6 Práctica 6. ASM con memoria

Esta práctica tiene dos objetivos. Por un lado, asentar los conocimientos sobre diseño de sistemas algorítmicos adquiridos en la práctica anterior y, por otro lado, aprender a manejar diseños que incluyan memorias.

6.1 Algoritmo de ordenación de burbuja

6.1.1 Especificaciones

En la primera parte de la práctica vamos a diseñar, simular y síntetizar un sistema algorítmico que dados 32 números de 4 bits en binario puro almacenados en una memoria SRAM los ordene de mayor a menor según el algoritmo de ordenamiento de burbuja.

Spec 1. El diseño se implementará como un sistema algorítmico que debe implementará el siguiente algoritmo:

```
i ← 0, Fin ← 0;
while i < n − 1 do
    j ← 0;
    while j < n − 2 do
        if M(j) ≤ M(j + 1) then
            M(j) ← M(j + 1), M(j + 1) ← M(j);
        end
        j ← j + 1;
    end
    i ← i + 1;
end
Fin ← 1;
```

Algorithm 2: Algoritmo de ordenamiento de burbuja.

En esta práctica $n = 32$. En el algoritmo $M(j)$ representa el contenido de la posición de memoria j-ésima. De esta forma $M(j) \leftarrow M(j+1)$ indica que el contenido de la palabra de memoria en la dirección de memoria $j+1$ se escribe en la palabra de memoria con dirección j. Y de forma similar, $M(j+1) \leftarrow M(j)$ indica que la palabra de memoria en la dirección j se escribe en la dirección de memoria $j+1$. En el algoritmo las dos operaciones de escritura se realizan de forma concurrente. Esto sólo es posible si disponemos de una memoria que permita realizar dos escrituras simultáneas. Caso de no disponer de esa memoria entonces será necesario serializar las escrituras: primero una y después otra, haciendo uso de una variable intermedia para almacenar el contenido de la palabra $M(j)$ y evitar que se pierda tras la primera escritura. Por último, tened en cuenta que en esta práctica la palabra de memoria tiene un ancho de 4 bits.

Spec 2. Su funcionamiento será síncrono y todos los registros serán activos por flanco de subida.

Spec 3. La señal de reloj será `clk`.

Spec 4. La señal de reset, `rst_n`, estará activa a nivel bajo.

Spec 5. El diseño tiene cuatro puertos de entrada: `ini` (1 bit), `debug_addr` (5 bits), `debug_din` (4 bits), `debug_we` (1 bit).

Spec 6. El diseño tiene dos puertos de salida: `fin` (1 bit) y `debug_dout` (4 bits).

Spec 7. La entidad `sort` viene definida por el siguiente código VHDL:

```
entity sort is
  port (clk        : in  std_logic;
        rst_n      : in  std_logic;
        ini        : in  std_logic;
        fin        : out std_logic;
        debug_addr : in  std_logic_vector(4 downto 0);
        debug_din  : in  std_logic_vector(3 downto 0);
        debug_we   : in  std_logic;
        debug_dout : out std_logic_vector(3 downto 0)
        );
end sort;
```

Spec 8. El sistema funciona en dos modos distintos: ordenar y depurar. En el modo ordenar ejecuta el Algoritmo 2. En el modo depurar se podrá ordenar operaciones de escritura y lectura sobre la memoria desde el exterior a través de los puertos de depuración.

Spec 9. Los puertos `debug_addr`, `debug_din`, `debug_we` y `debug_dout` son los puertos de depuración.

Spec 10. Entra en el modo ordenar cuando se active la señal `ini`. Una vez ordenados los 32 números se volverá al estado inicial y se activará la señal de salida `fin`. La señal `fin` permanecerá en alta hasta que que la señal `ini` tome el valor 1. En el modo ordenar la salida `debug_dout` estará a cero.

Spec 11. Mientras la señal `fin` sea '1', es decir mientras el sistema se encuentre en el estado inicial, el sistema se encuentra en el modo de depuración.

Spec 12. En el modo depurar se podrá ordenar escrituras sobre la memoria usando los puertos `debug_addr`, `debug_din` y `debug_we`. Así para realizar una escritura debe indicarse:

1. En el puerto `debug_addr` la dirección que se se desea escribir,
2. En el puerto `debug_din` el dato que se desea escribir.
3. En el puerto `debug_we` debe estar a 1 durante un ciclo de reloj.

