



DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Laboratorios de Sistemas Digitales Avanzados **5º curso de Ingeniería en Informática**

Paulino Ruiz de Clavijo Vázquez <paulino@dte.us.es>

Índice de contenido

1	Introducción a Verilog y XILINX.....	3
1.	Introducción y objetivos.....	3
2.	Simulación de los componentes.....	3
2.1.	Diseño del convertidor a 7 segmentos.....	4
2.2.	Diseño del contador de 4 bits.....	5
3.	Simulación en el entorno Xilinx.....	6
3.1.	Creación de un proyecto en Xilinx ISE.....	7
3.2.	Añadir ficheros al proyecto.....	9
3.3.	Simulación y verificación de un diseño.....	10
4.	Implementación en FPGA.....	13
2	Circuito de refresco de un display.....	16
1.	Introducción y objetivos.....	16
2.	Visión global del sistema a desarrollar.....	16
3.	Realización de un divisor de frecuencia.....	17
3	Diseño de máquinas de estado.....	21
1.	Introducción y objetivos.....	21
2.	Diseño de un detector de flancos.....	22
3.	Diseño de un cronómetro.....	22
3.1.	Metodología de diseño UD-UC.....	23
3.2.	Diseño de la Unidad de Datos.....	24
3.3.	Diseño de la Unidad de Control.....	26
4.	Implementación del sistema completo.....	26
4	Diseño de una calculadora.....	28
1.	Introducción y objetivos.....	28
2.	Descripción del multiplicador secuencial.....	28
2.1.	Algoritmo de multiplicación.....	32
3.	Desarrollo de la sesión de laboratorio.....	35
4.	Memoria del trabajo realizado.....	35



DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Introducción a Verilog y XILINX

Paulino Ruiz de Clavijo Vázquez <Paulino@dte.us.es>

1. Introducción y objetivos

Uno de los objetivos generales de estos laboratorios es llegar a conocer una metodología de diseño de sistemas digitales. Para completar la formación en sistemas digitales es importante no cubrir exclusivamente en la parte más teórica sino que es necesario complementarla con un conocimiento de implementación real de dichos sistemas digitales.

Los objetivos de este laboratorio son:

- Familiarizarse con el lenguaje Verilog-HDL.
- Conocer el entorno de diseño sobre FPGA, en concreto el entorno de diseño de XILINX¹.
- Conocer las herramientas de verificación del diseño desarrollado.
- Desarrollar el proceso de diseño y simulación. Para ello se han elegido dos circuitos muy simples, uno combinacional y otro secuencial. Concretamente un convertidor de código y un contador.

2. Simulación de los componentes

Esta sesión de laboratorio consiste en describir dos componentes en Verilog, uno combinacional y otro secuencial, concretamente:

5. Un convertidor de código de binario a siete segmentos.
6. Un contador ascendente de 4 bits con varias señales de control.

Para ambos circuitos se han preparado unas plantillas de código que se deberán rellenar adecuadamente y están incluidas en este documento².

¹ Xilinx Inc.: Compañía de desarrollo de FPGAs (www.xilinx.com)

² Los ficheros necesarios para realizar este laboratorio están incluidos como adjuntos en el propio fichero PDF, utilice la opción de ver adjunto de su lector PDF para obtenerlos.

Nombre del fichero	Contenido	Descripción
convertidor.v	Módulo con el código del convertidor 7 segmentos	Debe completarlo
convertidor_tb.v	Testbench para el convertidor 7 segmentos	Se debe utilizar sin modificar para realizar la simulación
contador.v	Módulo con el contador modulo 16	Debe completarlo
contador_tb.v	Testbench para el contador módulo 16	Se debe utilizar sin modificar para realizar la simulación
lab1.v	Descripción estructural con la unión de los 2 módulos	Debe completarlo
lab1_tb.v	Testbench del sistema completo	Se debe utilizar sin modificar para realizar la simulación
basys2.ucf	Xilinx constraint file	Conexión de los componentes de la placa Basys2 con los Pads de la FPGA

Tabla 1. Ficheros necesarios durante la sesión de laboratorio.

A continuación se detalla cada uno de los circuitos y como deben desarrollarse en Verilog.

2.1. Diseño del convertidor a 7 segmentos

Para poder visualizar números se utilizan displays 7 segmentos. Estos displays tienen 7 entradas, uno por cada segmento que se puede iluminar, de forma que, al iluminar algunos de ellos se puede componer visualmente un número. En la figura 1 se muestran los números desde el 0 al 9.

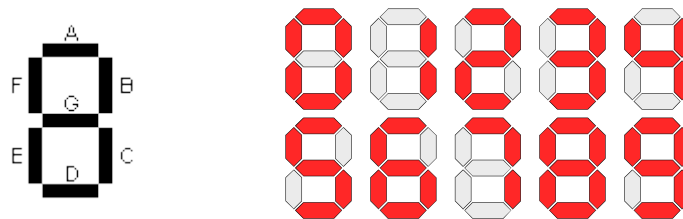


Figura 1. Ejemplo dígitos en un display de 7 segmentos.

Como primer paso para el diseño de este convertidor se presenta la tabla de verdad del circuito combinacional que hay que diseñar (tabla 2). Los nombres de los segmentos en la tabla 2 corresponden a los nombres asignados en la figura 1. La activación de los segmentos es en nivel **bajo**, es decir, para que un segmento se ilumine el valor de la señal debe colocarse a cero. En la tabla se representan los dígitos del 0 al 9 a partir de su valor en binario de 4 bits; también aparecen algunos números mayores de 9 para lo cual se iluminan ciertos segmentos de forma que aparezcan los dígitos hexadecimales.

bin ₃ bin ₂ bin ₁ bin ₀	A B C D E F G
0 0 0 0	0 0 0 0 0 0 1
0 0 0 1	1 0 0 1 1 1 1
0 0 1 0	0 0 1 0 0 1 0
0 0 1 1	0 0 0 0 1 1 0
0 1 0 0	1 0 0 1 1 0 0
0 1 0 1	0 1 0 0 1 0 0
0 1 1 0	0 1 0 0 0 0 0
0 1 1 1	0 0 0 1 1 1 1
1 0 0 0	0 0 0 0 0 0 0
1 0 0 1	0 0 0 0 1 0 0
1 0 1 0	0 0 0 1 0 0 0
1 0 1 1	1 1 0 0 0 0 0
1 1 0 0	0 1 1 0 0 0 1
1 1 0 1	1 0 0 0 0 1 0
1 1 1 0	0 1 1 0 0 0 0
1 1 1 1	0 1 1 1 0 0 0

Tabla 2: Tabla de verdad de conversión de binario a 7 segmentos.

Para diseñar el código Verilog se propone utilizar la estructura “case” de Verilog y usar como plantilla el siguiente fragmento de código (fichero *convertidor.v*):

```

module convertidor_bin7seg(
    input  [3:0] bin_in,      // entrada binaria 4 bits
    output reg a,b,c,d,e,f,g); // salida 7-segmentos

    // Escriba aqui el código
    // Se recomienda utilizar un proceso always con
    // una sentencia case

    ...

endmodule

```

Código 1. Fichero *convertidor.v*, plantilla de código Verilog para el convertidor 7 segmentos.

2.2. Diseño del contador de 4 bits

Dicho contador será disparado por el flanco de subida del reloj, tendrá una señal RESET síncrona para puesta a cero. Además, incluirá una señal de final de cuenta (CY) que se activará cuando el contador llegue al último estado de cuenta.

El contador debe cumplir la tabla comportamiento de la figura 2b y, para diseñar el código Verilog se propone utilizar como plantilla el fragmento de código 2 (fichero *contador.v*).

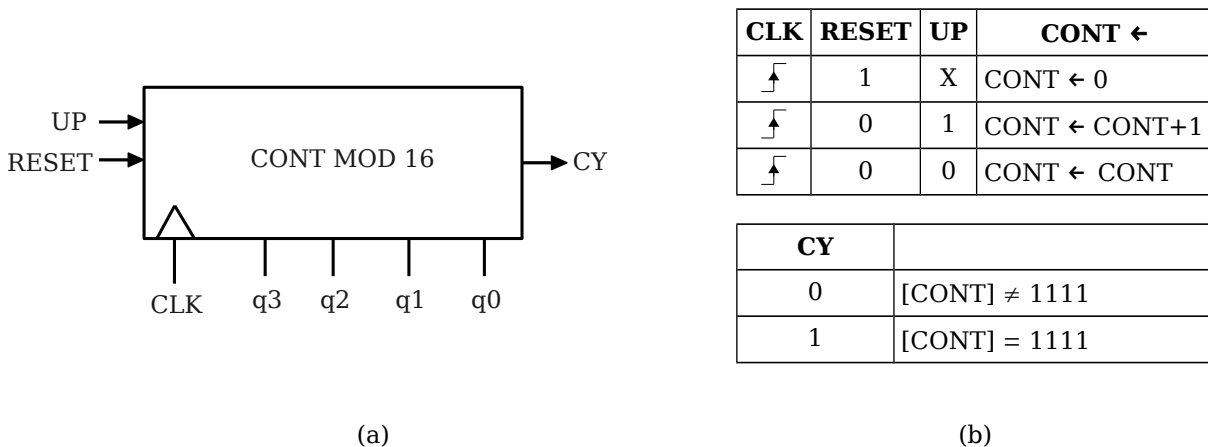


Figura 2. Descripción del contador módulo 16: (a) Descripción estructural, (b) Descripción funcional.

```

module contador_mod16(
  input clk,up,reset,
  output reg [3:0] q,
  output cy);

  // Escriba su código aqui
  // Cuidado con la señal cy, tiene que
  // activarse durante el último estado de cuenta

  ...

endmodule

```

Código 2. Fichero contador.v, plantilla de código Verilog para el contador módulo 16.

3. Simulación en el entorno Xilinx

Se va a realizar la simulación del sistema digital diseñado con el paquete de herramientas de Xilinx ISE. Se utilizarán unos ficheros con los *testbench* ya preparados. Los pasos a seguir son los siguientes:

1. Siga los pasos del tutorial de uso de Xilinx ISE, incluido en la siguiente sección, para crear un nuevo proyecto *Laboratorio1*, incluyendo los ficheros Verilog suministrados
2. Complete los ficheros indicados anteriormente: *convertidor.v* y *contador.v*
3. Realice la simulación del convertidor binario a 7 segmentos siguiendo los pasos descritos en el apéndice, con ayuda del testbench facilitado (fichero *convertidor_tb.v*). Compruebe si el bloque convertidor funciona correctamente.
4. Realice ahora la simulación del contador de 4 bits, con ayuda del testbench facilitado (fichero *contador_tb.v*). Compruebe si el contador funciona correctamente.
5. Fíjese que al simular el contador no aparecen todos los estados ni la activación de la señal *carry*. Haga los cambios oportunos para poder visualizar al menos un ciclo completo de cuenta y vuelva a realizar la simulación para comprobarlo.
6. Añada una descripción estructural en Verilog que interconecte ambos bloques: *convertidor* y *contador*, de forma que ambos aparezcan incluidos por este nuevo fichero. Utilice la plantilla de

código (véase Código 3) suministrada (fichero *lab1.v*).

7. Realice por último la simulación de este nuevo bloque, con ayuda del testbench facilitado (fichero *lab1_tb.v*). Compruebe si el sistema completo funciona correctamente.

```
module lab1(  
    input clk, up, reset,  
    output [0:6] seg,  
    output cy);  
  
    // Declare un cable para  
    // interconectar la salida del contador  
    // con la entrada del convertidor  
  
    ...  
  
    // Instancie el contador modulo 16  
    // y realice las conexiones correctamente  
  
    ...  
  
    // Instancie el convertidor binario a 7 segmentos  
    // y realice las conexiones correctamente con el  
    // modulo anterior  
  
    ...  
  
endmodule // lab1
```

Código 3. Fichero *lab1.v*, plantilla de código Verilog para el contador módulo 16.

Para realizar los pasos indicados siga el tutorial de Xilinx de la siguiente sección.

3.1. Creación de un proyecto en Xilinx ISE

Tras iniciarse el sistema operativo, el primer paso es arrancar el entorno ISE y crear un nuevo proyecto. En el menú **File** hay que utilizar la opción **New Project**, obteniéndose la ventana mostrada en la figura 3 donde hay que escribir un nombre para el proyecto, por ejemplo *Laboratorio2*. La herramienta creará una carpeta con ese mismo nombre y guardará en su interior todo lo que se va generando a medida que vamos trabajando en ese proyecto.

