DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Multiplicador Secuencial 4x4

Enunciados de Prácticas de Laboratorio
Estructura de Computadores

1. Introducción y objetivos

El propósito general de esta sesión de laboratorio es operar con sistemas digitales modernos. Este propósito se concreta en los siguientes objetivos:

- Conocer cómo opera un sistema digital con la estructura “Unidad de Datos y Unidad de Control”. En particular, el sistema propuesto realiza la multiplicación de dos números binarios A y B de cuatro bits mediante el algoritmo basado en sumas y desplazamientos a la derecha.

- Desarrollar las habilidades de diseño e implementación. Para ello el alumno deberá describir en Verilog el multiplicador, de forma estructural, como la interconexión de una unidad de datos y una unidad de control. La unidad de datos se le proporciona completamente especificada, mientras que la unidad de control deberá completarla el alumno a partir de la carta ASM “de control” que se le facilita totalmente terminada.

- Implementar el multiplicador, para lo cual se volcará el diseño del sistema digital sobre un chip FPGA instalado en una placa de desarrollo.

- Comprobar la implementación del multiplicador realizada sobre la FPGA, tanto a nivel de microoperaciones como a nivel de macrooperación usando los pulsadores, conmutadores y el display de la placa de desarrollo.

Para la realización de esta práctica de laboratorio se facilitan un conjunto de ficheros. La tabla 1 resume el contenido y el objetivo de cada uno de ellos.

Nombre del fichero	Contenido	Descripción
u_control.v	Unidad de control del multiplicador.	Debe completarlo el alumno durante la sesión de laboratorio.
u_datos.v	Unidad de datos del multiplicador.	Debe utilizarlo sin modificaciones para conectarlo a la unidad de control.
multiplicador.v	Multiplicador secuencial formado por unidad de datos y unidad de control.	Debe completar la descripción estructural de este módulo durante la sesión de laboratorio.
multiplicador_tb.v	Testbench para multiplicador.	Debe utilizarlo sin modificaciones para realizar la simulación.
sistema_completo.v	Sistema completo para implementar en el chip FPGA.	Se utilizará sin modificaciones durante la fase de implementación.
basys2.ucf	Fichero con la relación entre las señales del sistema y los pines del chip FPGA de la placa de desarrollo *BASYS2*.	Se utilizará sin modificaciones durante la fase de implementación.

Tabla 1. Ficheros necesarios durante la sesión de laboratorio.

2. Descripción del multiplicador secuencial

El multiplicador secuencial tiene la estructura típica de un sistema digital organizado en unidad de datos y unidad de control.

Desde el punto de vista de la descripción Verilog, el multiplicador será un módulo descrito de forma estructural, interconectando una instancia de la unidad de control y una instancia de la unidad de datos, tal y como muestra la figura 1.

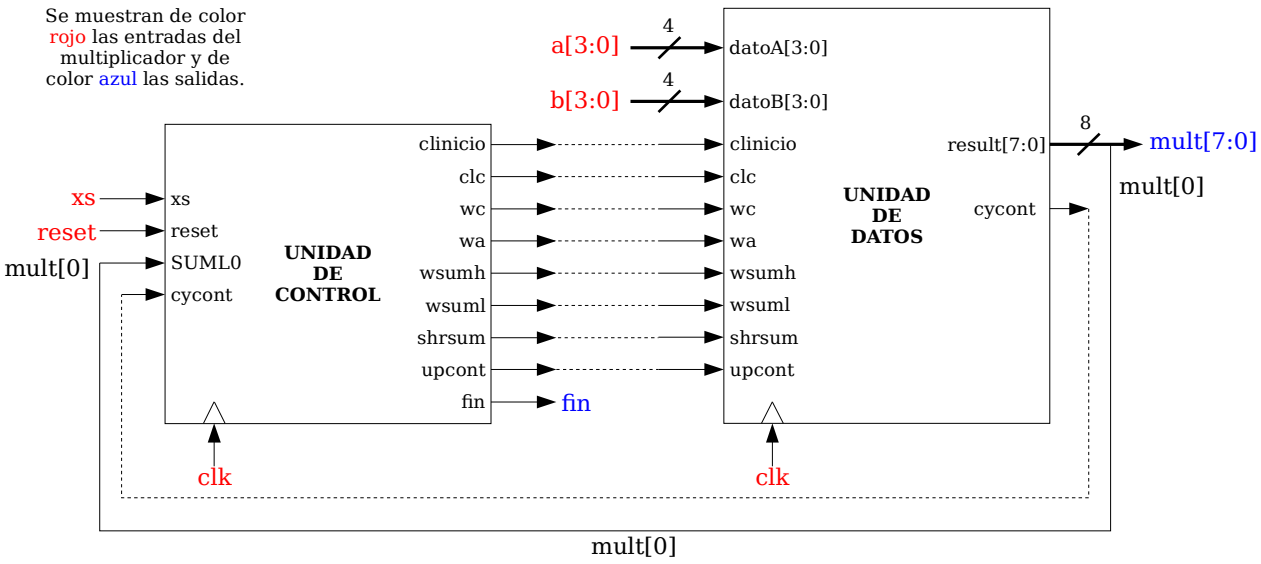


*Figura 1. Representación estructural del módulo Verilog **multiplicador**.*

Desde el punto de vista de un usuario del multiplicador, el funcionamiento sería el siguiente:

- Antes de empezar a usarlo por primera vez es necesario activar la señal de 'reset', que es asíncrona, para llevar la unidad de control al estado inicial. A partir de ese momento es posible solicitarle al multiplicador que haga las multiplicaciones que se deseen.

- Para efectuar una multiplicación el usuario coloca en las entradas 'a' y 'b' los dos números a multiplicar y activa la entrada 'xs' durante un ciclo de reloj, para indicarle al multiplicador que debe empezar el proceso de multiplicación.

- A partir de este momento, durante varios ciclos de reloj, la unidad de control va activando las señales correspondientes para que los elementos de la unidad de datos vayan ejecutando las diversas micro-operaciones en las que se ha descompuesto la operación de multiplicación.

- Cuando la unidad de control da por concluido el proceso de multiplicación, ésta activa la salida 'fin' para que el usuario sepa que la multiplicación ha terminado y que el dato que hay colocado en la salida 'mult' es el resultado correcto de la misma.

2.1. Algoritmo de multiplicación basado en sumas y desplazamientos

Para realizar la multiplicación de dos números binarios de 4 bits, A y B, el multiplicador va a seguir un algoritmo basado en sumas y desplazamientos a la derecha. A continuación se explica en detalle este procedimiento.

Si nos fijamos en la parte superior de la figura 2, en el que se muestra la multiplicación de dos números binarios (A=1110 y B=1010), vemos que el procedimiento de multiplicación tradicional consiste en realizar una suma cuyos sumandos se van desplazando a la izquierda. Además, al tratarse de datos binarios, los sumandos de esta suma sólo pueden ser el primer operando de la multiplicación (el número A) o bien el cero.

El algoritmo de multiplicación de sumas y desplazamientos propuesto consiste en obtener el resultado final de la suma mediante un proceso iterativo en el que se van haciendo pequeñas sumas simples (sumas parciales), de las cuales se va obteniendo un resultado parcial que sirve de punto de partida a la siguiente iteración del proceso. En la última iteración el resultado parcial obtenido es ya el resultado final.

En la parte inferior de la figura 2 se muestra un ejemplo de multiplicación de dos números binarios (A=1110 y B=1010) siguiendo paso a paso el algoritmo de sumas y desplazamientos.

1. **Paso 1:** En el paso 1 se le dará un valor inicial a los 8 bits del resultado parcial. Los cuatro primeros se ponen a 0000 y los cuatro últimos son el valor del operando B (fíjese más adelante en que estos últimos cuatro bits se 'pierden' todos tras acabar el algoritmo). Tras este paso 1 se va a entrar en un proceso iterativo en el que se realizarán los pasos 2 y 3 cuatro veces seguidas, trabajando sobre el resultado parcial obtenido en los pasos anteriores.
2. **Paso 2:** En el paso 2 se analizará el valor de uno de los bits del operando B (el que se muestra subrayado en cada uno de los 'pasos 2' de la figura 2). Este bit lo tenemos siempre disponible en

la posición menos significativa del resultado parcial de 8 bits. Dependiendo de si el valor del bit analizado (el subrayado) es 0 o es 1, en el paso 2 se hará uno de los dos acciones siguientes:

2.1 Si el bit analizado del dato B es 1, se le suma el valor del operando A a los cuatro bits más significativos del resultado parcial de 8 bits. Esto nos da un resultado parcial de 9 bits, donde el bit más significativo es el bit de carry de la suma.