La escritura será efectiva en el flanco de reloj en el que `debug_we`= 1.

Spec 13. En el modo depurar se podrá ordenar lecturas sobre la memoria usando los puertos `debug_addr` y `debug_we`. Así para realizar una lectura debe indicarse:

1. En el puerto `debug_addr` la dirección que se se desea leer.
2. En el puerto `debug_we` debe estar a 0 durante un ciclo de reloj.

La palabra de memoria leída aparecerá en el puerto `debug_dout` en el en el flanco de reloj en el que `debug_we`= 0.

Spec 14. En la ruta de datos solo puede usarse una memoria SRAM de 32 palabras de 4 bits, comparadores del ancho necesario, contadores ascendentes/descendentes, registros, multiplexores y sumadores.

6.1.2 Diseño

El circuito debe implementarse como un sistema algorítmico. Por lo tanto, a partir del Algoritmo 2 se debe definir el diagrama ASM y de él extraer la especificación de la ruta de datos y de la unidad de control. La Figura 6.1 presenta el diagrama ASM del Algoritmo 2. Este diagrama es una de las posibles implementaciones del algoritmo. En él hemos supuesto que usaremos una memoria SRAM de doble puerto, de forma que podrán hacerse las dos escrituras de forma simultanea.

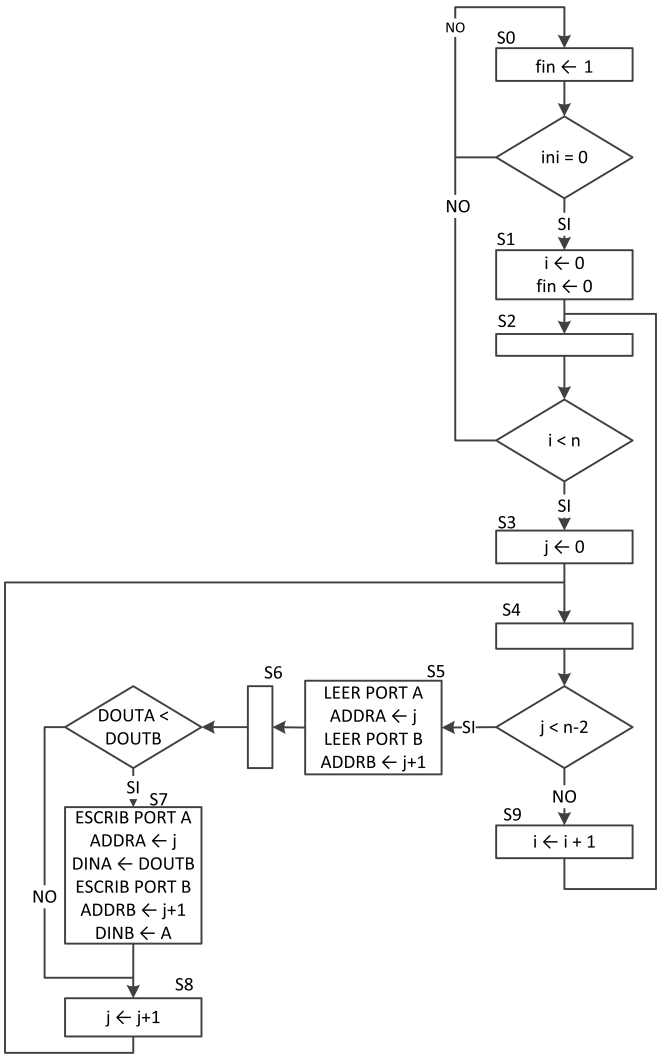

Figura 6.1: Diagrama ASM del Algoritmo 2.