Tras escribir el nombre se activa el botón **Next** y aparece el cuadro de diálogo mostrado en la figura 4, donde hay que establecer todas las opciones indicadas en la figura. Concretamente hay que asegurarse de establecer las siguientes opciones a su valor correcto:

- **Family:** Spartan 3E
- **Device:** XC3S100E
- **Package:** CP132
- **Preferred Language:** Verilog

El resto de opciones deberían estar por defecto a los mismos valores que los mostrados en la figura 4. Tras establecer las opciones correctas, utilizando el botón **Next** aparece una última ventana con información y un botón **Finish** que se pulsa para crear el proyecto. La figura 5 muestra el proyecto recién creado, sólo se muestra el nombre del proyecto y el tipo de FPGA "xc3s100e-cp132".

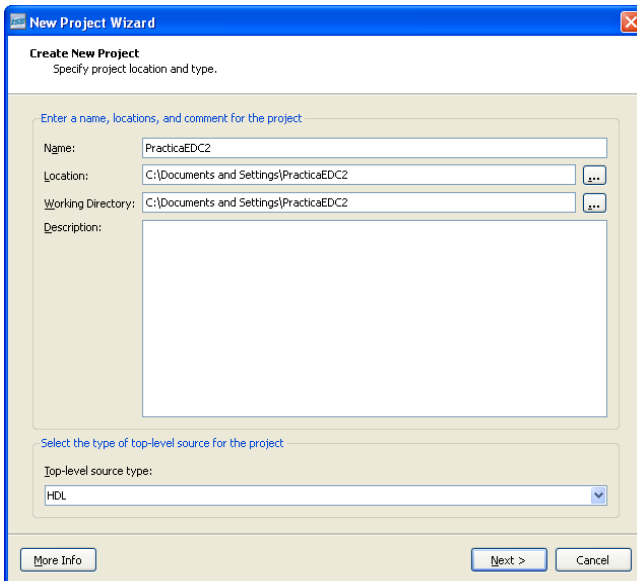


Figura 3. Ventana de creación del proyecto.

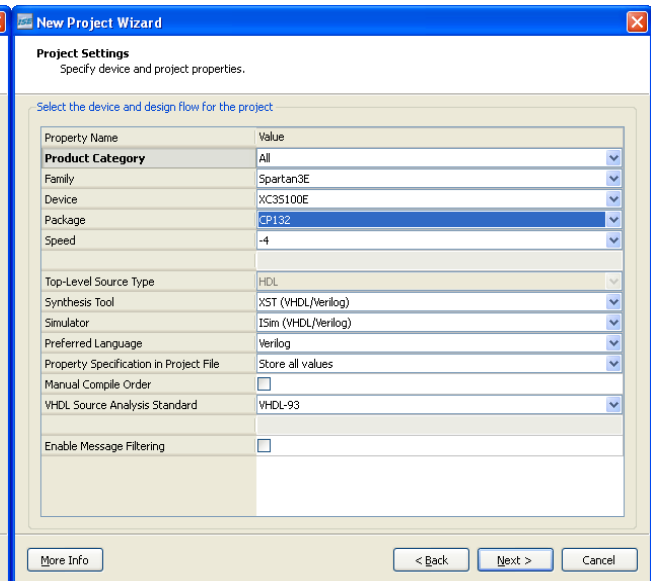


Figura 4. Ventana de propiedades del proyecto.

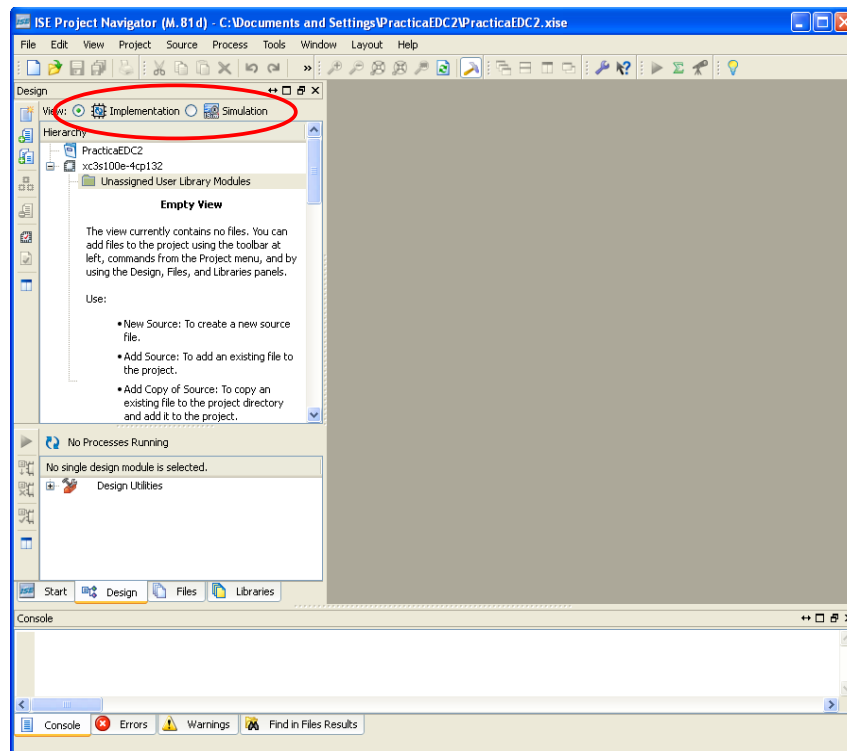


Figura 5. Vista general de un proyecto en el entorno Xilinx.

Nótese que encima del nombre del proyecto aparecen dos iconos *Implementation* y *Simulation*, (ver figura 5) cada uno con distintas vistas del mismo proyecto. Debemos procurar estar siempre en modo de **simulación** para evitar problemas con el entorno ISE. También se recomienda utilizar la entrada de menú **Layout** → **Restore Default Layout** en caso de no ver correctamente las ventanas o los controles de *ISE Project Navigator* (o del entorno de Simulación *Isim* que usará más adelante).

3.2. Añadir ficheros al proyecto

El primer paso tras la creación del proyecto es añadir los ficheros Verilog (diseños y testbenchs) al proyecto. Para añadir ficheros al proyecto se puede utilizar la opción de menú **Project** o, pulsar el botón derecho del ratón en la zona en blanco de la vista del proyecto, eligiendo entre **Add Source** o bien **Add copy of source** teniendo en cuenta que estas dos opciones son algo diferentes:

- **Add Copy of Source** crea un nuevo fichero dentro del proyecto que inicialmente será una copia del fichero fuente seleccionado. Así las modificaciones serán propias a este proyecto y el fichero fuente original no se modificará. La copia residirá dentro de la carpeta del proyecto.
- **Add Source** no crea un nuevo fichero dentro del proyecto, utiliza el propio fichero fuente seleccionado sin crear ninguna copia. Así las modificaciones afectarán al fichero fuente en su ubicación original, es decir, el fichero residirá en la misma carpeta dónde esté almacenado previamente.

Otra posible opción sería **New source** creándose un nuevo fichero fuente vacío donde habría que escribir el código.

La mejor opción es **Add Source** seleccionando uno o más ficheros a añadir al proyecto. Hay que confirmar los ficheros que son para implementación y cuales son exclusivamente para simulación (cómo los ficheros de *testbench* suministrados). En la figura 6 se muestran los ficheros a añadir y la asociación realizada en cada uno de ellos para que la simulación opere correctamente (elija *All* y *Simulation* según se indica).

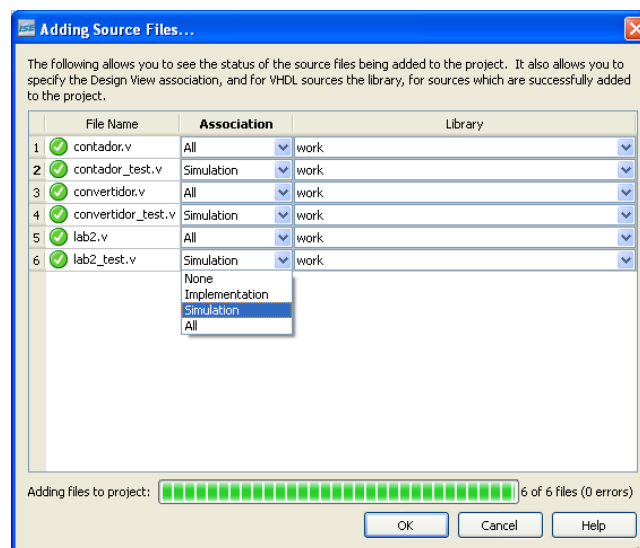


Figura 6. Ventana de inclusión de ficheros al proyecto.

Una vez añadidos los ficheros, éstos son mostrados en la vista del proyecto de forma jerárquica y, componiendo el árbol de proyecto de ficheros en función de que un fichero necesite de otro fichero, esto se puede visualizar en la figura 6. El fichero que incluye a los demás será el primer fichero del árbol de proyecto.

Para editar o ver cualquiera de los ficheros del proyecto, sólo hay que seleccionarlo con el ratón en el árbol de proyecto y pulsar el botón izquierdo del ratón dos veces.

3.3. Simulación y verificación de un diseño

La simulación nos permite verificar el correcto funcionamiento de una unidad/módulo diseñado, para los casos que se plantean en el fichero de testbench. Éstos podrán ser más o menos completos según el caso y si encontramos algún problema, nos permite indagar la causa del mismo antes de pasar a modificar el código. Efectuando correcciones y simulaciones se consigue solucionar todos los problemas que pueda tener el diseño.

Tras añadir un fichero de testbench, éste no se muestra en la vista de *implementación*, únicamente aparece en la vista *simulación*. Esto es debido a que dichos ficheros contienen información para realizar una simulación/verificación del diseño lógico, pero no se usa para realizar una implementación de la unidad (la última etapa del proceso de diseño). También puede comprobar como en la vista de simulación aparecen las unidades en un orden jerárquico distinto al de implementación. Concretamente, los ficheros de testbench aparecen en primer lugar, puesto que estos serán los que van a guiar la simulación, como se muestra en la figura 7.

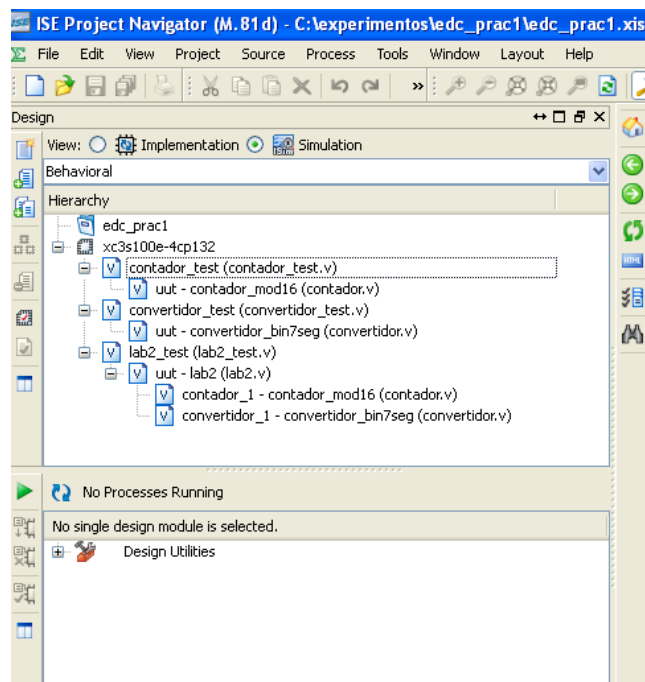


Figura 7. Vista jerárquica de un proyecto.

