7 segmentos conectados como aparece en la Figura 3. Observe como la figura refleja la estructura del código Verilog del diseño, que consiste en tres módulos diferentes:

- El módulo **counter** diseñado previamente.
- El módulo **sevenseg** con el convertidor de binario a 7 segmentos.
- El módulo **counter_7seg** que se forma mediante la conexión del contador con el convertidor.

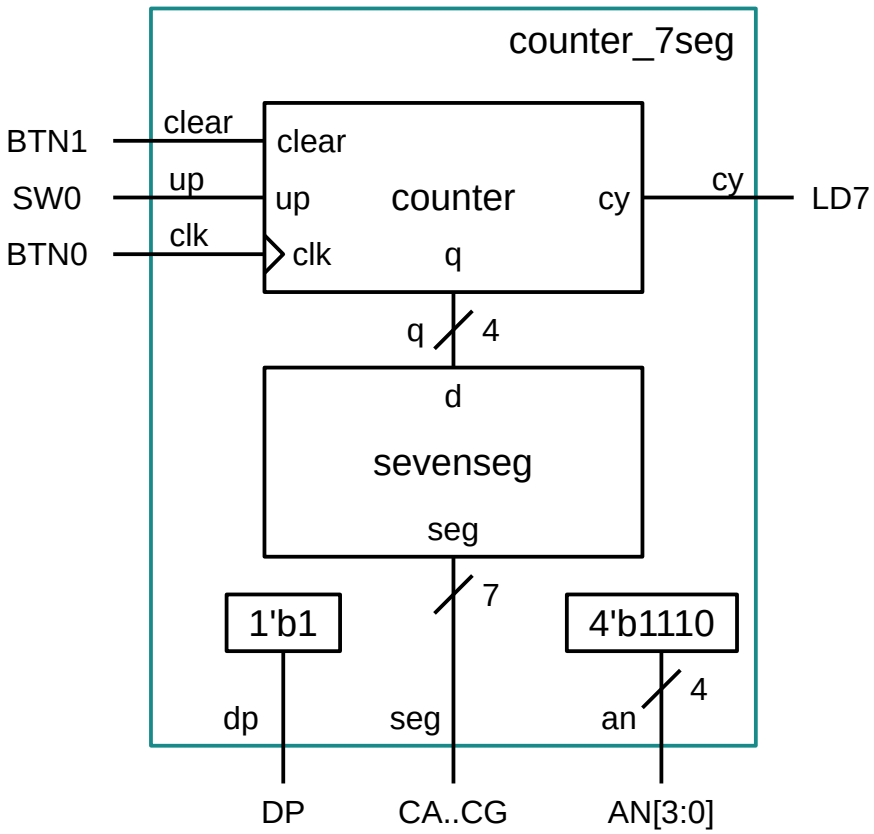

Figura 3: Conexión contador módulo 16.

La salida de 7 segmentos puede visualizarse en la placa Basys2 ya que ésta posee un visor de 7 segmentos de 4 cifras. Éste visor funciona de la siguiente forma: las señales de 7 segmentos (CA, CB, … CG) son comunes para las cuatro cifras del visor, pero podemos activar solo las cifras que deseemos mediante las señales AN3, AN2, AN1 y AN0. Cada cifra tiene además un punto decimal que se controla con una única señal DP para todas las cifras. El módulo **counter_7seg** emplea la señal *seg[0:6]* para la salida de 7 segmentos, *dp* para el punto decimal y *an[3:0]* para las señales de activación de las cifras. Estos nombres coinciden con los empleados en al archivo UCF del fabricante.

Tanto los segmentos como el punto decimal y las señales de activación de las cifras son activos en nivel bajo, esto es, se activan cuando la señal correspondiente vale 0. Para nuestro prototipo, se mantendrá apagado el punto decimal y solo se activará la cifra 0 del visor. Esto se consigue asignando las constantes adecuadas a las señales *dp* y *an* tal como se indica en la figura.