2.2 Si el bit analizado del dato B es 0, se le suma 0000 a los cuatro bits más significativos del resultado parcial de 8 bits. Esto nos da un resultado parcial de 9 bits, donde el bit más significativo es el bit de carry de la suma, que siempre será 0, pues sumar 0000 nunca provoca acarreo. Hay que darse cuenta de que en realidad en este caso nos podríamos ahorrar hacer la suma y, simplemente, limitarnos a añadir directamente un 0 por la izquierda al resultado parcial de 8 bits para obtener el resultado parcial de 9 bits.

3. **Paso 3:** En el paso 3 lo único que se hace es desplazar un bit a la derecha los 9 bits del resultado parcial del paso 2. Esto hace que se pierda un bit (el bit del operando B que se consultó en el paso 2 anterior) y que volvamos a tener un resultado parcial de 8 bits ubicado en la posición en que se encontraba originalmente el resultado parcial de 8 bits que teníamos antes del paso 2. Si ya hemos hecho 4 veces los pasos 2 y 3, el resultado parcial es el resultado final de la multiplicación y si no pues se vuelven a repetir otra vez los pasos 2 y 3 hasta llegar a 4 veces.

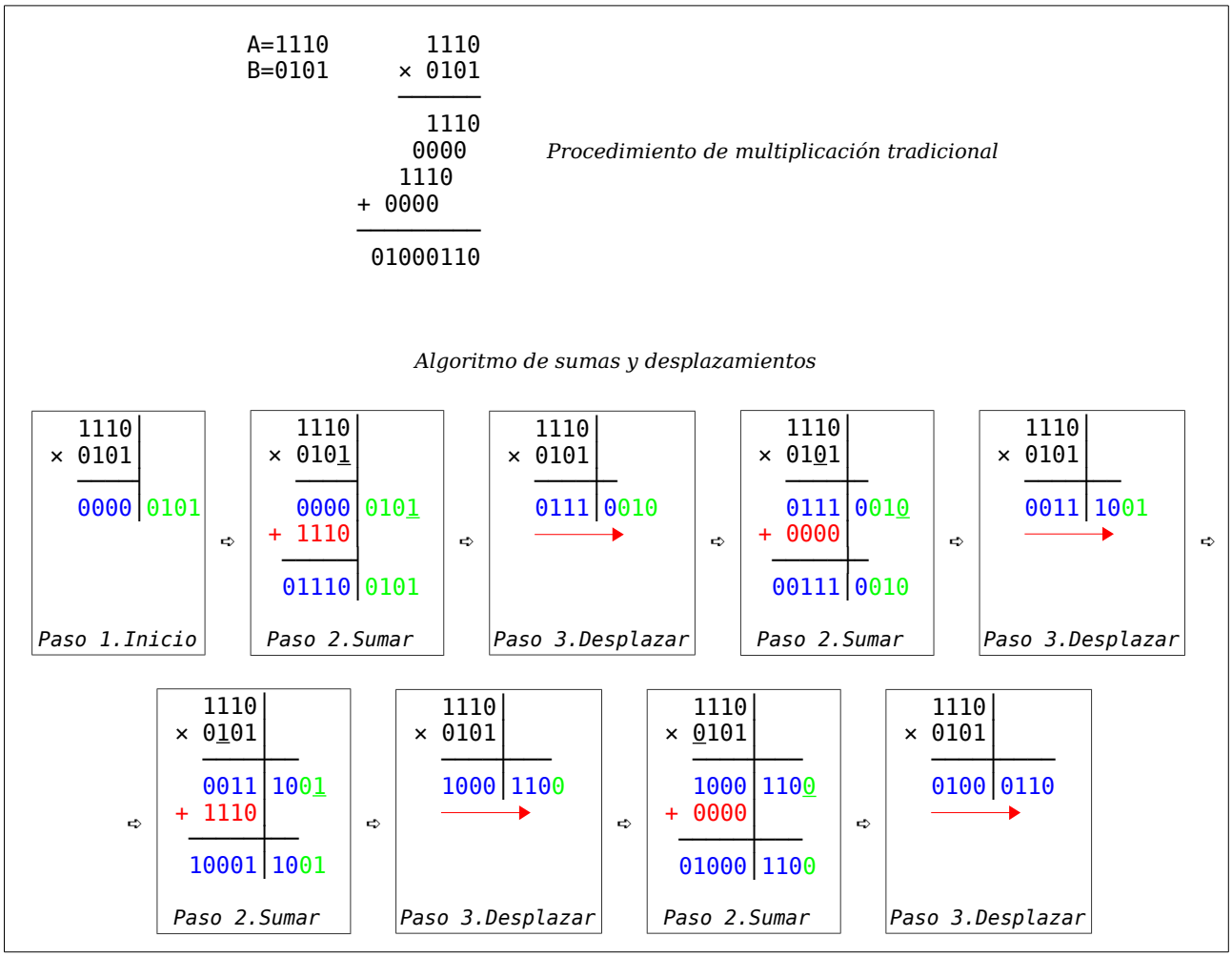


Figura 2. Algoritmo de multiplicación mediante sumas y desplazamientos.

2.2. Unidad de datos del multiplicador

La unidad de datos propuesta para un multiplicador secuencial basado en el algoritmo de sumas y desplazamientos descrito en la sección 2.1 se muestra en la figura 3.

Se dispone de un registro de un bit, llamado C, y dos registros de desplazamiento de 4 bits, llamados SUMH y SUML. La pareja de registros SUMH y SUML contendrá el resultado parcial del algoritmo cuando este se componga de 8 bits (tras el paso 1 y tras el paso 3 del algoritmo). El registro C contendrá el noveno bit del resultado parcial, cuando éste se componga de 9 bits (tras el paso 2 del algoritmo). Los tres registros juntos van a permitir hacer el desplazamiento de un bit descrito en el paso 3 del algoritmo.

Otros elementos importantes son un registro A, de 4 bits, para contener al operando A, y un sumador de 4 bits para sumar A con SUMH, tal y como dice el algoritmo que hay que hacer.

Por último, se tiene un contador de módulo 4, llamado CONT, que va a permitir llevar la cuenta de las 4 iteraciones que dice el algoritmo que hay que realizar sobre los pasos 2 y 3 del mismo.