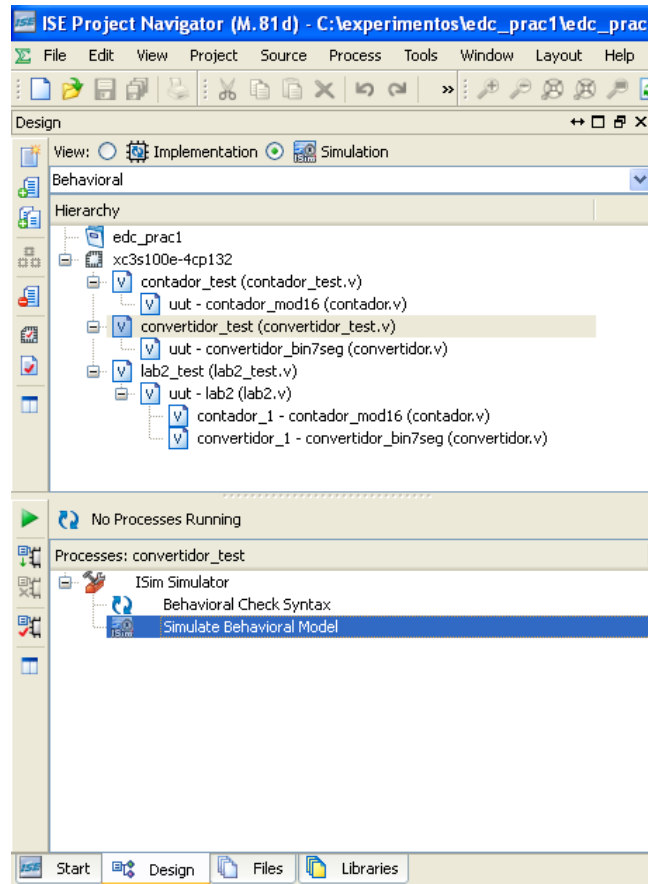



Figura 8. Selección de simulación para el convertidor binario 7 segmentos.

Así, para simular una unidad debemos resaltar el nombre de la unidad/fichero de testbench a simular y abajo en la caja titulada *Processes* desplegar la entrada *ISim Simulator* para ejecutar la orden *Simulate Behavioral Model* pulsando el ratón dos veces, como muestra la figura 8.

Si no hay errores en el diseño, se abrirá una nueva ventana con el simulador *ISim* y ejecutará una simulación durante un corto periodo de tiempo (habitualmente $1\mu\text{s}$), deteniéndose la simulación en ese punto, salvo que el testbench detenga la simulación con antelación (con la orden *\$finish*).

Si no hay errores en el código se abrirá el simulador *ISim* donde, para ver las formas de onda, hay que utilizar la pestaña **DEFAULT.WCFG**. Es aconsejable utilizar en este momento el icono  (**ZOOM TO FULL VIEW**) para tener una vista completa de toda la simulación.

En esta vista, mostrada en la figura 9, se dispone de una ventana con las formas de onda a la derecha en fondo negro y varias señales representadas con sus valores simulados, que deben ser las entradas y salidas de nuestra unidad (al menos aquellas que aparecen en el testbench). Si hacemos pulsamos el ratón sobre el gráfico de formas de ondas, se sitúa un cursor amarillo indicando datos exactos en ese instante de tiempo (nótese como cambian los valores de las señales al situarse en distintos instantes). Para las señales de varios bits podemos cambiar la codificación pulsando el botón derecho del ratón sobre el nombre de la señal y, accediendo en el menú flotante a la opción *RADIX* (binario, hexadecimal, decimal, etc.).

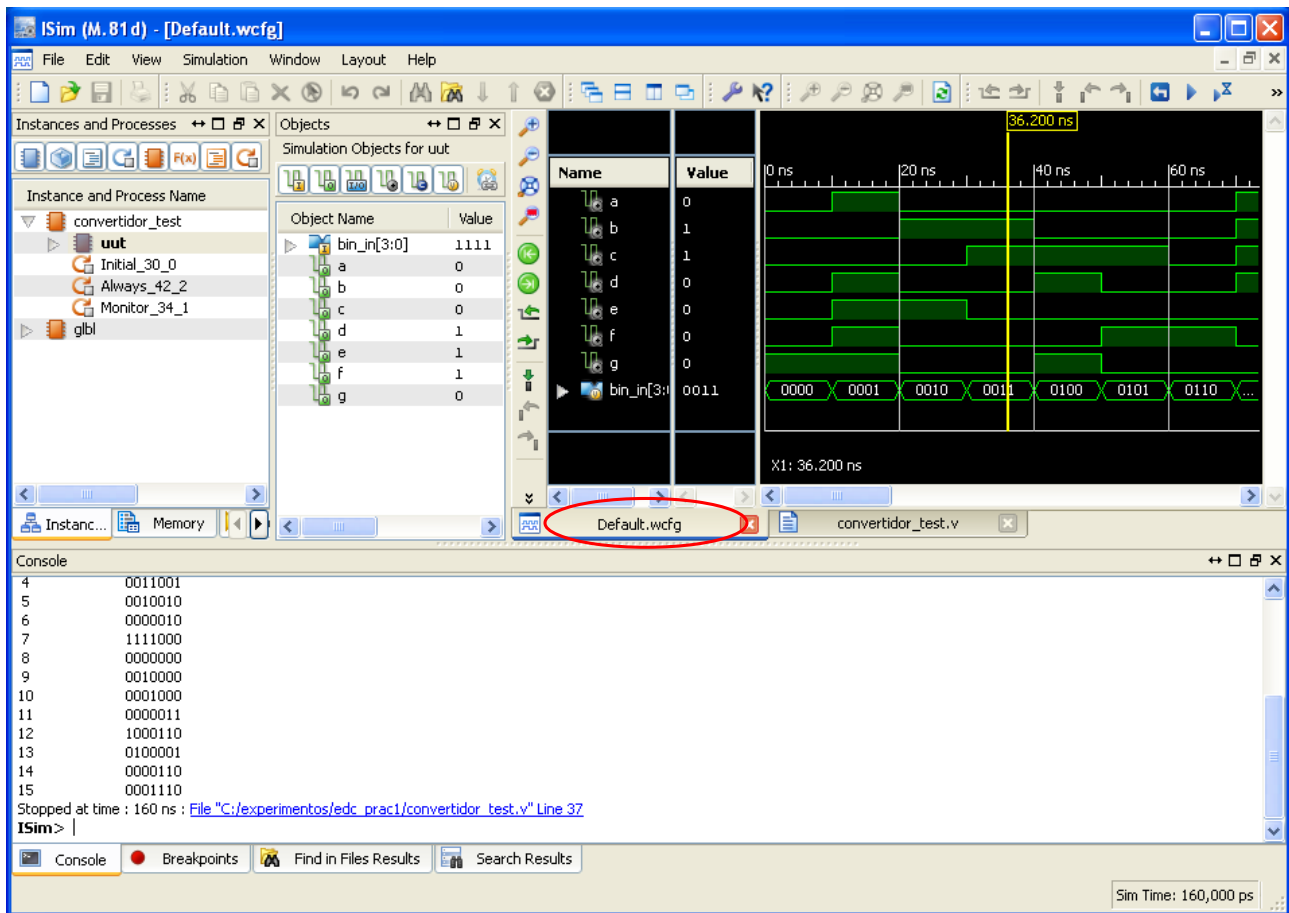


Figura 9. Simulación del convertidor binario 7 segmentos con ISim.

Otros controles importantes de *ISim* se encuentra en dos bloques situados a la izquierda con los que se puede navegar por todas las unidades que componen el diseño. Utilizando estos dos bloques se pueden buscar señales y componentes internos de cualquier módulo para poder mostrar sus formas de ondas, pero, para ello habría que reiniciar la simulación tras añadirlas a la vista de simulación.

Por último, en la barra de iconos superior hay varios iconos que permiten hacer ampliaciones y reducciones de escala en la imagen. Además de estos iconos, varios iconos verdes que hay a continuación sirven para navegar por la forma de onda y, los iconos azules, controlan la simulación utilizándose para continuar o detener el proceso.

Los iconos verdes se deben usar para buscar o centrarse en una parte concreta de las formas de onda, por ejemplo, *Previous Transition* y *Next Transition* nos permiten ir al anterior/siguiente flanco de una señal seleccionada previamente en la ventana de formas de onda.

Los iconos azules controlan el flujo de ejecución de la simulación permitiendo: borrar la simulación actual volviendo al instante cero (*Restart*), continuar la simulación indefinidamente (*Run All*), continuar un tiempo de simulación determinado deteniéndose automáticamente (*Run*), ejecutar la simulación línea a línea de Verilog (*Step*) y, por último, detener una simulación en ejecución (*Break*).

4. Implementación en FPGA

Finalmente se implementará el sistema digital realizado en un dispositivo programable tipo FPGA incluida en la placa de desarrollo Basys2.

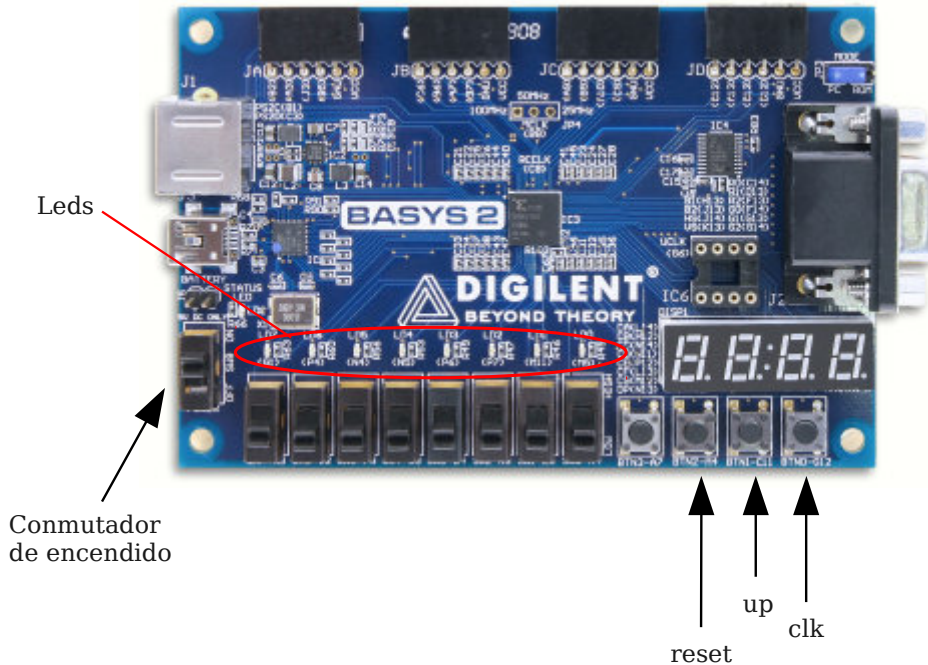


Figura 10. Fotografía de la placa de desarrollo Basys2.

En la implementación se conectarán las señales de control del contador a los botones de la placa de desarrollo. También, las salidas del convertidor de 7 segmentos se conectarán al display. Aunque las placas disponen de un reloj interno, para esta primera prueba se utilizará como señal de reloj uno de los botones (ver figura 10).

Para implementar un sistema completo es necesario utilizar otro fichero donde se indican las conexiones entre los Pads del chip FPGA y los componentes de la placa. Este fichero se llama *basys2.ucf* y ya contiene la asignación de las conexiones del sistema a los componentes de la placa.

Los pasos a seguir son los siguientes:

1. Cambiar el proyecto ISE del modo simulación al modo implementación tal y como se mostraba en la figura 8
2. Añadir dos ficheros al proyecto:
 - 2.1. Fichero *basys2.ucf*. Debe comprobar que también aparece en el árbol de proyecto.
 - 2.2. Fichero *lab1.v* como implementación. Este fichero aparecerá ahora como raíz del proyecto.
3. La implementación se realiza seleccionando el módulo *lab1* en el árbol de proyecto. Entonces debe seleccionar la opción **Generate Programming File** (figura 11) y seguir los siguientes pasos:
 - 3.1. Pulsando el **botón derecho** del ratón sobre la opción **Generate Programming File** aparecerá

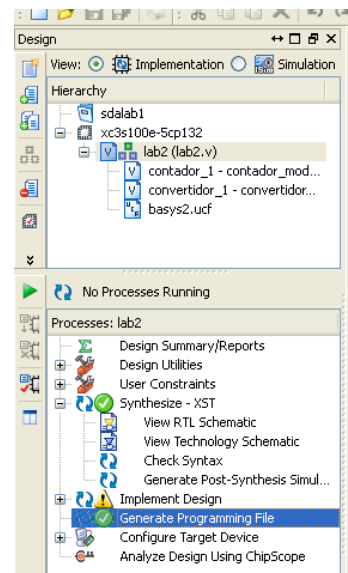


Figura 11. Implementación

un menú flotante como el mostrado en la figura 12a, seleccionando la opción de menú de **Process Properties** aparecerá el diálogo mostrado en la figura 12b.