A partir del diagrama ASM se deduce la ruta de datos que se presenta en la Figura 6.2. Como podéis apreciar, el camino de datos consta de: dos contadores ascendentes –cntri y cntrj– con sus señales de control –cntri_ld, cntri_cu, cntrj_ld y cntrj_cu–; tres comparadores –un comparador con 32, un comparador con 30 y un comparador de menor que– cada uno de los cuales genera una señal de estado –cmp_i, cmp_i y cmp_mem–; cuatro multiplexores que seleccionan las entradas y salidas del puerto A de la memoria dependiendo del modo de operación del sistema con su señal de control –debug_mode–; un sumador para calcular $j+1$; y una memoria SRAM de doble puerto. La descripción detallada de esta memoria se hará en la Sección 6.2 de esta práctica.

También a partir del diagrama ASM se extrae el diagrama de transición de estados de la uni-

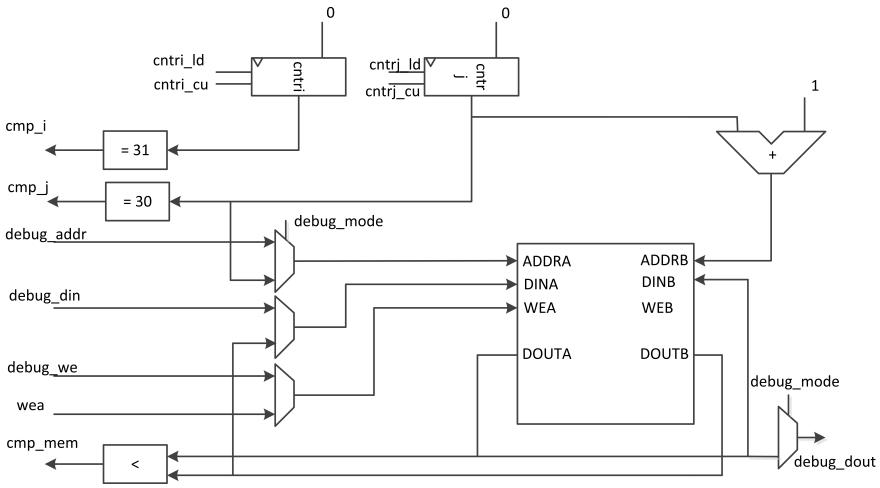

Figura 6.2: Camino de datos del ASM descrito en la Figura 6.1.

dad de control y la tabla de señales de control. La Tabla 6.1 presenta la plantilla de la tabla de señales de control. Se deja como ejercicio que el alumno deduzca el diagrama de transición de estados y que complete la Tabla 6.1.

Estado	debug_mode	web	wea	cntrj_cu	cntrj_ld	cntri_cu	cntri_ld	fin
S0	1	0	0	0	0	0	0	1
S1								
S2								
S3								
S4								
S5								
S6								
S7								
S8								
S9								

Tabla 6.1: Tabla de salidas de la unidad de control. A completar por el alumno.

6.1.3 Descripción VHDL

La estructura del diseño nos marca la forma en la que debemos describir el sistema algorítmico. Estará formado por dos entidades, cd y uc que contendrán el camino de datos y la unidad de control respectivamente. El camino de datos tendrá la siguiente definición de entidad.

```
entity cd is
  port (
    clk        : in  std_logic;                    -- clock
    rst_n      : in  std_logic;                    -- Async active low reset
    debug_addr : in  std_logic_vector(4 downto 0); -- Input address for debugging
    debug_we   : in  std_logic;                    -- Input write enable for debugging
    debug_din  : in  std_logic_vector(3 downto 0); -- Input data for debugging
    debug_dout : out std_logic_vector(3 downto 0); -- Debug output
    ctrl       : in  std_logic_vector(6 downto 0); -- Control
    status     : out std_logic_vector(2 downto 0)); -- Status
end entity cd;
```

La unidad de control tendrá la siguiente definición de entidad.

```
entity uc is
  port (
    clk    : in  std_logic;                      -- clock
    rst_n  : in  std_logic;                      -- Async active low reset
    ini    : in  std_logic;                      -- External control signal
    fin    : out std_logic;                      -- External control signal
    ctrl   : out std_logic_vector(6 downto 0);   -- Control vector
    status : in  std_logic_vector(2 downto 0));  -- Status vector
end entity uc;
```

Su definición seguirá las indicaciones que se presentaron en las prácticas 2 y 4 para el diseño de ambos módulos.