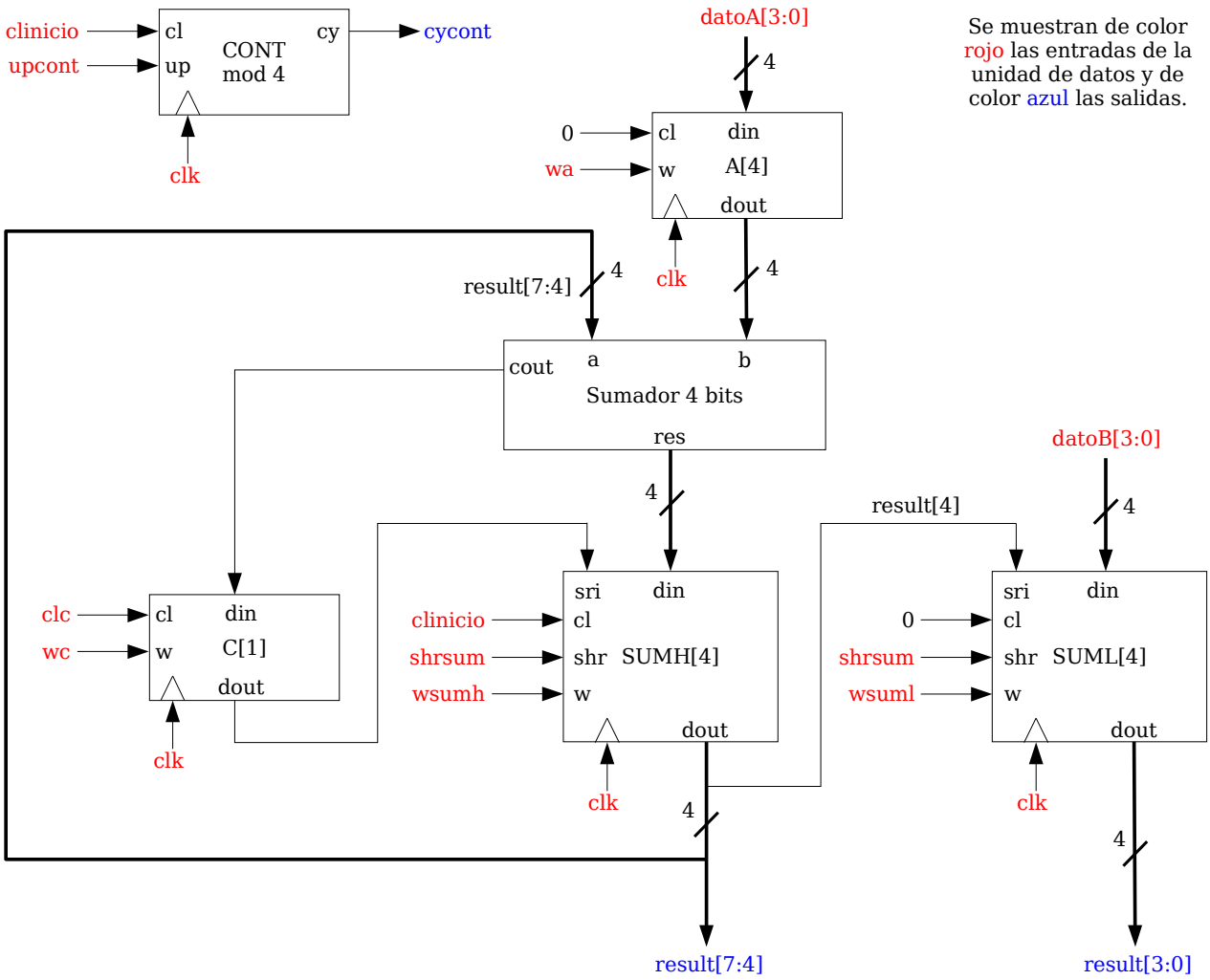


Figura 3. Representación estructural del módulo Verilog ***u_datos*** *(unidad de datos del multiplicador).*

La descripción RT de los elementos de la figura 3 se muestra en la figura 4.

clk	cl	w	Operación de escritura
↑	1	0	R ← 0
↑	x	1	R ← din
↑	0	0	R ← R

Registros con carga en paralelo (A y C)

clk	cl	up	Operación de escritura
↑	1	0	CONT ← 0
↑	x	1	CONT ← CONT+1
↑	0	0	CONT ← CONT

Contador módulo 4

clk	cl	w	shr	Operación de escritura
↑	1	0	0	R ← 0
↑	x	1	0	R ← din
↑	x	x	1	R ← SHR(R,sri)
↑	0	0	0	R ← R

Registros de desplazamiento (SUMH y SUML)

Todas las operaciones de escritura son síncronas, incluso las de puesta a cero (señal de control 'cl').

Todas las operaciones de lectura son incondicionales por lo que ninguna salida es tri-estado.

Los registros y el contador muestran siempre en su bus de salida el valor actual del dato almacenado.

El contador muestra, además, un 1 en su salida 'cy' cuando el dato almacenado es 1111 y un 0 en otro caso.

Figura 4. Descripción RT de los componentes de la unidad de datos.

2.3. Unidad de control del multiplicador

La unidad de control del multiplicador es la encargada de llevar a cabo el algoritmo de sumas y desplazamientos descrito en la sección 2.1. Para ello, la unidad de control debe comunicarse con la unidad de datos propuesta (ver figura 3), recibiendo señales de estado y enviándole señales de control.

Concretamente, el comportamiento de la unidad de control, una vez recibida la señal 'xs' de puesta en marcha, podría ser el siguiente:

1. **Paso 1:** Ordenar la puesta a cero inicial del registro registro SUMH (tal y como requiere el algoritmo) y del contador CONT (para empezar desde cero la cuenta de las 4 iteraciones). En este mismo paso se realiza una carga en paralelo de los datos *datoA* y *datoB* en los registros A y SUML respectivamente. Al acabar el paso 1 tendremos en SUMH y SUML el primer resultado parcial de 8 bits sobre el que hacer las 4 iteraciones.

2. **Paso 2**: Analizar el valor de uno de los bits del *datoB* (el bit que está actualmente en la posición menos significativa del registro SUML) y, en función de su valor, hacer lo siguiente:

   2.1. Si el bit es 1 se realiza una suma del dato del registro A con el dato presente en el registro SUMH, obteniéndose un resultado de 4 bits y un bit de carry. Los 4 bits del resultado se guardarán en el mismo registro SUMH y el bit de acarreo se guardará en el registro C.

   2.2. Si el bit es 0, escribiremos un 0 en el registro C. Esto es equivalente a sumar 0000 a SUMH, guardar el resultado de 4 bits en SUMH y guardar el acarreo (que será 0) en el registro C.

   En cualquiera de los dos casos anteriores, al acabar el paso 2 los registros C, SUMH y SUML contendrán los 9 bits del nuevo resultado parcial.

3. **Paso 3:** Tras el paso anterior, en el paso 3 se van a desplazar un bit a la derecha los 9 bits del resultado parcial de 9 bits contenido en los registros C, SUMH y SUML. Por tanto, al finalizar este paso, en SUMH y SUML tendremos almacenado el nuevo resultado parcial de 8 bits. Se da orden al contador CONT de que incremente su valor (cosa que hará al terminar este paso). Si el valor

actual de CONT en este paso es 3 (su carry vale 1) esto indica que ya hemos hecho 4 veces los pasos 2 y 3, así que se saltaría al paso 4 y de lo contrario se repetirían los pasos 2 y 3 nuevamente.

4. **Paso 4:** Venimos del paso 3, luego SUMH y SUML contienen un valor de 8 bits que es el resultado de la multiplicación de los datos *datoA* y *datoB*. Se activa la señal 'fin' para indicarle al usuario que la multiplicación ha concluido. Se vuelve a un estado de espera del cual no saldremos hasta que se produzca una nueva activación de la señal 'xs'.

Si describimos este mismo comportamiento de la unidad de control en lugar de con palabras mediante una carta ASM, el resultado sería la carta ASM mostrada en la figura 5.

A partir de esta carta ASM se puede obtener directamente la descripción procedimental en Verilog de la unidad de control (el módulo *u_control*), tarea que deberá realizar el alumno durante la sesión de laboratorio.

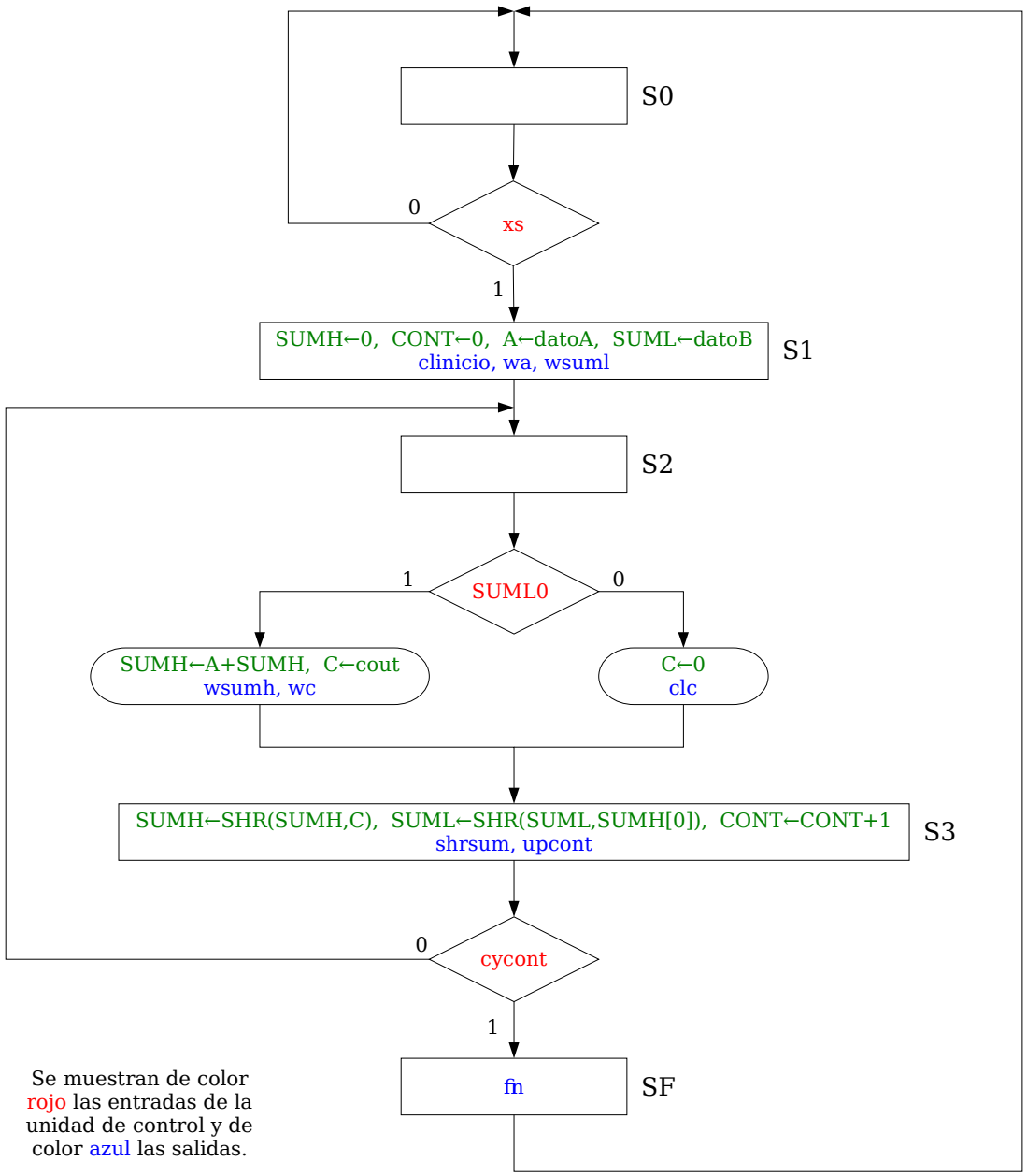


Figura 5. Carta ASM "de datos" y "de control" del multiplicador secuencial.