3.2. En el diálogo hay que elegir la categoría **Startup Options** y cambiar el valor del primer campo **FPGA Start-Up Clock** de **CCLK** a **Jtag-Clock**. Tras aceptar los cambios con el botón **OK** se volverá a la ventana principal de ISE.

3.3. Pulsando dos veces botón izquierdo del ratón sobre **Generate Programming File** se ejecuta el proceso completo y se genera el fichero de programación. Tal y como se muestra en la figura 11, si el proceso ha terminado con éxito aparecerá un indicador verde, en caso contrario un aspa rojo indica que hay errores que habría que corregir y repetir el proceso.

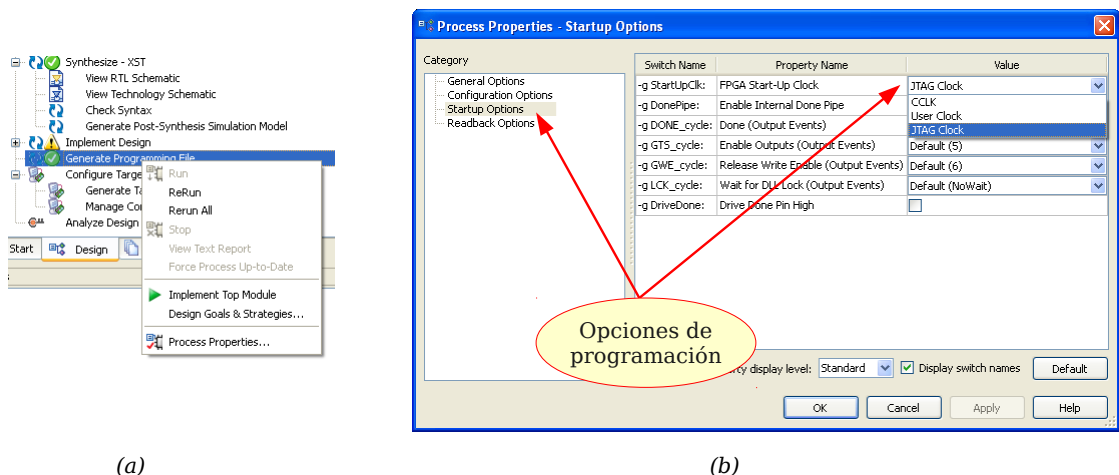



Figura 12. Opciones de generación del fichero de programación:
(a) menú desplegable, (b) diálogo con opciones

4. El último paso consiste en programar la placa de desarrollo con un fichero que se ha generado tras el proceso del paso anterior. Concretamente, el fichero debería llamarse *laboratorio2.bit* y hay que transferirlo por la conexión USB a la FPGA. Para ello siga los siguientes pasos:

4.1. Conecte el puerto USB de la placa de desarrollo y conmute la alimentación de la placa. Por defecto debe cargarse un programa de test de la propia placa que cuenta dígitos BCD en el Display. Además, puede comprobar el correcto funcionamiento de los conmutadores y los botones antes de proceder con la programación.

4.2. Inicie el programa *Adept* desde el menú de inicio (menú *Digilent* → *Adept*, icono ) y aparecerá el programa mostrado en la figura 13. Con este programa se puede transferir el fichero de programación (*bitstream*) a la FPGA. El programa *Adept* permite programar los componentes de la placa Basys2, estos son, una FPGA y una PROM. Se programará la FPGA, por tanto, se debe utilizar el botón *Browse* indicado en la figura y seleccionar el fichero *.bit*. Hay buscar la carpeta del proyecto ISE en el que está trabajando, allí encontrará el resultado de la síntesis en un fichero *.bit*, concretamente, *laboratorio2.bit*. Una vez seleccionado este fichero se activará el botón *Program* y bastará con pulsarlo para la placa se programe.

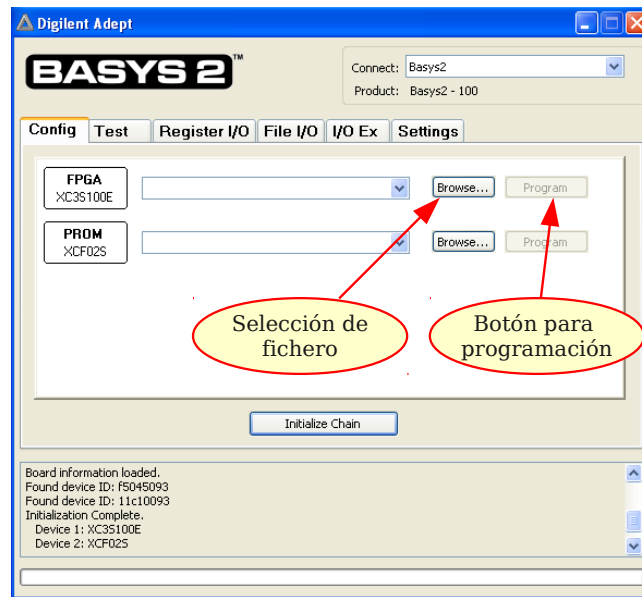


Figura 13. Programación de placas Digilent con Adept.

5. Utilice los botones para conseguir que el contador avance:
 - 5.1. Mantenga pulsado el botón UP y genere un pulso de reloj
 - 5.2. Mantenga pulsado el botón RESET y genere pulsos de reloj
 - 5.3. Compruebe lo que sucede en el fin de cuenta del contador.



DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Circuito de refresco de un display

Paulino Ruiz de Clavijo Vázquez <paulino@dte.us.es>

1. Introducción y objetivos

Uno de los objetivos generales de la asignatura es llegar a conocer metodologías de diseño de sistemas digitales, así, la metodología utilizada a lo largo del curso será la denominada *bottom-up*. Esta metodología consiste básicamente en desarrollar en primer lugar los módulos de menor nivel en el diseño para unirlos finalmente en módulos de nivel superior hasta completar el diseño del sistema.

Los objetivos de este laboratorio es desarrollar un módulo reutilizable en los posteriores diseños. Concretamente se tratan estos aspectos:

- Familiarización con el uso del reloj.
- Comprobar el efecto del refresco en pantallas.

Nombre del fichero	Contenido	Descripción
convertidor.v	Módulo con el código del convertidor 7 segmentos	Proviene de la sesión anterior
display_mem_tb.v	Testbench para el módulo display_mem.v	Debe corregir los errores que contiene
basys2.ucf	Xilinx constraint file	Conexión de los componentes de la placa Basys2 con los Pads de la FPGA

Tabla 3. Ficheros necesarios durante la sesión de laboratorio.

2. Visión global del sistema a desarrollar

Se diseñará un controlador de *display* como el mostrado a nivel de bloques en la figura 14. Este controlador contiene una memoria de 16bits donde se escribirá el dato a mostrar. Cada uno de los

displays mostrará los dígitos 0-9-A-F, representándose en total 4 bits en cada uno de los dígitos.

Habitualmente para ahorrar Pads en las conexiones de displays cada segmento de todos los diferentes dígitos están conectados entre sí. Esto significa que al intentar iluminar un segmento se ilumina dicho segmento en todos los dígitos. Para visualizar diferentes segmentos simultáneamente en cada dígito se controlan los ánodos de los dígitos, éstos, se conectan a otros Pads. Estos ánodos se comportan como interruptores ON-OFF de los displays.

Como veremos en esta sesión de laboratorio, activando los diferentes dígitos de manera consecutiva y a intervalos de tiempo regulares se pueden visualizar diferentes dígitos en cada display. Sólo hay que estableciendo los valores correctos los segmentos en cada intervalo temporal para visualizar diferentes números.

Es importante controlar el display con una señal de una frecuencia adecuada para que el ojo humano no perciba el efecto, esto se mostrará a lo largo del desarrollo.

Por otro lado, indicar que la señal de control $w_display$ se utiliza para cambiar el número a mostrar, realmente se escribe en la memoria interna de 16bits.

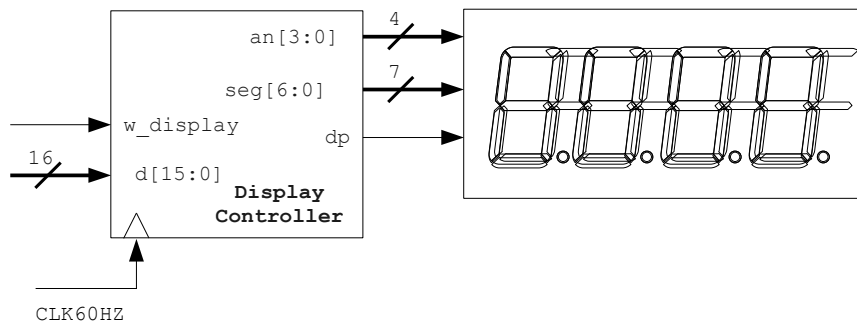


Figura 14. Controlador de display 7 segmentos.

3. Realización de un divisor de frecuencia

Para comenzar el desarrollo comprobará el correcto funcionamiento del reloj de la placa mediante la implementación de un divisor de frecuencia conectado a un LED. Se utilizará un contador como divisor de frecuencia.

Partiendo de un proyecto vacío realice lo siguiente:

6. Añada un nuevo módulo al proyecto en un fichero llamado *contador_generico.v*. Diseñe como módulo independiente un contador de 32 bits como el de la figura 15.
7. Ahora cree un nuevo testbench para el contador, para ello, añada un nuevo fichero al proyecto con el asistente indicando el tipo de fichero como *Verilog Test Fixture* y llamado *contador_generico_tb.v*. Al tener el mismo nombre de fichero, pero terminado en *"_tb"* asociará automáticamente el módulo con el mismo nombre.

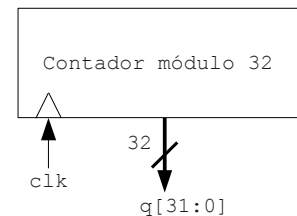


Figura 15. Contador de 32 bits.

Los testbenchs en Verilog están formados por un conjunto de procesos ejecutados concurrentemente.

Existe un bloque especial llamado *initial* que es ejecutado una sola vez aquí, se establecen los distintos valores que tomarán las señales de entrada. Este proceso se suele finalizar con la sentencia `$finish;` con ella se para el simulador. En caso de existir otro proceso concurrente que no hubiera finalizado, si no se utiliza esta sentencia la simulación continuaría.

En la generación del reloj se suele utilizar un proceso separado y ejecutado concurrentemente donde se establece cada cuanto tiempo se debe invertir la señal de reloj. Para ello se utiliza la sentencia `always`.

Dentro de cada proceso los cambios de las señales se retrasan un determinado intervalo de tiempo para construir el test. Esta espera de tiempo se consigue precediendo cualquier bloque Verilog por `#<numero>` donde el número indica las unidades de tiempo a esperar según la escala de tiempo establecida al principio del fichero.

Tras estas indicaciones se propone realizar lo siguiente:

8. Utilizando como ejemplo el testbench del contador del laboratorio anterior, cree un testbench usando la sentencia Verilog `repeat (<numero>) @(<señal>)` generando un reloj con al menos 1000 ciclos. Realice una simulación y no se olvide de añadir el proceso encargado de invertir el reloj.

8.1. Solucione el problema observado en la simulación alterando el contador y el testbench.

8.2. Realice la simulación y cambie la opción *Radix* de la salida Q del contador a decimal. Despliegue las formas de onda individuales de cada bit para comprobar la división de frecuencia en cada bit del contador.

El reloj de la placa Basys2 funciona por defecto a 50Mhz y está conectado al Pad B8 de la FPGA. El objetivo es conseguir que un Led de la placa parpadee entre 1-5 veces por segundo, es decir, a una frecuencia entre 1hz-5hz. Debe escoger una salida adecuada del contador para conseguirlo conectándola con el Led.

9. Cree un nuevo módulo en un fichero separado con dos entradas y una salida: EXTCLK, RESET y LED0 respectivamente.

9.1. Instancie el contador de 32 bits interconectando el reloj externo y la señal RESET con el contador.

9.2. Escoja una salida adecuada de las 32 disponibles en el contador para conectarla a la señal LED0.

9.3. Utilice el fichero adjunto llamado *basys2.ucf* para conectar correctamente las entradas y salidas de su módulo a los componentes de la placa Basys2.

9.4. Implemente el desarrollo, programe la placa y compruebe si el LED0 parpadea.

Se realizará ahora un segundo circuito consistente en un generador de secuencia para controlar la iluminación de los cuatro displays 7 segmentos. La figura 16 muestra el diagrama de bloques con los componentes que forman el controlador de display a diseñar.