El proceso de síntesis e implementación del contador con salida 7 segmentos es idéntico al seguido para el contador binario.

4.2.1 Creación del proyecto y archivo UCF

- Cree un nuevo proyecto en ISE para el contador con salida de 7 segmentos. Puede usar el nombre **counter_7seg** y crear la carpeta del proyecto junto a los archivos base del diseño. Use los mismos datos de tipo de FPGA que en el apartado anterior.
- Añada los archivos Verilog al proyecto. Use la opción *Add Source* para usar directamente los archivos base. Añada los archivos que contienen todos los módulos que componen el

proyecto: `counter.v`, `sevenseg.v` y `counter_7seg.v`. Habrá observado que el nombre de los archivos es igual que el del módulo principal que contienen. Este es un criterio habitual en diseños Verilog.

- Debe completar el diseño del módulo de nivel superior `counter_7seg.v` si no lo ha hecho anteriormente. Partiendo de la plantilla en el archivo base, basta con colocar instancias del contador binario y del convertidor de 7 segmentos y conectarlos entre si y a las señales de entrada y salida tal como indica la Figura 3.
- Añada al proyecto una copia del archivo UCF base usando la opción *Add Copy of Source…*.
- Modifique el archivo UCF igual que se hizo en el apartado anterior para realizar las conexiones de las señales del circuito con los pines de la placa tal y como se muestra en la Figura 3. Observe que la selección de bits en el archivo UCF se hace con los marcadores "<>" y no con "[]" como en Verilog. Por ejemplo, para conectar el bit 3 de la señal *an* al pin AN3 la línea de restricción es:

```
NET "an<3>" LOC = "K14"; # Bank = 1, Signal name = AN3
```

4.2.2 Simulación del proyecto (opcional)

En este punto se podría escribir un banco de pruebas para el módulo **counter_7seg** para comprobar que el diseño es correcto. No obstante, como el diseño es simple y tanto el contador binario como el convertidor 7 segmentos son módulos ya probados, se seguirá el proceso de diseño sin hacer simulación. Más adelante, si falla el circuito ya implementado, se podrá hacer un banco de pruebas para intentar localizar el fallo.

En cualquier caso, sí se debe usar el proceso *Behavioral Check Syntax* sobre todos los módulos del diseño para comprobar que no se han cometido errores de sintaxis.

4.2.3 Implementación y test

Implemente el proyecto en la placa siguiendo el mismo procedimiento que en el apartado anterior. Si todo ha ido bien, observará que ahora el valor de cuenta del contador se muestra en la cifra de la derecha del visor de 7 segmentos. ¡Enhorabuena! Pruebe el funcionamiento de la señal de *clear* y de la señal de fin de cuenta.

¿Hay algún problema? Observará que el visor de 7 segmentos muestra los valores de cuenta hasta el número 9. Luego muestra un guión (-) aunque sabemos cuando el contador ha llegado a su valor máximo (15) porque se activará la señal de fin de cuenta encendiendo el LED 7 (LD7). El problema es que nuestro convertidor de binario a 7 segmentos solo convierte cifras decimales: del 0 al 9. Cualquier otro valor de entrada produce que se active solo el segmento central. Revise el código del convertidor para ver que esta es la funcionalidad implementada. Lo resolveremos en el siguiente apartado.

4.2.4 Convertidor hexadecimal

Modifique el diseño del convertidor binario a 7 segmentos para que convierta números del 0 al 15 y los represente con una sola cifra hexadecimal. Debe pensar cómo construir las cifras desde la A a la F mediante 7 segmentos (puede usar letras minúsculas) y ampliar el diseño del convertidor actual con los códigos correspondientes.

Reinicie el proceso de síntesis, programe el nuevo *bitstream* y compruebe el funcionamiento. Compruebe con cuidado que no ha representado dos cifras mediante el mismo símbolo. Si así fuera, corrija el diseño.