2.4. Sistema completo en la placa de desarrollo

Para probar el diseño digital realizado se utilizará una placa desarrollo del fabricante Digilent[1] llamada *BASYS2* que incluye un chip FPGA de 100.000 puertas (modelo XC3S100E) y algunos componentes electrónicos para poder realizar una entrada y salida básica, como por ejemplo 8 conmutadores, 4 displays de 7 segmentos, 8 leds y 4 pulsadores, entre otros. Para poder interactuar con el sistema utilizando la placa de desarrollo, es necesario añadir al diseño un procedimiento para introducir los datos a multiplicar, poder visualizar los resultados y consultar el estado del sistema.

Mediante varios módulos descritos en Verilog y ya preparados (fichero *sistema_completo.v*) se ha construido un sistema digital como el indicado de manera esquemática en la figura 6. Esta figura muestra el módulo *multiplicador* construido por el alumno, conectado a un módulo capaz de controlar el display 7 segmentos. Este controlador consta de 4 entradas de 4 bits (d3, d2 d1 y d0) correspondiéndose en ese orden con los 4 dígitos del display (de izquierda a derecha). Además, se ha conectado el reloj 'clk', la señal 'xs' y la señal 'reset' a tres pulsadores para poder controlar el funcionamiento del sistema manualmente. La señal 'fin' y el reloj llegan a un módulo que controla los leds para poder visualizar el efecto de la pulsación del botón asociado al reloj e indicar la finalización de la multiplicación. Por último, los datos A y B se pueden introducir en binario mediante los 8 conmutadores representados en la parte superior izquierda. Observe en la figura como los datos A=1110 y B=0101 codificados en los conmutadores se muestran en hexadecimal en las dos primeras posiciones del display como 'E' y '5'.

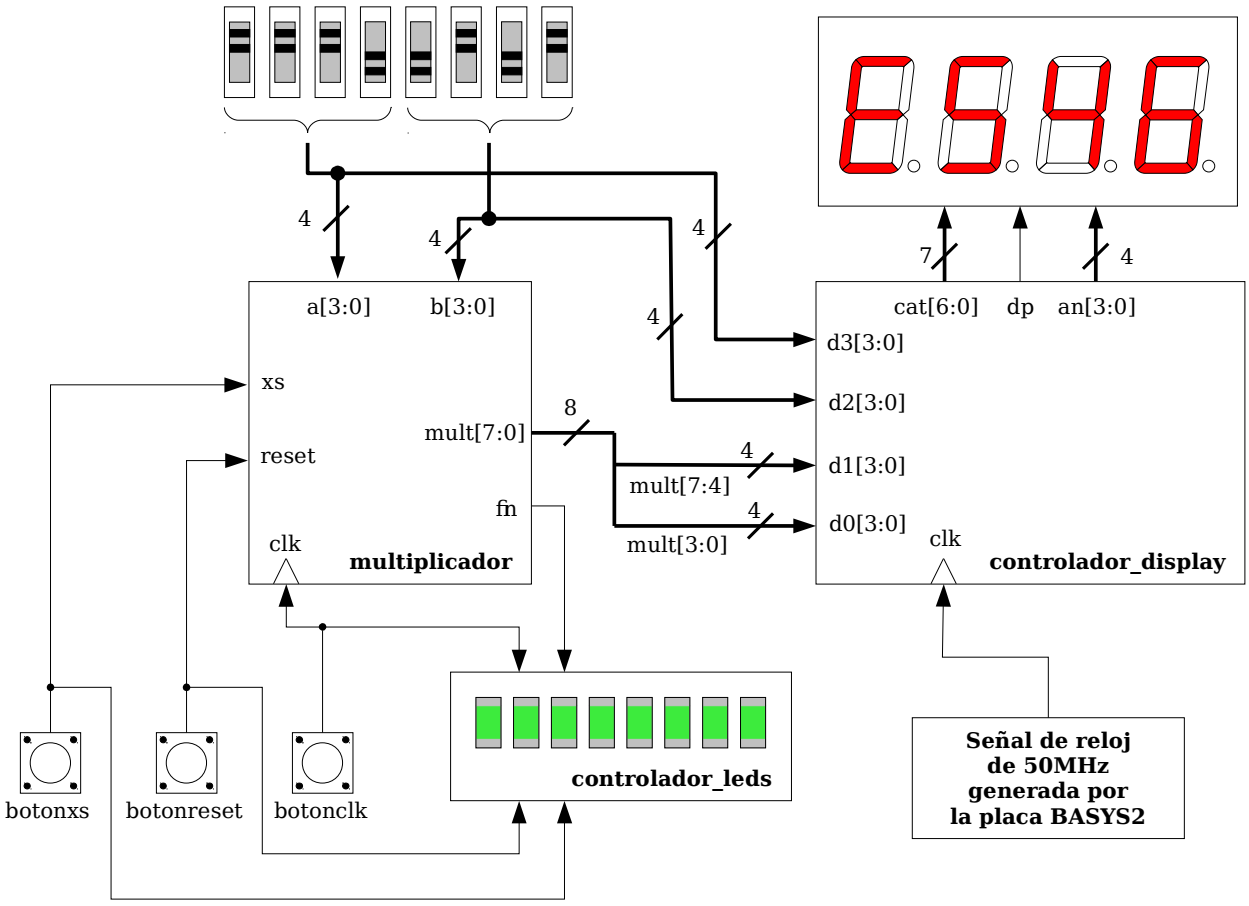


Figura 6. Esquema del módulo Verilog ***sistema_completo*** *conectado a los elemento de la placa BASYS2.*