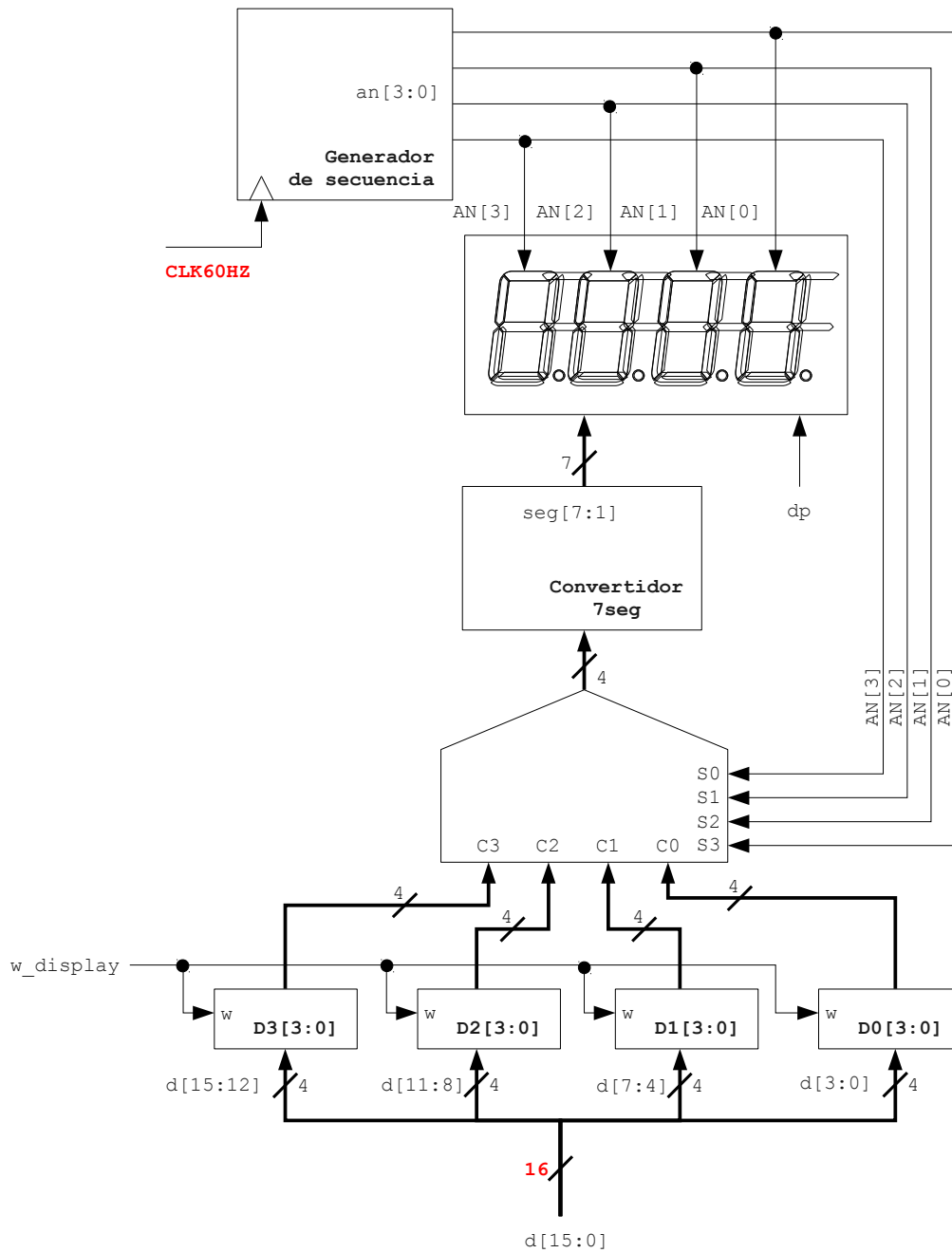


Figura 16. Diagrama de bloques de un controlador para un display 7 segmentos.

10. Realice un módulo llamado *secuencia* con una entrada de reloj y otra de reset que genere consecutivamente la secuencia: 0111, 1011, 1101, 1110. Las salidas serán un bus de 4 bits llamado AN. Simule el circuito para comprobar el funcionamiento.
11. En un nuevo fichero cree otro módulo llamado *display* con la siguiente interfaz: `module display(input extclk, input reset, output [3:0] an)`
 - 11.1. Instancie en el módulo del contador de 32 bits y el generador de secuencia. Debe usar como reloj para el generador de secuencia una de las salidas el contador para que oscile a 1-5hz.
 - 11.2. Cree un fichero UCF para conectar las señales AN a 4 Leds, la señal *extclk* al reloj al de la placa y la señal *reset* a un botón.
 - 11.3. Programe la placa y compruebe los cuatro leds.

12. Los cuatro registros y el multiplexor se realizarán en un único módulo llamado *display_mem* con la siguiente interfaz: `module display_mem(input [15:0] d_in, input w, input reset, input [3:0] sel, output [3:0] d_out)`.
 - 12.1. Se recomienda realizar el multiplexor con una sentencia *case*. Considere utilizar la clausula *default* en la sentencia ya que no se cubrirán todos los casos.
 - 12.2. Utilice el fichero *display_mem_tb.v* para realizar la simulación. Este fichero no se limita a realizar una simulación, además, comprueba si el módulo funciona correctamente, pero contiene un error.
 - 12.3. Ejecute el testbench y compruebe la salida de texto mostrada tras la simulación donde se indicarán los errores obtenidos. Corrija el testbench para que opere correctamente y compruebe si su módulo contiene errores.
13. Para terminar el controlador de display se modificará el módulo *display* con la siguiente interfaz: `module display(input extclk, input reset, output [6:0] seg, output [3:0] an)`.
 - 13.1. En el módulo debe instanciar: el contador de 32bits, el generador de secuencia, la memoria y el convertidor 7 segmentos, este último, realizado en el laboratorio anterior.
 - 13.2. Para realizar una prueba rápida se conectará la salida de 1-5hz del contador a la señal de escritura de la memoria del displays. Además debe establecer constante el valor del dato *d_in* de la instancia de la memora a un valor de 16 bits, por ejemplo, 16'ABCD.
 - 13.3. Edite el fichero UCF asociando correctamente las entradas y salidas de este nuevo módulo.
 - 13.4. Programe la placa para comprobar el resultado.
 - 13.5. En una segunda prueba se propone conseguir una tasa de refresco del display al menos de 25hz, para ello, cambie la frecuencia del reloj del generador de secuencia a una señal del divisor de frecuencia adecuada para 25-60hz.
 - 13.6. Vuelva a programar la placa para comprobar el resultado.
14. Por último se propone que vuelva a implementar el módulo *display* mediante una única descripción Verilog, es decir, sin utilizar instancias de otros módulos. Para ello utilice varias sentencias *always* en el mismo módulo.



DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Diseño de máquinas de estado

Sistemas Digitales Avanzados

1. Introducción y objetivos

Las máquinas de estado finitas (FSM) son fundamentales en el diseño circuitos digitales secuenciales . Con ellas se consigue implementar algoritmos complejos existiendo procedimientos para obtener el código HDL de manera sistemática. Utilizaremos una descripción llamada en la bibliografía descripción *canónica de máquinas de estado*.

Los objetivos de esta sesión de laboratorio son realizar la implementación de máquinas de estado mediante un procedimiento sistemático, facilitando así la codificación HDL. Se realizará lo siguiente;

- Implementación de una pequeña máquina de estados para detección de flancos.
- Implementación de una máquina de estados para control de un cronómetro.
- Implementación completa de un cronómetro usando la metodología de diseño basada en Unidad de Datos y Unidad de Control.

Nombre del fichero	Contenido	Descripción
display.v	Controlador de display.	Proviene de la sesión anterior.
contador_bcd_tb.v	Testbench para el contador sexagesimal	Debe completarlo
segundero.v	Generador de pulsos	Genera un pulso cada segundo, referente a un reloj de 50MHZ.
unidad_control_tb.v	Testbench para la unidad de control	No debe ser necesario modificarlo
basys2.ucf	Xilinx constraint file.	Conexión de los componentes de la placa Basys2 con los Pads de la FPGA.

Tabla 4. Ficheros necesarios durante la sesión de laboratorio.

2. Diseño de un detector de flancos

En primer lugar se diseñará una máquina de estados para detectar flancos de subida. Esta máquina se utilizará para conectar cada uno de los botones existentes en la placa de desarrollo a los sistemas a desarrollar.

El objetivo es generar un pulso de un ciclo de duración cuando un botón se pulse, independiente del tiempo que esté el botón pulsado. Puesto que el reloj de la placa funciona a 50Mhz cualquier pulsación realizada en un botón durará multitud de pulsos de reloj, con este sistema, se detecta una pulsación larga usando su flanco de subida, independientemente del tiempo que tarde el botón en soltarse. En la figura 17 se muestra el módulo a diseñar cuyo diagrama de estados es el mostrado en la figura 18.

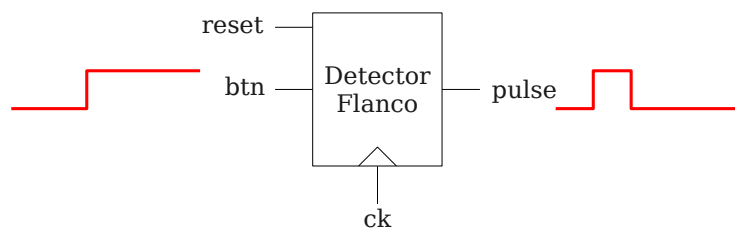


Figura 17. Esquema del detector de flancos.

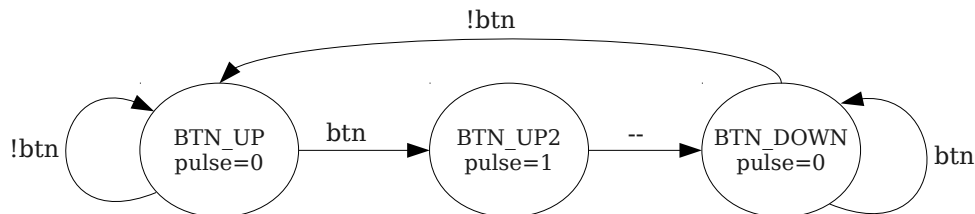


Figura 18. Máquina de estados para el detector de flancos.

Se propone realizar lo siguiente:

15. Cree un módulo Verilog llamado *detecta_flanco* correspondiente a la figura 17 que implemente la máquina de estados de la figura 18. Defina los estados con la sentencia *parameter* para poder realizar diferentes asignaciones.
 - 15.1. Realice un testbench con varios pulsos de varios ciclos de duración y compruebe si la salida genera un único pulso en cada flanco de subida. No se olvide de realizar un *reset* inicial.
 - 15.2. Busque en Internet un diseño en Verilog de un detector de flancos más simple que su diseño. Se recomienda realizar la búsqueda en inglés.

3. Diseño de un cronómetro

Ahora se pretende diseñar un cronómetro con dos modos de funcionamiento: cronómetro y temporizador. Para el control del mismo se utilizará el detectores de flancos conectados cada uno de los botones disponibles en la placa de desarrollo.

El núcleo de este diseño es un contador en sexagesimal capaz de realizar la cuenta de minutos y

segundos, tanto en modo ascendente como descendente. Este contador, mostrado en la figura 19, consiste en 4 salidas BCD. Las salidas BCD representan en decimal los minutos (en los dos dígitos más significativos) y los segundos (en los dos dígitos menos significativos), por tanto, el formato de salida es *mm:ss*.

El funcionamiento de este contador se puede resumir indicando los valores que tomarán cada una de las salidas BCD:

- Dígito BCD1 y BCD0: Corresponden a los segundos toma valores 0-5 y 0-9 respectivamente.
- Dígito BCD3 y BCD2: Corresponden a los minutos ambos toman valores 0-9, por tanto se puede llegar 99 minutos ya que no tenemos dígitos para indicar las horas.