5 Modificaciones y mejoras del diseño (opcional)

El sistema actual no cumple una función bien definida, pero es un buen punto de partida para diseñar un sistema más complejo o un módulo que podamos reutilizar en otros proyectos: un contador de los objetos que pasan por una cinta, un temporizador para abrir o cerrar una puerta, etc. A menudo los sistemas digitales complejos se diseñan así: se parte de una funcionalidad básica y luego se va mejorando el diseño hasta que realice todas las operaciones que se necesitan.

En los siguiente subapartados se proponen algunas modificaciones opcionales del diseño que pueden hacerlo más útil y divertido. Todas las modificaciones propuestas son compatibles entre si.

5.1 Cronómetro muy simple

Controlar la señal *clk* con un botón ha estado bien para hacer pruebas, pero en los diseños prácticos la señal de reloj es una señal periódica que cambia constantemente a una determinada frecuencia. La placa Basys2 incluye una señal de reloj a 50MHz en el pin MCLK. Simplemente cambiando el archivo UCF podemos asignar la entrada de reloj *clk* al reloj generado por la placa, pero si hacemos esto el contador se incrementará 50 millones de veces por segundo y no será demasiado útil. Necesitamos hacer que el contador no incremente su estado en todos los ciclos, sino cada varios ciclos.

Si hacemos que nuestro contador se incremente automáticamente cada segundo, tendremos un contador de segundos que podremos activar o detener con la señal *up* y podremos poner a cero con la señal *clear*. ¡Tendremos un cronómetro de segundos!

Para “reducir” la frecuencia del reloj o, mejor dicho, para hacer cosas a menor frecuencia de la que marca el reloj del sistema se utilizan los circuitos pre-escaladores o *prescalers*. Los *prescalers* son básicamente contadores que activan salidas de habilitación cada cierto número de ciclos de reloj. Estas salidas, a su vez, sirven para activar otros circuitos solo cada cierto tiempo.

Las modificaciones a realizar para hacer nuestro cronómetro serían las siguientes. Úselas como guía y pregunte las dudas al profesor:

- Modifique el diseño del módulo **counter** para que incluya un *prescaler*. El *prescaler* se puede hacer con otro proceso *always* que incremente un contador. Cuando el contador llega a un valor máximo, se activa una señal *enable* y se reinicia a cero. Tiene que calcular cuál es el valor máximo para que la señal se active cada segundo y el número de bits del contador del prescaler para que pueda llegar hasta ese valor, sabiendo que la frecuencia es 50MHz.

- Modifique el procedimiento del contador normal para que solo incremente cuando se activa la señal *up* y además esté activa la señal *enable*.
- Cambie el archivo UCF para conectar la señal *clk* a la señal de reloj del sistema MCLK. La restricción “CLOCK_DEDICATED_ROUTE = FALSE” que habíamos impuesto a la señal de reloj puede eliminarse (o comentarse) porque la señal MCLK de la placa está asociada a un IOB de reloj de la FPGA.
- Implemente y pruebe el diseño. ¡Suerte!

5.2 Contador decimal

Antes hemos modificado el convertidor para que represente hasta el número 15 (en hexadecimal). Otra opción interesante sería modificar el contador para que cuente de 0 a 9 (módulo 10). De esta forma podría ser la base para un contador decimal de varias cifras, un cronómetro o un reloj digital.

Modifique el código Verilog del contador para que cuente de 0 a 9. Tenga en cuenta la señal de fin de cuenta. Pruebe el nuevo diseño.

5.3 Contador ascendente/descendente (reversible)

Añada una señal *down* al contador. Cuando se activa *down* el contador hará una cuenta descendente. Las señales *up* y *down* no deberían activarse a la vez. Tendrá que decidir que hará el contador si esto ocurre. Tenga en cuenta que cuando la cuenta sea descendente, la señal de fin de cuenta debe activarse en el valor 0 y no en 15. Realice los cambios necesarios y pruebe el nuevo diseño.