1 Empresa dedicada al diseño en tecnologías basadas en FPGA y microcontroladores. Web: http://www.digilentinc.com

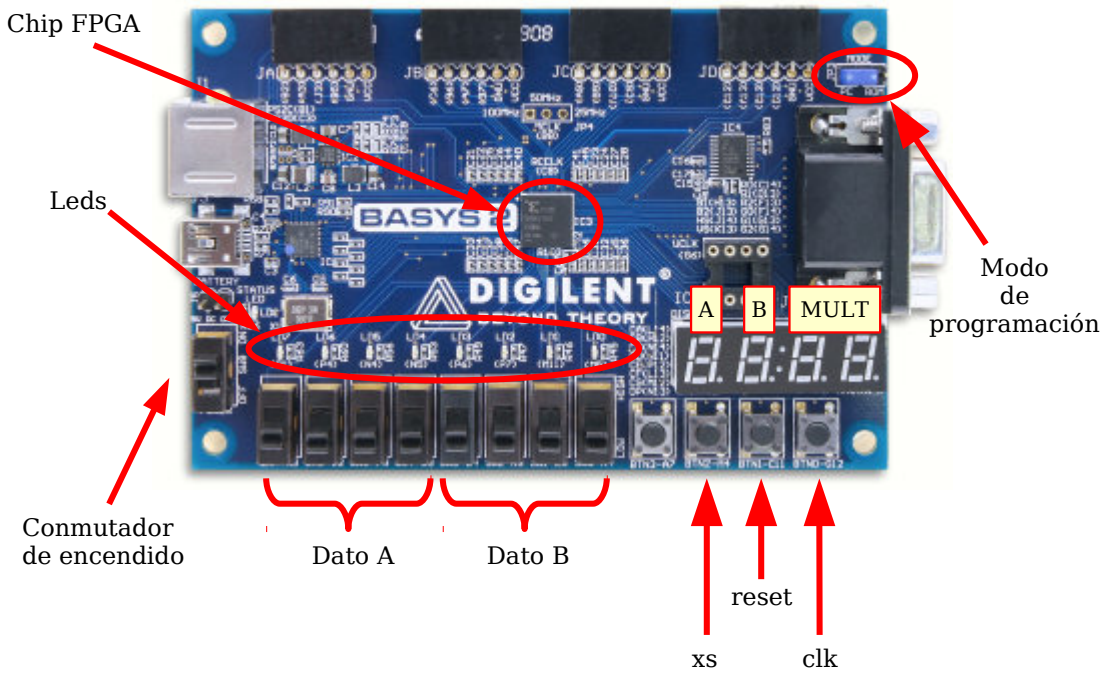


Figura 7. Fotografía de la placa de desarrollo BASYS2

La figura 7 muestra una fotografía de la placa de desarrollo donde puede ver el aspecto real de algunos de los elementos representados en la figura 6. De acuerdo a dicha figura, puede ver que los 4 primeros conmutadores permiten introducir en binario el dato A, que se representará en todo momento, en hexadecimal en el primer dígito del display. Con el dato B sucede lo mismo, sólo que con los últimos cuatro conmutadores y el segundo dígito del display. El resultado (formado por SUMH y SUML) se irá mostrando ciclo a ciclo de reloj en los dos últimos dígitos del display. Los botones señalados en la figura introducen un 1 lógico cuando se pulsan y están conectados a las señales 'reset', 'xs', y 'clk' del módulo *multiplicador*.

Para operar, lo primero a realizar es asignarle valor a los datos A y B usando los conmutadores.

El procedimiento para hacer que la unidad de control salga del estado de espera S0 consiste en pulsar en primer lugar el botón 'xs' y, **sin levantar el dedo** de él, pulsar una vez el botón 'clk'. Esto hace que el sistema avance hacia el estado S1 y empiece a multiplicar. En este momento debería ver un único LED encendido (ya puede levantar el dedo del botón 'xs') y a partir de ahora, cada vez que pulse el botón 'clk' el LED encendido cambiará de posición moviéndose hacia la derecha, mientras el sistema va operando ciclo a ciclo y cambiando de estado.

Si el sistema está operando correctamente, tras cada pulsación de 'clk' los dos últimos dígitos irán mostrando el resultado parcial de la multiplicación. Cuando el multiplicador active la señal de 'fin' se iluminarán todos los LEDs indicando que ha terminado la multiplicación. En este momento los dos últimos dígitos del display estarán mostrando (en hexadecimal) el resultado final. Por ejemplo, si multiplicamos A=1110 por B=0101, cuyo resultado final es 01000110, el display mostrará 'E546' al terminar, puesto que 'E' por '5' da como resultado '46' (todos ellos números hexadecimales).

Si sospecha que la unidad de control del multiplicador está en un estado incorrecto, puede pulsar el botón de 'reset' para hacer que la unidad de control vuelva, de forma asíncrona, al estado inicial S0.

3. Estudio experimental

Para poder realizar el estudio experimental debe estudiar previamente la descripción del multiplicador secuencial que aparece en la sección 2. Cuando lo haya hecho puede proceder con la parte experimental de la práctica, que de forma esquemática podría resumirse así:

1. Completar y verificar el multiplicador.
   1.1. Crear un proyecto en el entorno ISE.
   1.2. Añadir los 4 archivos Verilog del multiplicador.
   1.3. Completar el módulo Verilog de la unidad de control.
   1.4. Completar la descripción estructural en Verilog del multiplicador.
   1.5. Simular el multiplicador para verificar su correcto funcionamiento.
2. Realizar la implementación del sistema completo en un chip FPGA y verificar su funcionamiento.

A continuación se detallan los pasos a seguir para realizar cada uno de los apartados.

3.1. Completar y verificar el multiplicador

Para completar el diseño del multiplicador y verificar su funcionamiento siga los siguientes pasos:

1. Cree un nuevo proyecto vacío en el entorno ISE siguiendo los mismos pasos de la sesión de laboratorio anterior. Procure darle un nombre distintivo (por ejemplo su nombre propio) porque más adelante en la práctica deberá localizar esta carpeta dentro del disco duro.

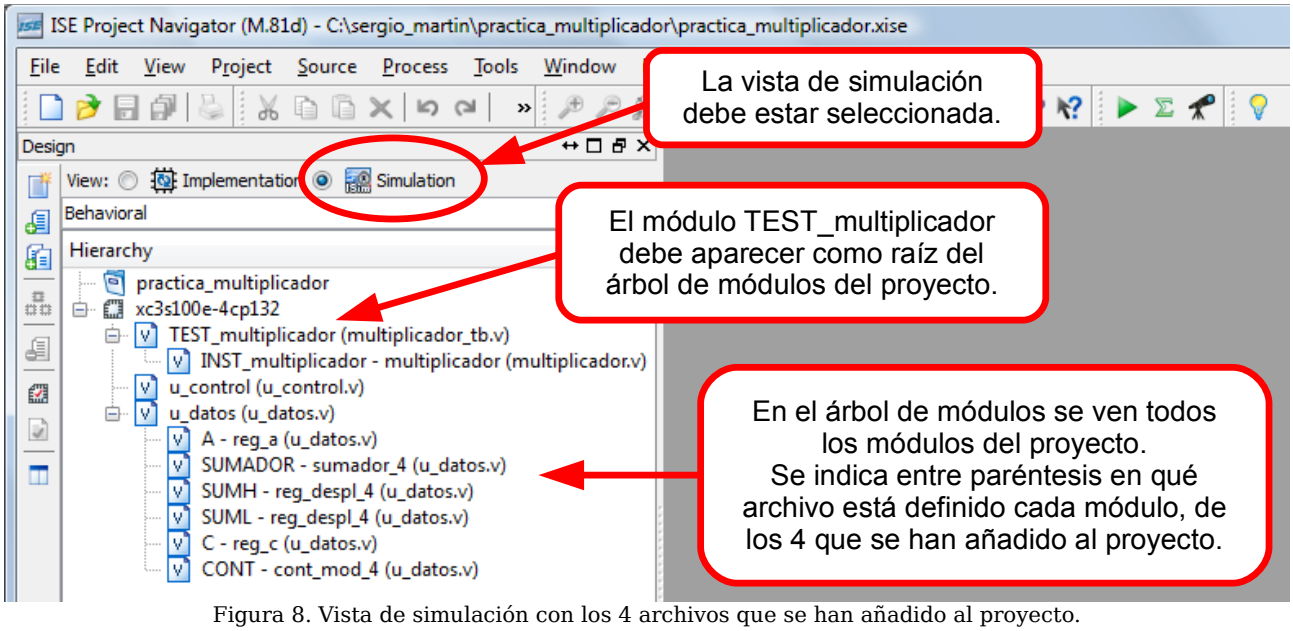


Figura 8. Vista de simulación con los 4 archivos que se han añadido al proyecto.

2. Seleccione la vista de **simulación** en el proyecto tal y como se muestra en la parte superior de la figura 8. Añada al proyecto los 4 ficheros que se muestran a continuación mediante la opción Add Source (o Add Copy of Source) dentro del menú Project:
   2.1. Fichero *u_datos.v*: Añádalo como *All*. No necesitará modificar su contenido.