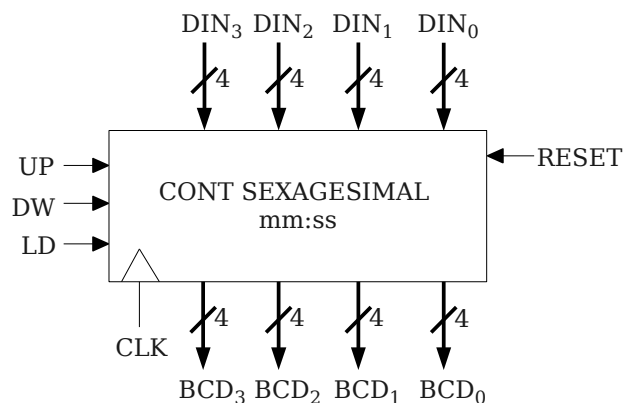


Figura 19. Diagrama de bloques Contador Sexagesimal con salida BCD.

Siga los siguientes pasos:

16. Cree un nuevo modulo llamado *contador_bcd* correspondiente a la figura 19. Respete los nombres de la conexiones para que el testbench suministrado opere correctamente.
 - 16.1. Considere en el contador que los dos últimos dígitos variarán de 00 a 59 mientras que los dos primeros desde 00 a 99.
 - 16.2. Utilice el testbench *contador_bcd_tb.v* para comprobar su diseño. Observe si hay errores en la salida de texto del simulador. En caso de encontrar errores, en las formas de onda de la simulación hay una señal llamada *error* que se activa durante el error detectado, puede utilizarla para buscar los errores.
 - 16.3. El test anterior es incompleto, sólo comprueba los valores en modo ascendente. Complete el código para comprobar todos los valores en modo descendentes.
 - 16.4. Contemple en su diseño la llegada a cero del contador en modo descendente. Se propone bloquearlo en cero cuando se alcance este valor.

3.1. Metodología de diseño UD-UC

Una metodología de diseño ampliamente utilizada para el diseño de circuitos digitales consiste dividir los diseños en dos componentes llamados Unidad de Datos y Unidad de Control. A su vez estos componentes según su complejidad, también se subdividen.

Básicamente la Unidad de Datos consiste en agrupar los componentes encargados de almacenar y operar con datos. Estos componentes se interconectan mediante buses y las señales de control de los componentes quedan como señales E/S de la Unidad de Datos. Estos componentes suelen ser registros, unidades aritméticas/lógicas, contadores, etc.

Por otro lado la Unidad de Control no es más que una máquina de estados secuencial conectada a las señales de control de todos los elementos que forman la Unidad de Datos. Esta máquina secuencial se encarga de activar señales de la Unidad de Datos ciclo a ciclo de reloj. Así, consigue realizar operaciones con los componentes de la unidad de datos, tras una secuencia determinada de activación de señales en la Unidad de Datos, habrá terminado la operación para la que estaba diseñada.

Según esta metodología, el diseño del cronómetro se dividirá en dos módulos denominados *unidad_datos* y *unidad_control*. Ambos estarán interconectados siguiendo el esquema mostrado en la figura 20. La Unidad de Control recibirá ordenes del usuario a través de los pulsadores y la Unidad de Datos mostrará en el display los minutos y segundos pudiendo recibir datos desde los conmutadores.

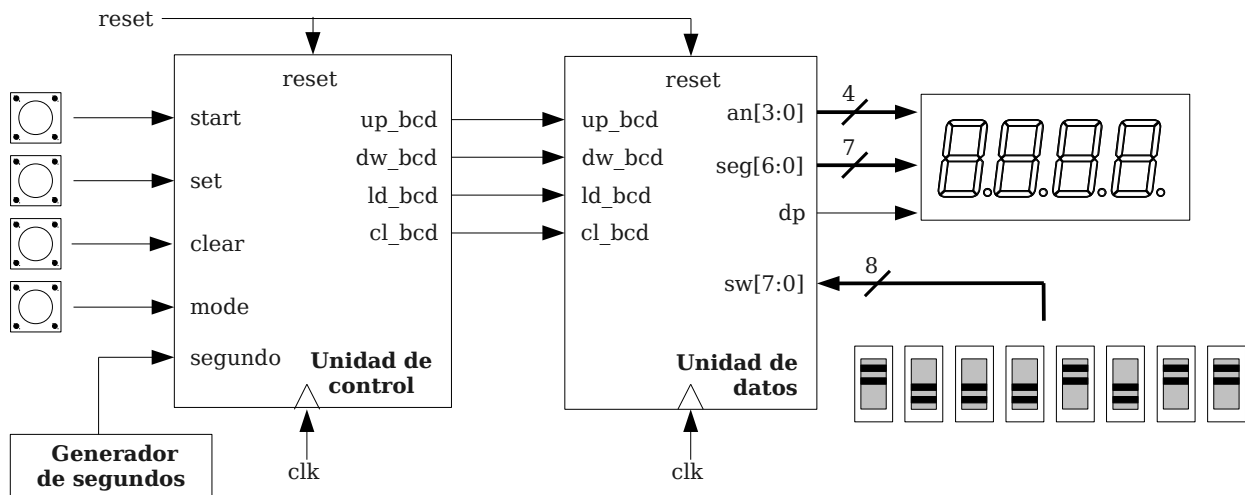


Figura 20. Conexión de la unidad de datos y de control del cronómetro.

3.2. Diseño de la Unidad de Datos

El componente principal de la unidad de datos de este sistema digital es el contador sexagesimal, el cual, se conectará adecuadamente al controlador de display para conseguir mostrar los minutos y segundos. Para el uso en modo temporizador se utilizarán los conmutadores con lo que se consigue introducir el número de minutos deseados en la cuenta atrás. En la figura 21 se muestra el diagrama de bloques de la unidad de datos a diseñar.

Se propone que realice el diseño siguiendo los siguientes pasos:

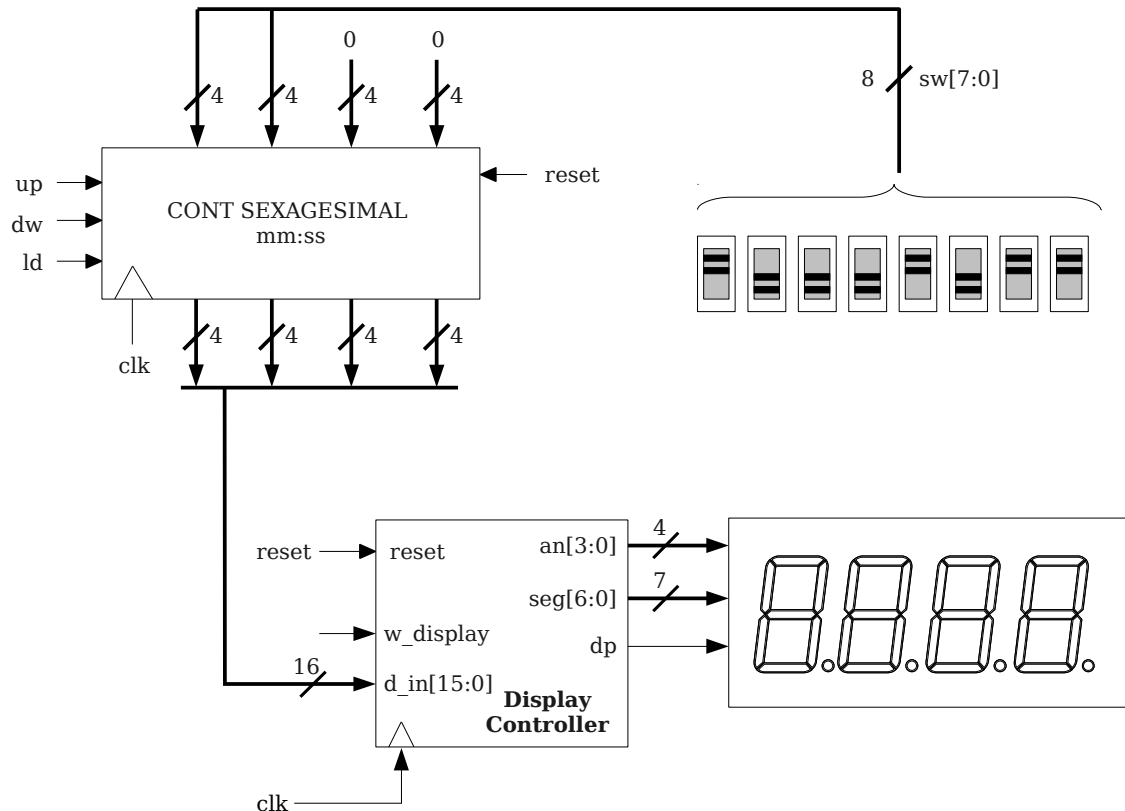


Figura 21. Diagrama de bloques de la Unidad de Datos

Añada al proyecto los componentes del módulo controlador de display diseñado en el laboratorio anterior. No añada el módulo principal ni el fichero UCF.

Debe modificar el módulo *display* para que admita carga de datos del exterior mediante un BUS. El interfaz del módulo debe quedar:

```
module display( input extclk, reset, w_display,
  input  [15:0] d_in,
  output [6:0] seg,
  output [3:0] an);
```

16.5. Modifique el modo de escritura en la memoria del display. La escritura debe ser únicamente disparada por el reloj externo en el flanco positivo, el resto de señales no debe aparecer en la lista de sensibilidad.

17. Cree un nuevo módulo llamado *unidad_datos* que incluya todas las señales mostradas en la figura 20.

17.1. Instancie el módulo *display*, el módulo *contador*, interconecte los dos componentes y la E/S del módulo correctamente.

17.2. Al añadir el contador sexagesimal diseñado anteriormente considere que sólo tiene una señal *reset*. Debe conectarla correctamente ya que la unidad de datos recibe dos señales *reset* y *cl_bcd*, ambas inicializan el contador.

17.3. La señal de escritura del display debe fijarla a 1 para conseguir escritura continua en el display.

3.3. Diseño de la Unidad de Control

El diseño de la unidad de control consiste en la implementación de una máquina de estados finita encargada de activar secuencialmente las señales de la unidad de datos y, consiguiendo el funcionamiento deseado.

El cronómetro a diseñar dispondrá de dos modos de funcionamiento, uno ascendente (cronómetro) y otro descendente (temporizador). El diagrama de estados de la figura 22 muestra los estados y las señales a activar en cada uno de los estados. La máquina de estados funcionará con el reloj del sistema pero recibe una señal denominada segundo proveniente de un generador de pulsos a intervalos de 1 segundo.

No se confunda con el generador de pulsos, éste genera un pulso de un ciclo de reloj de duración cada segundo, no una señal cuadrada de un segundo de período.

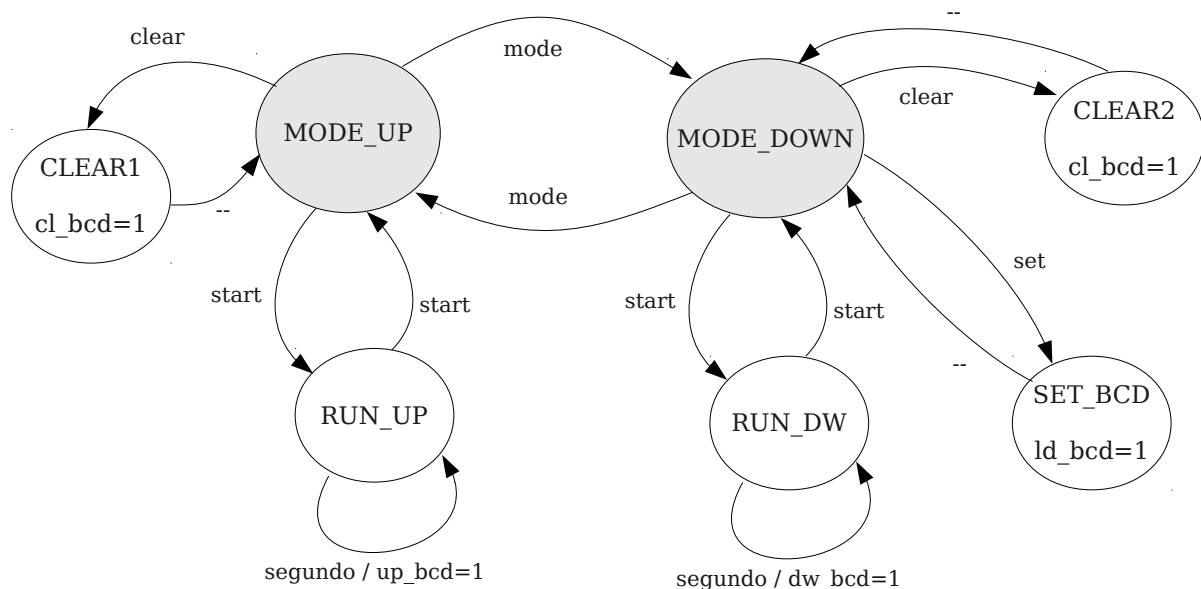


Figura 22. Diagrama de estados de la Unidad de Control.