6 Notas de diseño

6.1 CCLK y JTAG CLK

Tras el proceso de configuración de la FPGA, el chip necesita hacer un proceso de inicio para comenzar a operar como el circuito configurado. Este proceso de inicio necesita una señal de reloj. El propio chip FPGA genera una señal de reloj interna CCLK. A su vez, el interfaz de configuración que conecta con el ordenador, denominado JTAG, provee una señal de reloj externa la FPGA, JTAG CLK.

Entre muchos otros parámetros, el bitstream que contiene la configuración de la FPGA indica si hay que usar CCLK o JTAG CLK para el proceso de iniciación del chip. Cuanto la configuración se carga directamente desde el ordenador en la FPGA debe usarse la configuración JTAG CLK, ya que esta señal está presente ya que la interfaz de configuración está activada, mientras que la señal interna CCLK podría no estar activa. En cambio, cuando la configuración se carga en la memoria PROM que acompaña a la FPGA debe usarse la configuración CCLK, ya que será la propia FPGA la que cargará la configuración desde la PROM y se iniciará incluso cuando el cable de configuración no esté conectado.

Si la herramienta de configuración de la FPGA detecta que se está programando el chip FPGA pero el reloj de inicio configurado es CCLK, dará un aviso, ya que con esta configuración es posible que la FPGA no pueda hacer el proceso de inicio y, por tanto, no funcione nuestro diseño.

Para seleccionar el tipo de reloj de inicio para, por ejemplo, seleccionar JTAG CLK cuando se configura directamente la FPGA, puedes hacer lo siguiente:

- En el panel de procesos, haz click derecho sobre *Generate Programming File* y seleccione *Process Properties…*
- En la ventana que se abre seleccione la categoría *Startup Options* y elija el valor adecuado para la propiedad *FPGA Start-Up Clock*, por ejemplo, *JTAG Clock*.
- Pulsa *OK* para cerrar la ventana.

6.2 Programación desde el entorno ISE (iMPACT)

Una alternativa al uso de *Adept* para la programación de la placa es emplear la herramienta de configuración del entorno ISE (iMPACT). El procedimiento es el siguiente:

- Asegúrese de que la placa está conectada a un puerto USB y está encendida.
- Haga click derecho en el proceso *Configure Target Device* y selecciona *Run*. La herramienta iMPACT se abrirá en una nueva ventana.
- En el panel *iMPACT Flows* (arriba a la izquierda) haga doble click en *Boundary scan*. Se crea un nuevo proyecto de programación en el panel principal.
- Haga click derecho en el panel principal y seleccione *Cable Setup…*. En la ventana que aparece, seleccione *Open Cable Plugin* y escriba *digilent_plugin* en la entrada de texto. Pulsa OK. Con esto hemos configurado la herramienta para que use el complemento de programación suministrado por el fabricante de la placa, que ha debido ser instalado en el sistema previamente.
- Haga click derecho en el panel principal y seleccione *Initialize Chain*. La herramienta ahora detectará la placa y analizará qué dispositivos contiene. Cierre las ventanas que aparecen eligiendo “*No*” y/o “*Cancel*”.
- Ahora puede ver que la placa Basys2 tiene dos dispositivos que pueden ser programados, la propia FPGA (xc3s100e) y una memoria ROM (xcf02s). Para probar nuestro diseño lo programaremos directamente en la FPGA. El problema es que al desconectar la placa la programación de la FPGA se borrará. Podemos guardar la programación de forma permanente en el chip ROM, pero esta opción no la veremos en esta práctica.
- Para programar la FPGA haga lo siguiente:
- Haga click derecho en el icono del dispositivo xc3s100e y elige *Assign New Configuration File…*.
- Selecciona el archivo `counter.bit` en la carpeta del proyecto y pulse *OK*.
- Vuelva a hacer click derecho en el icono de la FPGA y selecciona *Program*. Uno de los LEDs de la FPGA parpadeará unos segundos. Luego el dispositivo estará configurado.
- Cierre la ventana de iMPACT.