2.2. Fichero *u_control.v*: Añádalo como *All.* Más adelante deberá editarlo para completarlo.

2.3. Fichero *multiplicador.v*: Añádalo como *All*. Más adelante deberá editarlo para completarlo.

2.4. Fichero *multiplicador_tb.v*: Añádalo como *Simulation*. No necesitará modificar su contenido.

Tras añadir los 4 ficheros, compruebe que el módulo llamado *TEST_multiplicador,* que es el testbench del módulo *multiplicador,* aparece en la raíz del árbol de módulos tal y como muestra la figura 8. Es normal que al añadir los 4 archivos en la consola se le muestre algún mensaje de error, pues hay archivos que están incompletos y contienen errores sintácticos que deberá corregir más adelante durante el desarrollo de la práctica.

```
module u_control(
  input  wire xs, reset, SUML0, cycont, clk,
  output reg  clinicio, clc, wc, wa, wsumh, wsuml, shrsum, upcont, fin
  );

  // Declaración de la lista de estados de la carta ASM.
  // Debe COMPLETARLA con los estados que faltan en la lista.
  parameter S0 = 3'b000,
            // COMPLETE CON LOS ESTADOS QUE FALTAN
            // COMPLETE CON LOS ESTADOS QUE FALTAN
            // COMPLETE CON LOS ESTADOS QUE FALTAN
            SF = 3'b100;

  // Declaración de las variables estado_actual y siguiente_estado.
  // Debe COMPLETARLA para que tengan el TAMAÑO CORRECTO.
  reg [ :0] estado_actual, siguiente_estado;


  // Proceso de cambio de estado. Utiliza las variables definidas previamente.
  always @(posedge clk,posedge reset)
      if(reset)
        estado_actual <= S0;
      else
        estado_actual <= siguiente_estado;


  // Proceso combinacional que calcula las salidas y el proximo estado.
  // Debe COMPLETARLO con la información obtenida a partir de la carta ASM.
  always @(*)
    begin
      // Ponemos a 0 todas las salidas, para que dentro del 'case' solo haya
      // que modificar el valor de las salidas que tengan que ponerse a 1.
      clinicio=0; clc=0; wc=0; wa=0; wsumh=0; wsuml=0; shrsum=0; upcont=0; fin=0;
      // Ponemos 'siguiente_estado' a S0 para que dentro del 'case' solo haya que
      // modificar 'siguiente_estado' cuando tenga que ponerse a un valor distinto de S0.
      siguiente_estado = S0;

      case (estado_actual)
        S0:
          if(xs)
            siguiente_estado = S1;


           // COMPLETAR, incluyendo el resto de estados de la carta ASM.
           // Obtenga la información necesaria a partir de la carta ASM.


       endcase
    end  // (del always)

endmodule
```

Fragmento de código 1. Fichero u_control.v con el diseño (incompleto) de la unidad de control

3. El siguiente paso es completar el fichero *u_control.v* (mostrado en el fragmento de código 1). Recuerde que puede abrir cualquier archivo para editarlo haciendo doble click sobre él en el árbol de módulos del proyecto. El objetivo es completar la descripción procedimental Verilog de la carta ASM que describe el funcionamiento de la unidad de control. Lea las indicaciones presentes en los comentarios del archivo *u_control.v* y complete todo lo que se pide en él fijándose atentamente en la carta ASM que está implementando y que puede ver en la figura 5, en la cual están claramente marcados los nombres de las cajas de estado y los nombres de las entradas y salidas de la unidad de control. De forma resumida, las tareas son estas tres:

   3.1. Completar la declaración '*parameter*' con **los estados que faltan** de la carta ASM.

   3.2. Declarar las variables '*estado_actual*' y '*siguiente_estado*' con su **tamaño correcto**.

   3.3. Completar **la descripción del comportamiento de cada estado** en la sentencia '*case*'.

   Siga adelante solo cuando al efectuar un *Behavioral Check Syntax* sobre este módulo, en la consola no veamos ningún error ni ningún aviso (warning).

```
module multiplicador(
  input  wire [3:0] a,      // Entrada del dato A
  input  wire [3:0] b,      // Entrada del dato B
  input  wire clk,          // Reloj del multiplicador
  input  wire xs,           // Señal de comienzo (para la U. de Control)
  input  wire reset,        // Reset asíncrono del sistema (para la U. de Control)
  output wire fin,          // Señal de fin de operacion
  output wire [7:0] mult    // Resultado multiplicación (solo es válido al activarse 'fin')
  );

  // Aquí debe definir 9 cables internos, cada uno de un solo bit,
  // para efectuar las conexiones entre la instancia del módulo
  // 'u_control' y la instancia del módulo 'u_datos'.
  // Puede darle los nombres que quiera siempre que no se estén usando ya.


  // Aquí debe crear una instancia del módulo 'u_datos'.
  // Use 'INST_u_datos' para el nombre de la instancia.
  // Al crear la instancia debe conectar sus entradas y salidas
  // a alguno de los cables internos que ha definido arriba y también
  // a alguno de los puertos de entrada y de salida del 'multiplicador'
  // Ayúdese de las figuras que aparecen en el manual de la práctica.
  // También puede abrir el archivo 'u_datos.v' para ver la
  // definición exacta de las entradas y salidas del módulo 'u_datos'.


  // Aquí debe crear una instancia del módulo 'u_control'.
  // Use 'INST_u_control' para el nombre de la instancia.
  // Al crear la instancia debe conectar sus entradas y salidas
  // a alguno de los cables internos que ha definido arriba y también
  // a alguno de los puertos de entrada y de salida del 'multiplicador'
  // Ayúdese de las figuras que aparecen en el manual de la práctica.
  // También puede abrir el archivo 'u_control.v' para ver la
  // definición exacta de las entradas y salidas del módulo 'u_control'.

endmodule
```

Fragmento de código 2. Fichero multiplicador.v con la interconexión de la unidad de datos y la unidad de control.

4. En este punto debe abrir el fichero *multiplicador.v* (mostrado en el fragmento de código 2) para completar la descripción estructural del módulo *multiplicador*. Lea las indicaciones presentes en los comentarios del archivo *multiplicador.v* y complete todo lo que se pide en él fijándose atentamente en la figura 1, que es donde se muestran qué conexiones debe hacer entre la unidad de datos y la unidad de control. De forma resumida, las acciones a realizar en este fichero son las siguientes:

   4.1. Completar la declaración de los cables internos que vaya a necesitar para la interconexión de las instancias.

   4.2. Completar las conexiones de la instancia de la unidad de control.

   4.3. Completar las conexiones de la instancia de la unidad de datos.

   Siga adelante solo cuando al efectuar un *Behavioral Check Syntax* sobre este módulo, en la consola no veamos ningún error ni ningún aviso (warning).

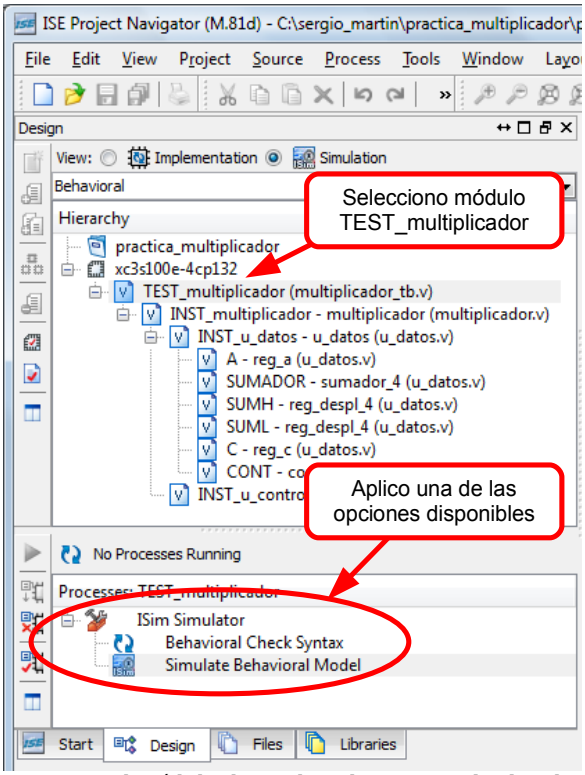


Figura 9. El módulo de testbench TEST_multiplicador

5. Por último se va a realizar la simulación del multiplicador y se va a verificar su correcto funcionamiento. Para ello siga los siguientes pasos (fíjese bien en la figura 9):

   5.1. Seleccione el módulo de testbench en el árbol de proyecto. Es el módulo superior en el árbol de proyecto con el nombre *TEST_multiplicador* (está dentro del fichero *multiplicador_tb.v).*

   5.2. Tras esta selección debe ejecutar el proceso *Behavioural Check Syntax* para comprobar que no hay hay errores en ningún módulo que forme parte del proyecto. En realidad, cuando estuvimos completando los archivos *u_control.v* y *multiplicador.v*, ya chequeamos la sintaxis de dichos módulos, por lo que no debería aparecer ahora ningún mensaje de error ni de aviso nuevo.

   5.3. Una vez corregidos los errores, realice la simulación del módulo *TEST_multiplicador* seleccionándolo en el árbol del módulos y usando la opción *Simulate Behavioural Model*. Si todo

va bien se le abrirá la aplicación ISim. Tenga en cuenta las siguientes cuestiones:

5.3.1. Si la simulación se para al llegar a 1 microsegundo use el icono del triángulo azul (*Run All*) para hacer que se ejecute la simulación hasta encontrar la orden $finish.

5.3.2. El fichero de testbench está programado para hacer 12 multiplicaciones distintas con el módulo *multiplicador*. Preste mucha atención a la consola, pues el testbench está programado para comprobar si el multiplicador funciona correctamente y le avisará si detecta errores.