18. Cree un nuevo módulo llamado *unidad_control* con las conexiones E/S indicadas en la figura 20 e implemente la máquina de estados de la figura 22.

18.1. Asigne al estado MODE_UP el valor binario 000.

Utilice el testbench del fichero *unidad_control_tb.v* para comprobar la máquina de estados opera correctamente.

4. Implementación del sistema completo

El último paso de diseño consiste en realizar módulo superior donde aparecen la unidad de datos y de control interconectada junto a los componentes E/S adicionales.

19. Cree un nuevo módulo llamado *cronometro* con las siguientes características:
- 19.1. Las señales E/S son: input `start,set,clear,mode,clk`, output `[3:0] an`, output

[6:0] seg, input [7:0] sw, output dp

19.2. Añada una instancia de la unidad de datos y otra de la de control, interconectada correctamente.

19.3. Para conectar los pulsadores a la unidad de control se utilizará el detector de flanco diseñado al principio de este laboratorio. Añádalo al proyecto e instancie el detector 4 veces, una para cada pulsador. Debe conectar cada una de las 4 salidas del detector a las señales de entrada de la unidad de control.

19.4. Al tener el sistema 4 entradas y solo disponer de 4 botones en la placa de desarrollo, se propone generar señal *reset* mediante la pulsación de dos botones simultáneos. Declare un cable interno en el módulo *cronometro* y mediante una sentencia *assign* active *reset* cuando el primer y el último botón estén ambos pulsados.

19.5. Utilice el fichero suministrado *segundero.v*. Este fichero contiene un generador de un pulso de reloj cada segundo en base a un reloj de 50Mhz. Debe instanciarlo y conectar su salida a la señal de entrada de la Unidad de Control llamada *segundo*.

19.6. Por último se recomienda eliminar el divisor de frecuencia existente en el módulo *display*. Debe sustituirlo por un módulo similar al que acaba de añadir (*segundero.v*). Con esto evitará dejar señales del contador sin conectar.

20. Para la implementación conecte los botones como se indica en la figura 23. Implemente el diseño y utilice la combinación de botones correcta para generar señal *reset* en el sistema.

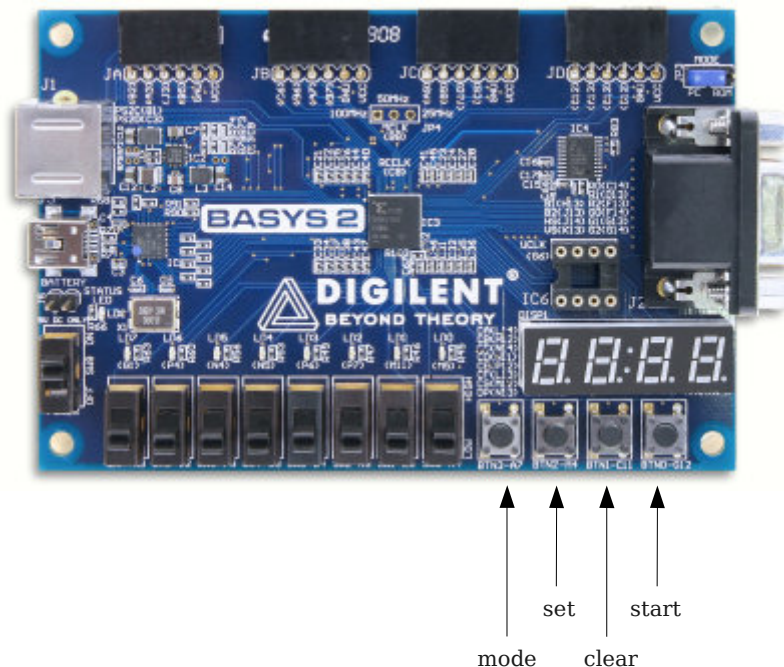


Figura 23. Diagrama de estados de la Unidad de Control.



DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Diseño de una calculadora

Sistemas Digitales Avanzados

1. Introducción y objetivos

El propósito general de esta sesión de laboratorio es la realización de un sistema digital partiendo de las especificaciones. Con las habilidades adquiridas con los desarrollos anteriores se deben cubrir los siguientes objetivos:

Diseño de un multiplicador secuencial mediante la división del sistema digital en Unidad de Datos y Unidad de Control. En particular, el sistema propuesto realiza la multiplicación de dos números binarios A y B de ocho bits mediante el algoritmo basado en sumas y desplazamientos a la derecha.

- Partiendo de un esquema de la Unidad de Datos deberá implementarla realizando el diseño de la Unidad Control completa. Previo estudio y comprensión del funcionamiento de algoritmo de multiplicación propuesto debe obtener un diagrama de estados válido.
- Deberá realizar un testbench del multiplicador donde se comprueben varias multiplicaciones.
- Realizará la implementación añadiendo los componentes diseñados en las sesiones anteriores como son el controlador de display, el detector de flancos, etc.

Finalmente debe ampliar el sistema para realizar más operaciones aritméticas convirtiéndolo en una calculadora.

2. Descripción del multiplicador secuencial

El primer paso del desarrollo consiste en el diseño de un multiplicador secuencial para posteriormente ampliar el sistema con otras operaciones aritméticas. En la implementación del sistema deberá utilizar módulos desarrollados en las sesiones anteriores para controlar la entrada y salida.

Los componentes de entrada salida deben utilizarse de la siguiente forma:

- Display: Mostrará siempre datos de 16 bits, cuando finalice la operación mostrará el resultado final, pero, durante la introducción de datos desde los conmutadores debe mostrar los datos entrantes.

Conmutadores: Se utilizarán los 8 conmutadores para introducir datos de 8 bits en el sistema.

Se utilizarán al menos 3 botones con la siguiente funcionalidad:

- Reset: Realiza la inicialización completa del sistema.
- Modo: Cambio del modo de operación.

Siguiente: Utilizada reiteradamente para introducir los operandos de la operación paso a paso.

Leds: Se utilizarán para mostrar el estado del sistema, se recomiendan su utilización para indicar el modo de operación.

Con esta configuración E/S se construirá un sistema digital como el indicado de manera esquemática en la figura 24. Esta figura muestra la calculadora conectada a los componentes de la placa donde la señal *siguiente* controla los pasos para que el usuario introduzca datos. Un posible ejemplo de uso sería:

21. El usuario realiza una pulsación en el botón *reset* y el sistema se inicializa.
22. Pulsando el botón *modo* se conmuta entre las diferentes operaciones disponibles en el sistema: suma, resta, multiplicación, etc. Un Led diferente se iluminará para cada uno de los modos así, el usuario conocerá el modo seleccionado en cada momento.
23. Tras seleccionar el modo se establece el primer número en binario en los conmutadores, una vez establecido, se realiza una pulsación en el botón *siguiente* y el sistema almacena el número.

Se repite el proceso anterior para establecer el segundo número y al pulsar el botón *siguiente* se realiza la operación mostrándose el resultado en el display.

Para la realización de este multiplicador se divide el diseño en dos bloques: Unidad de datos y Unidad de control. Ambos bloques se representan en la figura 25 interconectados y aparecen las señales de control necesarias para manejar la Unidad de Datos. Estas señales provienen de la estructura propuesta para la Unidad de Datos y mostrada en la figura 26.

Para comprender la estructura propuesta de la Unidad de Datos es necesario estudiar el algoritmo de multiplicación de sumas y desplazamiento detallado en la siguiente sección. Debe establecer una secuencia correcta de señales de control para la Unidad de Datos de forma que realice la multiplicación.

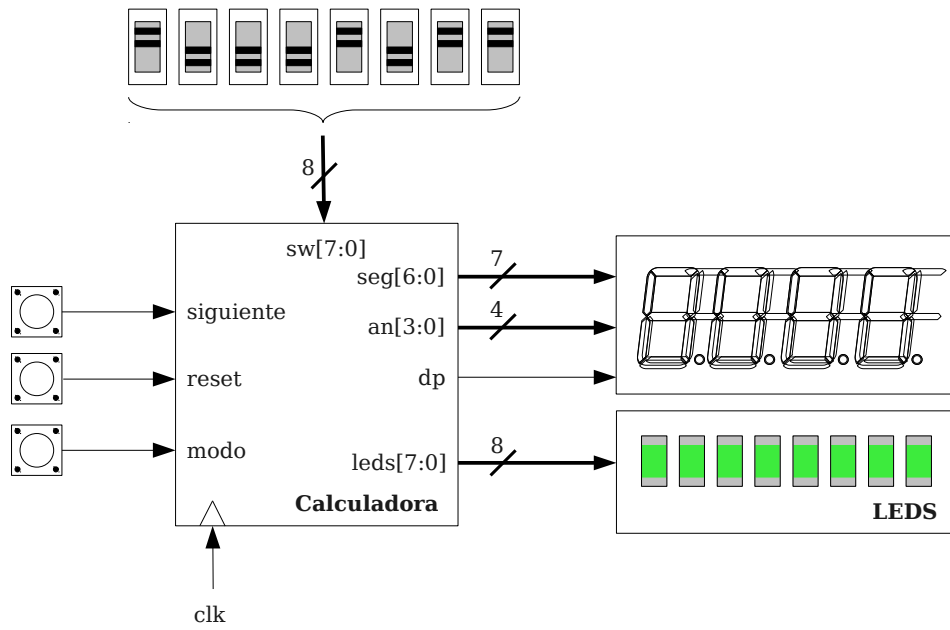


Figura 24. Esquema del sistema digital completo para la calculadora.

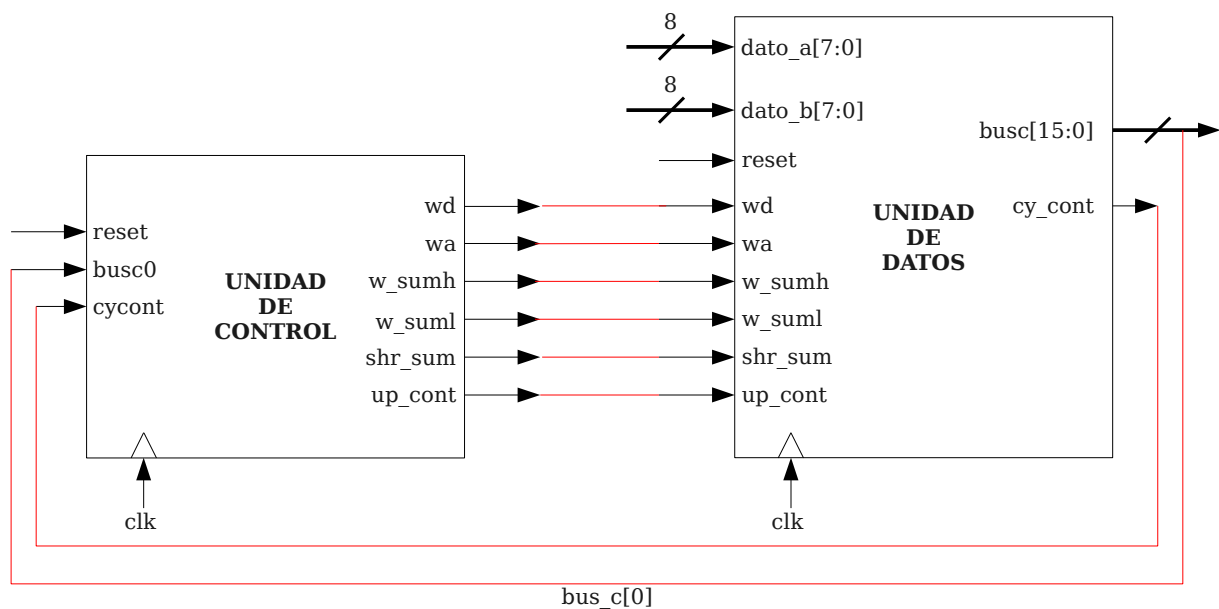


Figura 25. Representación a nivel de bloques del multiplicador secuencial.