La asignación de las señales de control al vector de control seguirá el orden presentado en la Tabla 6.2.

bit	Señal de control
0	cntri_ld
1	cntri_cu
2	cntrj_ld
3	cntrj_cu
4	wea
5	web
6	debug_mode

Tabla 6.2: Asignación de señales de control a cada bit del vector de control, `ctrl`.

Y por último, la asignación de las señales de estado al vector de estado seguirá el orden presentado en la Tabla 6.3.

bit	Señal de estado
0	cmp_mem
1	cmp_j
2	cmp_i

Tabla 6.3: Asignación de señales de estado a cada bit del vector de estado, `status`.

6.2 Memoria

6.2.1 Generación

En esta práctica utilizaremos una memoria de doble puerto que será capaz de realizar dos operaciones independientes simultáneamente: una por cada uno de los puertos. De esta forma no será necesario serializar las dos operaciones de escritura que se indicaban en el algoritmo.

Antes de utilizar esta memoria debemos generarla y para ello utilizaremos la herramienta *IPCore Generator* incluida en el paquete ISE. Para arrancar la herramienta vamos a la pestaña *Tools* y seleccionamos la opción *Core Generator*. Esta acción arranca la herramienta y lo primero que debemos hacer es crear un nuevo proyecto dentro de la herramienta *Xilinx Core Generator*. Para ello ejecutad la acción "File → New project" y creamos un nuevo proyecto con la siguiente configuración:

- Project Name: sort_mem

- Family: Spartan3
- Device: XC3S1000
- Package: FT256
- Speed: -5

Una vez configurado, debajo del panel "*View by Function*" seleccionamos la carpeta *Memories & Storage Elements*, dentro de ella la carpeta *RAMs & ROMs* y arrancamos el wizard para *Block Memory Generator*, que es el tipo de memoria que vamos a utilizar. Este wizard nos guiará en el proceso para definir el tipo de memoria que deseamos y una vez completada la definición generará todos los archivos necesarios para la simulación, síntesis y P&R de la memoria. Veamos a continuación los pasos a seguir para realizar la definición:

1. En la primera página debemos asignar un nombre a la memoria –en nuestro caso vamos a asignar el nombre sort_mem– y la opción *Native*.

2. En la segunda página escogemos el tipo de memoria. En esta práctica escogeremos *True Dual Port RAM* y las opciones *Common Clock* y *Minimum Area*. La FPGA de la familiar Spartan-3 pueden definirse distintos tipos de memoria. Nosotros escogemos *True Dual Port RAM* porque ser el único tipo con dos puertos que permite dos operaciones independientes de forma concurrente. Es decir, con este tipo de memoria podemos leer y escribir de forma independiente sobre cada uno de los dos puertos de la memoria –portA y portB–.

3. En la página 3 seleccionamos los anchos de escritura –*Write Width*– a 4 bits y la profundidad de la memoria –*Write Depth*– a 32 para ambos puertos. También seleccionamos las opciones "*Operating Mode Write First*" y "*Enable Always*" para ambos puertos. Estas opciones establecen que:

   - Durante la operación de escritura las memorias de la Spartan-3 siempre vuelcan en el puerto de salida, y en el mismo flanco de reloj con el que se efectúa la operación de escritura, la palabra de memoria que está siendo direccionada. Como el valor que hay en esa posición es distinto antes y después de la escritura con la opción "*Operating Mode Write First*" indicamos que primero se efectúe la escritura y que el valor que aparece en el puerto de salida sea el valor que se acaba de escribir en memoria. Tened en cuenta que el hecho de que la memoria ponga en su puerto de salida el valor que se ha escrito no quiere decir que tengáis que hacer nada con él. Si no es necesario este valor en los cómputos que realiza vuestro diseño, basta con no usarlo para nada.
   - La memoria puede o bien estar habilitada siempre o estarlo sólo cuando se indique mediante una señal de habilitación. Con la opción "*Enable Always*" estamos indicando que la memoria estará habilitada siempre y, por lo tanto, no habrá señal de habilitación. Al estar siempre habilitada, si no estamos haciendo una operación de escritura entonces estamos haciendo una operación de lectura. Es decir, cuando la señal de write enable esté a '1' se estará realizando una escritura y cuando esté a '0' se estará realizando una lectura. No existe la posibilidad, con esta configuración, de no hacer ni lectura ni escritura.