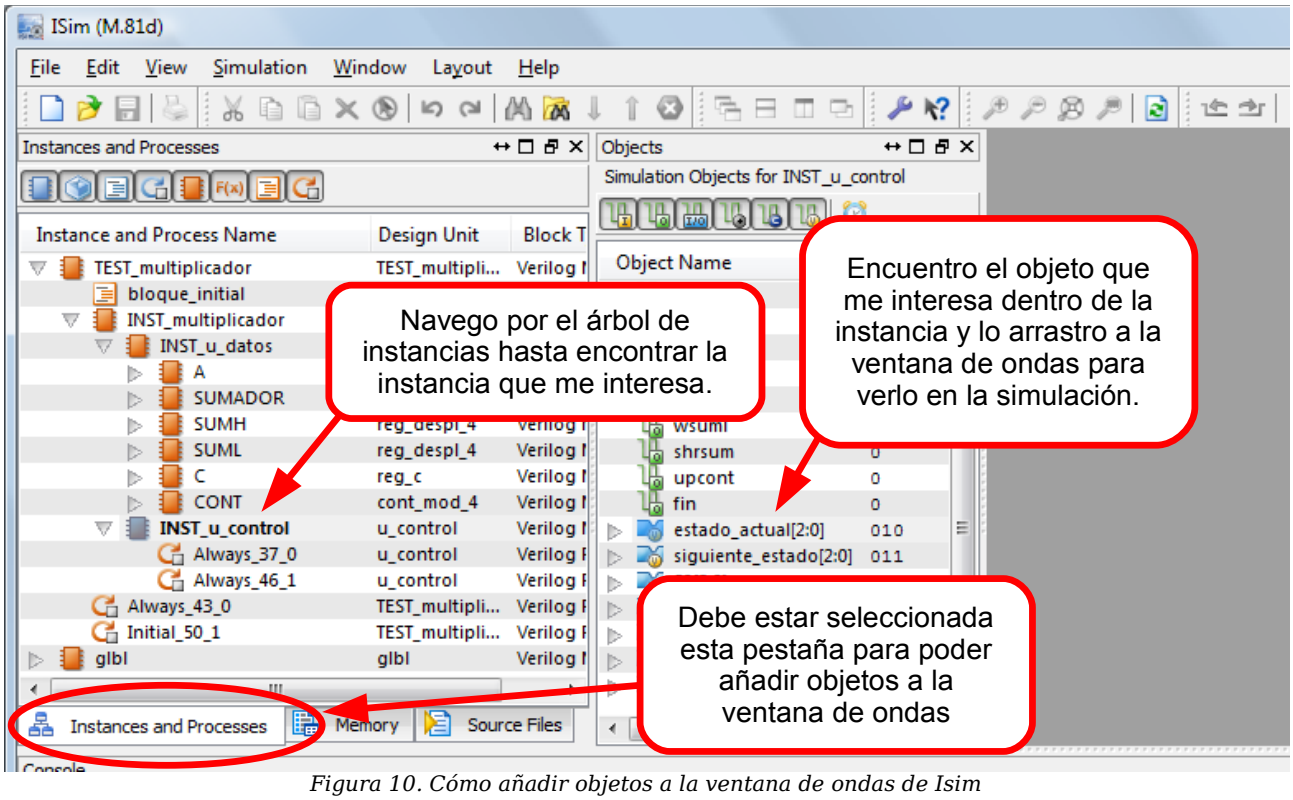


Figura 10. Cómo añadir objetos a la ventana de ondas de Isim

5.3.3. Si tras examinar la consola o las formas de onda considera que el multiplicador no está funcionando correctamente, debe encontrar el fallo y arreglarlo. Para ello es una buena idea hacer que la simulación nos muestre más información del funcionamiento interno de los módulos. Debería añadir a la ventana de formas de onda las señales internas de los módulos de la unidad de datos que considere más importantes (Los valores de las salidas del registro C, SUMH, SUML, o la variable estado_actual, por ejemplo). Fíjese en la figura 10 si no recuerda cómo se hacía. También es importante añadir el módulo multiplicador con todos sus elementos, para detectar cables sueltos, entradas no conectadas y fallos similares en la descripción estructural de este módulo. Obviamente esto también le debe ayudar a ver, paso a paso, si la unidad de datos y la unidad de control se están comportando exactamente como deberían. La primera multiplicación que prueba el testbench es la de los números 1110 y 1010, que es la misma que se explica paso a paso en la figura 5, así que puede ser buena idea fijarse en la figura e intentar ver que está yendo mal, sobre todo si el fallo está en la unidad de control. Esta misma multiplicación se muestra paso a paso en la última página del manual de la práctica, para que pueda comparar los resultados con los suyos.

No siga adelante hasta que los resultados de las 12 multiplicaciones sean correctos.

3.2. Implementar el multiplicador en la FPGA y verificar su funcionamiento

Finalmente se implementará el sistema digital realizado en un dispositivo programable tipo FPGA incluido en la placa de desarrollo *BASYS2*.

Tal y como se detalló en la sección 2.4 el multiplicador necesita conectarse a los conmutadores, botones y el display de la placa de desarrollo para poder interactuar con él. Los módulos que controlan estos componentes se encuentran ya diseñados y testados en el fichero *sistema_completo.v*. Además del sistema completo, es necesario utilizar otro fichero donde se indican las conexiones entre los pines del chip FPGA y las señales de entrada y salida del módulo *sistema_completo*. Este fichero se llama *basys2.ucf* y está completamente terminado, así que no es necesario tocar nada en él.

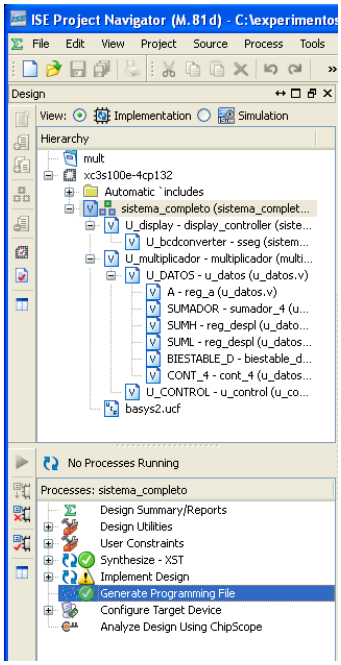


Figura 11. Implementación

Los pasos a seguir son los siguientes:

1. Cambiar el proyecto ISE de la vista de simulación a la vista de implementación (fíjese en la figura 11)
2. Añadir dos ficheros al proyecto:

   2.1. Fichero *sistema_completo.v*: Añadirlo como *Implementation*. Este fichero aparecerá ahora como raíz del árbol de módulos proyecto.

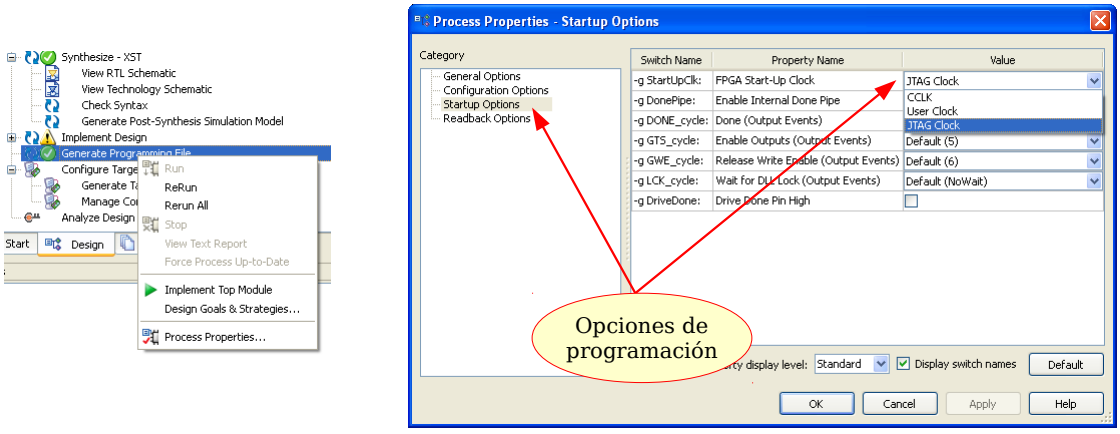
   2.2. Fichero *basys2.ucf*: Añadirlo como *Implementation*. Debe comprobar que también aparece en el árbol de proyecto.

3. La implementación se realiza seleccionando el módulo *sistema_completo* en el árbol de proyecto. 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 un menú flotante como el mostrado en la figura 12a. Seleccionando en ese menú 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 el 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 indicando la existencia de errores. En caso de existir errores debe corregirlos y repetir el proceso.