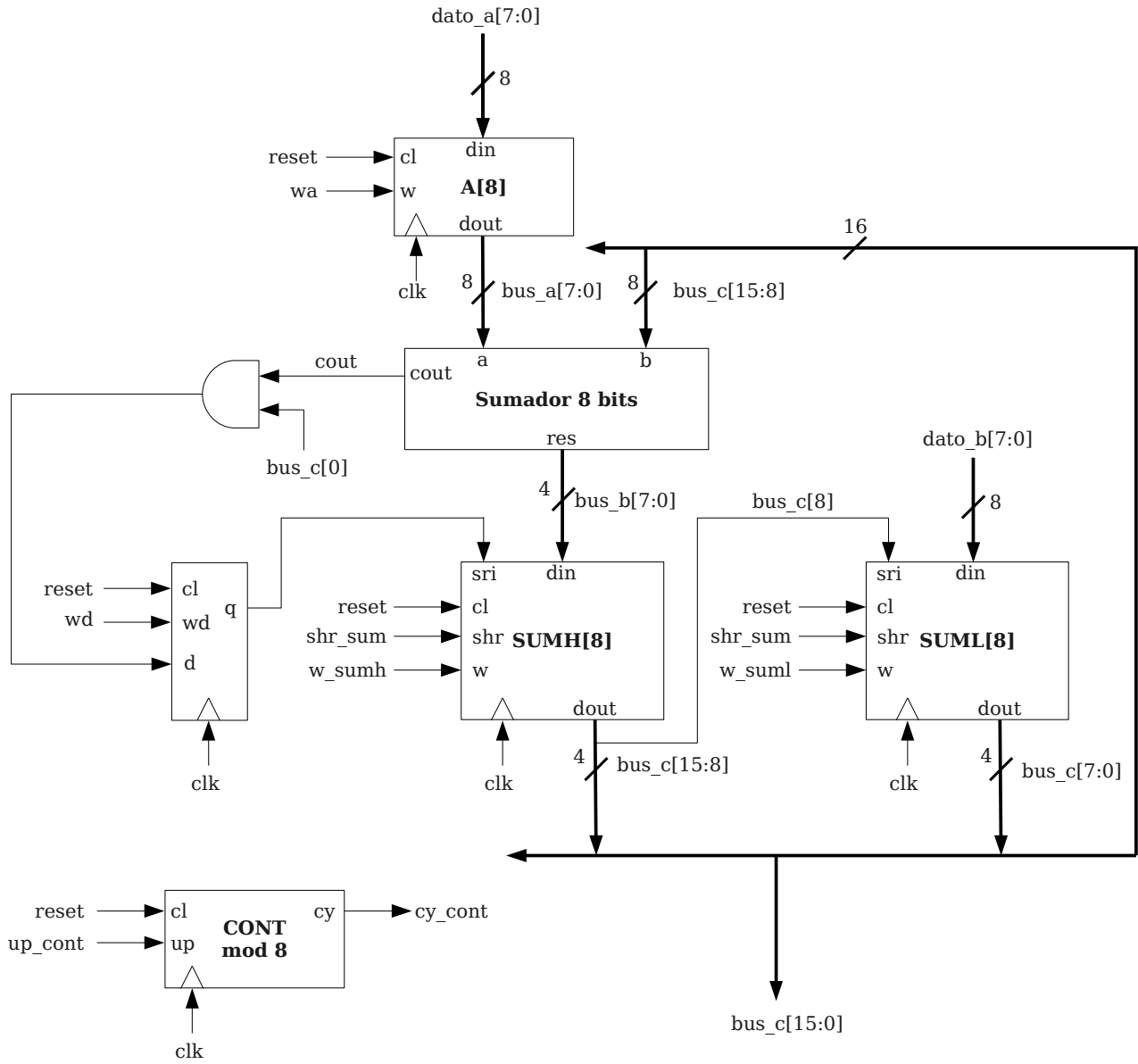


Figura 26. Representación estructural de la unidad de datos.

clk	cl	w	Operación
\uparrow	1	-	$A \leftarrow 0$
\uparrow	0	1	$A \leftarrow \text{dato_a}[7:0]$
\uparrow	0	0	$A \leftarrow A$

Registro A

clk	cl	up	Operación
\uparrow	1	-	$\text{CONT} \leftarrow 0$
\uparrow	0	1	$\text{CONT} \leftarrow \text{CONT} + 1$
\uparrow	0	0	$\text{CONT} \leftarrow \text{CONT}$

Contador

clk	cl	w	shr	Operación
\uparrow	1	-	-	$R \leftarrow 0$
\uparrow	0	1	-	$R \leftarrow \text{Din}[7:0]$
\uparrow	0	0	1	$R \leftarrow \text{SHR}(R, \text{sri})$
\uparrow	0	0	0	$R \leftarrow R$

Registro de desplazamiento

clk	cl	w	Operación
\uparrow	1	-	$D \leftarrow 0$
\uparrow	0	1	$D \leftarrow \text{Din}$
\uparrow	0	0	$D \leftarrow D$

Bistable

Figura 27. Descripción de los componentes de la unidad de datos.

2.1. Algoritmo de multiplicación

Para realizar la multiplicación de dos números binarios de 8 bits, A y B, se utilizará el algoritmo de sumas y desplazamientos. La idea básica de este algoritmo es la siguiente:

1. Se cargan los datos A y B en los registros A y SUML respectivamente.
2. Se ha dispuesto de un contador módulo-8 para contar los 8 desplazamientos necesarios. Será el bit de acarreo de este contador el que indique cuándo finaliza la multiplicación. Debe iniciarse este contador a cero.
3. Se procede a analizar el bit menos significativo de B (SUML[0]). Según sea 0 o 1, se realiza una suma del dato de A con el dato presente en un tercer registro (SUMH).
4. Tras esto, se realiza una operación de desplazamiento a la derecha de los registros SUMH y SUML conjuntamente:
 - 4.1. SUMH desplaza su bit LSB a SUML así, en la próxima operación de suma este bit no debe sumarse.
 - 4.2. Este bit desplazado pasa a convertirse en el siguiente bit del resultado.
 - 4.3. Se decrementa el contador
5. Se realizarán los pasos 3 y 4, 8 veces ya que el dato B posee 8 bits. Finaliza si el contador llega a cero.

Al finalizar, la concatenación de los registros SUMH y SUML contiene el resultado de la multiplicación.

Para comprender este algoritmo en la figura 28 se ha realizado un ejemplo de multiplicación de 8 bits mediante este procedimiento y paso a paso. Observe como en cada paso se analiza un bit del operando

B. Siempre que encuentre un 1 realiza una suma y un desplazamiento en 2 pasos. En caso de ser este bit 0, se realiza únicamente el desplazamiento en un único paso.

$$\begin{array}{r}
 10101010 \\
 \times 01010101 \\
 \hline
 10101010 \\
 00000000 \\
 10101010 \\
 00000000 \\
 10101010 \\
 00000000 \\
 10101010 \\
 00000000 \\
 \hline
 11100001110010
 \end{array}$$

Procedimiento de multiplicación manual

Algoritmo de sumas y desplazamientos

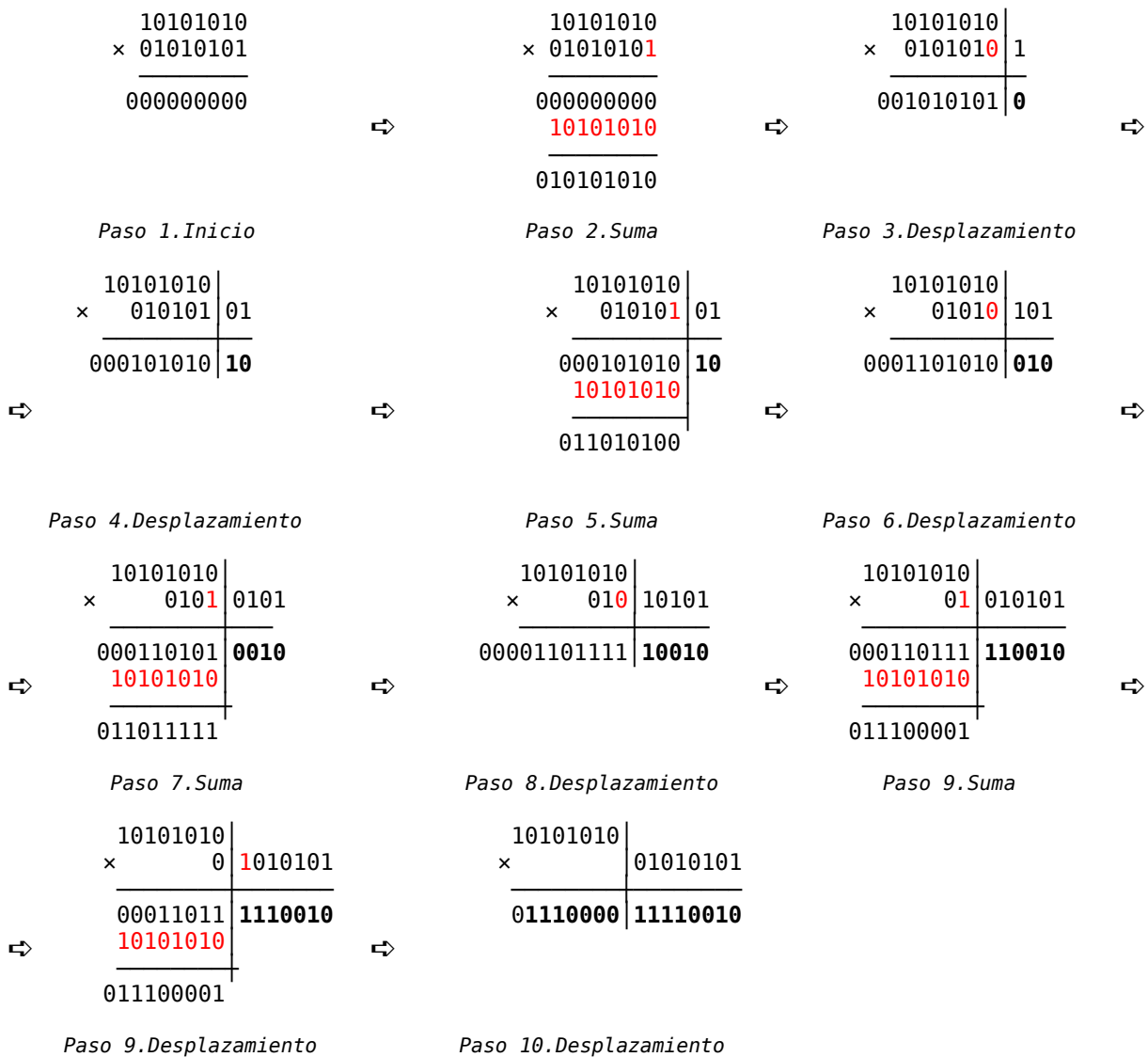


Figura 28. Algoritmo de multiplicación por suma y desplazamiento.

3. Desarrollo de la sesión de laboratorio

En la sección anterior se ha descrito el multiplicador como Sistema Digital y el algoritmo de multiplicación. El multiplicador consta de dos partes (Unidad de Datos y Unidad de Control) que deberá desarrollar. Concretamente debe realizar las siguientes tareas:

1. Diseñar todos los componentes de la unidad de datos e interconectarlos correctamente.
2. Proponer un diagrama de estados que implemente el algoritmo de multiplicación sobre la ruta de datos diseñada anteriormente y realizar su diseño en Verilog.
3. Realizar la unión estructural de la Unidad de Datos y la Unidad de Control junto con un testbench que compruebe las multiplicaciones. Este testbench debe comprobar al menos 16 multiplicaciones aleatorias mediante un bucle FOR.
4. Completar la descripción estructural del multiplicador y simular el multiplicador para verificar el correcto funcionamiento.
5. Realizar la implementación del sistema completo en FPGA y verificar su funcionamiento manualmente.
6. Añada alguna funcionalidad más al multiplicador convirtiendo el sistema en una calculadora con varias operaciones, al menos, suma y resta. Estas operaciones se seleccionarán mediante un botón MODO. El modo de operación debe ser visible en los LEDs de la placa de desarrollo.

4. Memoria del trabajo realizado.

Esta última sesión de laboratorio concluye con la entrega por parte del alumno de una memoria donde se debe incluir:

- Diagrama de estados de la unidad de control diseñada.
- Simulación de las multiplicaciones.
- Datos de utilización de recursos en la FPGA utilizada.
- A partir de los datos proporcionados por el proceso de síntesis debe calcularse la frecuencia máxima de operación en MIPS del multiplicador.

Junto con la entrega de la memoria deberá entregarse el proyecto ISE realizado.