4. En la página 4 no escogemos ninguna de las opciones para ninguno de los dos puertos.

5. En la página 5 no escogemos nada.

6. En la página 6 seleccionamos la opción *All* para detectar colisiones de escritura-lectura y escritura-escritura y finalmente pulsamos la opción generar. La opción escogida en esta página determina que los modelos de simulación de la memoria serán capaces de detectar

colisiones en los accesos a la memoria desde los dos puertos. Ocurrirá una colisión cuando desde los dos puertos se accede a la misma posición de memoria y una de las operaciones sea de escritura. Si esto ocurre, el modelo detecta la colisión y lo indicará mediante un mensaje en la consola del simulador.

Toda la información de la memoria se genera en un directorio que recibe el nombre ipcore_dir. Para añadir la memoria al proyecto, debéis incorporar el archivo sort_mem.xco, que se encuentra en ese directorio, al proyecto siguiendo el mismo procedimiento que usáis cuando incorporáis un archivo vhd. Una vez lo hayáis incorporado podréis instanciar la memoria y usarla tanto en simulación como en síntesis.

6.2.2 Interfaz y funcionamiento

La memoria que hayáis obtenido deberá tener la siguiente definición de puertos:

```
component sort_mem
  port (
    clka  : in  std_logic;
    wea   : in  std_logic_vector(0 downto 0);
    addra : in  std_logic_vector(4 downto 0);
    dina  : in  std_logic_vector(3 downto 0);
    douta : out std_logic_vector(3 downto 0);
    clkb  : in  std_logic;
    web   : in  std_logic_vector(0 downto 0);
    addrb : in  std_logic_vector(4 downto 0);
    dinb  : in  std_logic_vector(3 downto 0);
    doutb : out std_logic_vector(3 downto 0)
    );
end component;
```

Para ver que ésta es la definición del componente que habéis obtenido podéis seleccionar la memoria en la ventana *design* la vista *Implementation* y en la ventana de las herramientas escoger la opción "*View HDL Instantiation Template*".

La definición de los puertos es la siguiente:

clka : Puerto de reloj del puerto A.

wea : Write enable del puerto A. Cuando esta señal está a 1 en un flanco de reloj se realiza la escritura del dato que hay en dina en la dirección indicada por addra.

addra : Bus de direcciones del puerto A.

dina : Bus de datos de entrada del puerto A.

douta : Bus de datos de salida del puerto A.

clkb : Puerto de reloj del puerto B.

web : Write enable del puerto B. Cuando esta señal está a 1 en un flanco de reloj se realiza la escritura del dato que hay en dinb en la dirección indicada por addrb.

addrb : Bus de direcciones del puerto B.

dinb : Bus de datos de entrada del puerto B.

doutb : Bus de datos de salida del puerto B.

Téngase en cuenta que los bloques de memoria de Xilinx son síncronos tanto para lectura como para escritura. Es decir, a efectos de temporización la memoria se comporta como un enorme registro. Luego:

- Para la lectura: la dirección del dato buscado tiene que estar en addra o addrb en el ciclo de reloj anterior al flanco en el que se ordena la operación y el dato leído aparecerá en el puerto douta o doutb en el ciclo de reloj que sigue al flanco donde se ha ordenado la operación.
- Para la escritura: el dato no estará efectivamente escrito en la memoria hasta que no se produzca un flanco de subida en el reloj.

6.3 Cuestiones y resultados experimentales

La documentación a presentar en la memoria del apartado es:

1. Diagrama de transición de estados.
2. Tabla de salidas de la unidad de control.
3. Indicar el número de FF, LUT y puertas básicas que ha inferido XST.
4. Describir el camino crítico encontrado por la herramienta: señales fuente y destino e indicar sobre el diagrama de bloques de la Figura 4.1 por donde transcurre dicho camino. ¿Cuál es la frecuencia máxima de trabajo?

Para que el código de la práctica sea considerado correcto se deben cumplir los siguientes criterios:

1. La simulación del sistema debe ser correcta.
2. El código debe reflejar los detalles del diseño presentado en la memoria.
3. El código debe seguir todas las reglas incluidas en el documento “Reglas de estilo”.