(a) *(b)*

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 *sistema_completo.bit* y hay que transferirlo por la conexión USB a la FPGA. Para ello siga los siguientes pasos:

   4.1. Compruebe que el modo de programación de la FPGA está establecido en modo *PC*. Para ello fíjese en la figura 7 y vea dónde está situado el conmutador *Modo de Programación*. Asegúrese de que la pieza de color azul está está colocada en modo *PC* y no en modo *ROM*.

   4.2. Conecte el puerto USB de la placa de desarrollo y mueva hacia arriba el conmutador de encendido (fíjese en la figura 7 si no sabe cuál es). Si ya estuviera encendida la placa, apáguela y enciéndala de nuevo.

   4.3. Inicie el programa Adept desde el menú de inicio (menú Digilent → Adept, icono Adept) 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. Solo vamos a programar la FPGA, por tanto se debe utilizar el botón Browse indicado en la figura y seleccionar el fichero *.bit* adecuado Para ello hay buscar la carpeta del proyecto ISE en el que se está trabajando y allí encontraremos el resultado de la síntesis en un fichero *.bit*, que en nuestro caso se llamará concretamente, *sistema_completo.bit*. Una vez seleccionado este fichero se activará el botón Program *y* bastará con pulsarlo para que la FPGA se programe.

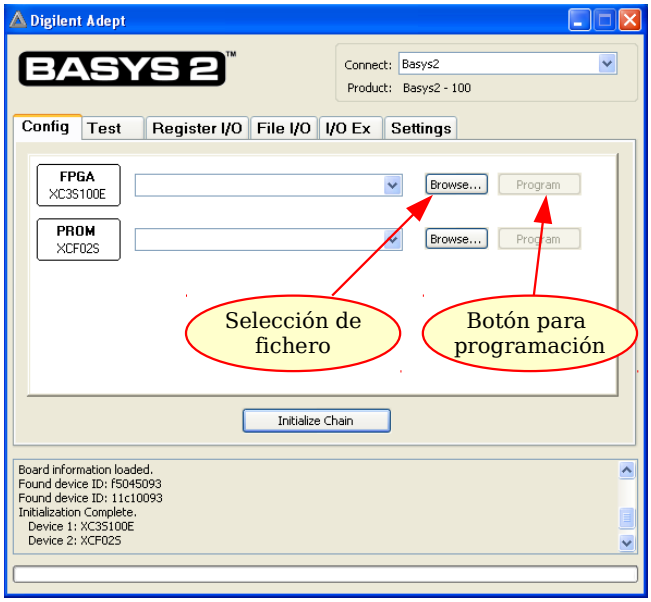


Figura 13. Programación de placas Digilent con Adept.

5. Debería haber leído previamente el apartado 2.4 de la página 8, donde se explica en detalle la forma en la que el módulo *multiplicador* se ha incluido dentro de otro módulo Verilog llamado *sistema_completo* y éste a su vez se interconectado a los elementos de la placa *BASYS2* para que podamos interactuar con él y hacer multiplicaciones "a mano". En la figura 7 (página 9), se nos muestra una fotografía de la placa de desarrollo *BASYS2* y en esta misma página se explica la forma en que debemos manipular los conmutadores y botones para hacer una multiplicación. Fijándose en esta figura y en las instrucciones de esa página, utilice los conmutadores y los botones para realizar la multiplicación de varios números y compruebe si el resultado es correcto, teniendo en cuenta que los números se representan en hexadecimal en el display. El procedimiento, de forma resumida, sería el siguiente:

   5.1. Colocar los números con los conmutadores.

   5.2. Pulsar el botón '*xs*' y, **sin** levantar el dedo, pulsar el botón 'clk' una vez para que comience la multiplicación. Ya puede levantar el dedo del botón 'xs'.

   5.3. Ahora nada más que resta generar más flancos de subida en el reloj, volviendo a pulsar el botón 'clk' para hacer avanzar el multiplicador ciclo a ciclo desde el estado S0 hasta el estado SF, momento en que se activará la señal de 'fin' y se encenderán todos los LEDS y se verá en el display el resultado de la multiplicación

3.3. Trabajo opcional

Intente conseguir que el multiplicador tarde menos ciclos en obtener el resultado final, estudiando las siguientes posibilidades.

1. Piense detenidamente si podemos eliminar la trasferencia C←0 que se hace en el estado S2 cuando SUML0 vale 0 si, a cambio, colocamos la transferencia C←0 en el estado S1 y también en el estado S3. Si cree que es posible hacerlo, modifique la carta ASM, la descripción Verilog de la misma y ejecute el testbench del multiplicador para comprobar que el multiplicador sigue funcionando bien.

2. Si ha hecho el apartado anterior, en el estado S2 de la carta ASM, cuando SUML0 es 0, no se hace ninguna acción. Simplemente se deja pasar un ciclo de reloj (sin razón alguna) para en el siguiente ciclo pasar a S3. Piense si sería posible, sin afectar al resultado final del algoritmo, modificar el comportamiento de S2, de forma que cuando SUML0 valga 0, se ejecutasen en S2 las acciones de S3, es decir, se ordenasen las transferencias SUMH←SHR(SUMH,C), SUML←SHR(SUML,SUMH[0]) y CONT←CONT+1 y luego se saltase al estado S2 o al estado SF dependiendo de si la entrada 'cycont' vale 0 o vale 1. Si cree que eso es posible, modifique la carta ASM, el código Verilog correspondiente y simule las multiplicaciones para ver si ahora tardan menos en realizarse. ¿Tardan todas las multiplicaciones el mismo tiempo? ¿Cuál es la que tarda más? ¿Cuál es la que tarda menos? ¿Cuánto tardaban antes de este cambio todas todas las multiplicaciones? ¿Cuántos ciclos nos ahorramos ahora en una multiplicación? ¿Tendría sentido multiplicar B x A en lugar de B x A en ciertos casos, para ahorrar tiempo en la multiplicación?

3. Si ha hecho todo lo anterior, piense si puede acelerar aún más la multiplicación eliminando el estado S1 y moviendo todas sus acciones a una caja de acción condicional que se ejecute solo cuando 'xs' valga 1. Si cree que es posible, cambie la carta ASM, impleméntela en Verilog y simule las multiplicaciones para ver si son correctas y tarda menos que antes. ¿Cuánto es el ahorro de tiempo, en ciclos, comparado con la anterior versión de las multiplicaciones?

4. Piense si sería posible modificar la unidad de datos para que los pasos 2 y 3 del algoritmo de multiplicación por sumas y desplazamientos se hicieran en un único ciclo de reloj. Si cree que es posible, haga esos cambios en la unidad de datos, cambie también en la carta ASM, modifique los módulos Verilog correspondientes y haga las simulaciones para ver que efectivamente puede hacerse.

Producto de 1110 × 0101

Ciclo de reloj	Unidad de Control												Unidad de Datos					micro-operaciones
	Entradas			Salidas								estado_actual	Contenido de los registros					
	xs	SUML0	cycont	clinicio	wa	wsuml	wsumh	wc	clc	shrsum	upcont	fin	A	CONT	C	SUMH	SUML	
0	0	x	x										S0	xxxx	x	x	xxxx	xxxx	
1	1	x	x										S0	xxxx	x	x	xxxx	xxxx	
2	0	x	x	1	1	1							S1	xxxx	x	x	xxxx	xxxx	SUMH←0, CONT←0, A←datoA, SUML←datoB
3	0	1	0				1	1					S2	1110	0	x	0000	0101	SUMH←A+SUMH, C←cout
4	0	1	0							1	1		S3	1110	0	0	1110	0101	SUMH←SHR(SUMH,C), SUML←SHR(SUML,SUMH[0]), CONT←CONT+1
5	0	0	0						1				S2	1110	1	0	0111	0010	C←0
6	0	0	0							1	1		S3	1110	1	0	0111	0010	SUMH←SHR(SUMH,C), SUML←SHR(SUML,SUMH[0]), CONT←CONT+1
7	0	1	0				1	1					S2	1110	2	0	0011	1001	SUMH←A+SUMH, C←cout
8	0	1	0							1	1		S3	1110	2	1	0001	1001	SUMH←SHR(SUMH,C), SUML←SHR(SUML,SUMH[0]), CONT←CONT+1
9	0	0	1						1				S2	1110	3	1	1000	1100	C←0
10	0	0	1							1	1		S3	1110	3	0	1000	1100	SUMH←SHR(SUMH,C), SUML←SHR(SUML,SUMH[0]), CONT←CONT+1
11	0	0	0									1	SF	1110	0	0	0100	0110	
12	0	0	0										S0	1110	0	0	0100	0110	

Nota: Las salidas de la U. de Control que están a valor 0 en un determinado ciclo de reloj se representan dejando vacía la casilla correspondiente.