5 Práctica 4. Divisor secuencial con restauración de números enteros sin signo

El objetivo de esta práctica es asentar los conocimientos sobre el diseño de sistemas algorítmicos usando como demostrador un divisor secuencial de números enteros sin signo.

5.1 Divisor secuencial de tamaño fijo

5.1.1 Especificaciones

En la primera parte de la práctica vamos a diseñar, simular y síntetizar un divisor secuencial con restauración de números enteros sin signo de 8 bits.

Spec 1. El divisor se implementará como un sistema algorítmico.

Spec 2. El divisor debe obedecer el siguiente algoritmo:

```
dndo_r ← dndo;
dsor_r ← alinear(dsor) & ( others => '0');
coc_r ← 0;
cntr ← valor_desplazamiento_dsor;
cntr_d1 ← 7;
repeat
    dndo_r ← dndo_r − dsor_r;
    cntr_d1 ← cntr;
    cntr ← cntr−1;
    if dndo_r[msb] = 0 then
        coc_r ← coc_r[msb-1:0] & '1';
    else
        coc_r ← coc_r[msb-1:0] & '0';
        dndo_r ← dndo_r + dsor_r;
    end
    dsor_r ← '0' & dsor_r[msb:1];
until cntr_d1 = 0;
```

Algorithm 1: Algoritmo de división secuencial con restauración.

Spec 3. Su funcionamiento será síncrono y todos los registros serán activos por flanco de subida.

Spec 4. La señal de reloj, `clk`, deberá tener una frecuencia superior a 25 MHz.

Spec 5. La señal de reset, `rst_n`, estará activa a nivel bajo.

Spec 6. El divisor tiene dos puertos de entrada, `dndo` y `dsor`, cada uno de 8 bits de ancho: `dndo` será el dividendo y `dsor` el divisor.

Spec 7. El divisor tiene dos puertos de salida, `coc` y `rem`, de 8 bits de ancho: `coc` será el cociente y `res` el resto.

Spec 8. El sistema tiene un puerto de entrada, `ini`, de 1 bit de ancho que indica comienzo de la división. Cuando `ini`= 1 los valores a dividir deberán estar en los puertos de entrada, `dndo` y `dsor`. La señal `ini` debe permanecer en alta la menos durante 1 ciclo de reloj.

Spec 9. El sistema tiene un puerto de salida, `ready`, de 1 bit de ancho que indica final de división. Cuando `ini`= 1 el resultado de la división deberá estar en los puertos de salida, `coc` y `res`. La señal `ready` permanecerá en alta hasta que que la señal `ini` tome el valor 1.

Spec 10. La entidad `divisor` viene definida por el siguiente código VHDL:

```
entity divisor is
  port (clk   : in std_logic;
        rst_n : in std_logic;
        ini   : in std_logic;
        ready : out std_logic;
        dndo  : in  std_logic_vector(7 downto 0);
        dsor  : in  std_logic_vector(7 downto 0);
        coc   : out std_logic_vector(7 downto 0);
        res   : out std_logic_vector(7 downto 0)
        );
end divisor;
```

5.1.2 Diseño

La Figura 5.1 ilustra la división de dos números enteros de 8 bits — 00101101 ($186|_{10}$) y 00000110 ($6|_{10}$) — siguiendo el Algoritmo 1. El divisor está alineado antes de proceder a realizar la división. Esto es, se han eleminado los ceros en los bits más significativos de forma que siempre el bit más significativo sea 1. Además, el número de bits que se ha desplazado el divisor debe almacenarse en un registro para determinar el número de veces que es necesario hacer la resta. El cociente de esta división es 11111 ($31|_{10}$) y el resto 0000 ($0|_{10}$). Al realizar la división recuérdese que la resta de dos números, $x - y$, puede realizarse sumando el primer operando con el complemento a 2 del segundo, $x + C2(y)$ siendo en este caso $C2(0110) = 1010$.

1	0	1	1	1	0	1	0	1	1	0				
0	0	0						0	1	1	1	1	1	
1	0	1	1											
1	0	1	0											
0	1	0	1	1										
	1	0	1	0										
	0	1	0	1	0									
		1	0	1	0									
		0	1	0	0	1								
			1	0	1	0								
			0	0	1	1	0							
				1	0	1	0							
				0	0	0	0							

Cuadro 5.1: Ejemplo de división de los números 00101101 ($186|_{10}$) y 00000110 ($6|_{10}$) según el Algoritmo 1. El divisor ha sido alineado para que el bit más significativo sea 1.

El método tradicional consiste en ir obteniendo los restos parciales y los bits del cociente recorriendo de izquierda a derecha los bits del dividendo:

- Si el resto parcial es mayor que el divisor, añadir un 1 al cociente; el nuevo resto parcial será la resta del resto parcial y del divisor
- Si el resto parcial es menor que el divisor, añadir un 0 al cociente y ampliar el resto parcial con un bit más del dividendo.

Este método es viable siempre que se cumplan las siguientes dos restricciones:

1. No admite división por cero. Es decir, el divisor debe ser distinto de cero.
2. El divisor debe ser menor que el dividendo.

La Figura 5.2 presenta cómo cambia el valor de las cuatro variables –`dndo_r`, `dsor_r`, `coc_r` y `cntr`– con cada paso del control del Algoritmo 1 sobre los datos del primer ejemplo de la práctica. Al finalizar el algoritmo el registro `dndo_r` contendrá el resto de la división.

dndo_r	dsor_r	coc_r	cntr	cntr_d1
10111010	11000000	00000000	5	7
10111010	01100000	00000000	4	5
01011010	00110000	00000001	3	4
00101010	00011000	00000011	2	3
00010010	00001100	00000111	1	2
00000110	00000110	00001111	0	1
00000000	00000011	00011111	7	0

Cuadro 5.2: Valores tomados por las variables del algoritmo para los datos del ejemplo. El contenido del registro `dsor_r` es el divisor de 8 bits alineado para que el bit más significativo sea 1.

El circuito debe implementarse como un sistema algorítmico. Por lo tanto, a partir del Algoritmo 1 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 5.1 presenta el diagrama ASM del Algoritmo 1. Este diagrama es una de las implementaciones del algoritmo. Otra posibilidad es eliminar la asignación sobre la variable `cntr_d1` y comprobar la condición de salto directamente sobre `cntr` en el mismo estado que se decrementa su valor.

A partir del diagrama ASM se deduce la ruta de datos que se presenta en la Figura 5.2.

También a partir del diagrama ASM se extrae el diagrama de transición de estados de la unidad de control y la tabla de señales de control. La Tabla 5.3 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 5.3.

Est.	add_sub	cntr_d1_ld	cntr_ld	cntr_cu	dsor_ld	dsor_sh	dndo_ld	coc_ld	coc_sh	mux	ready
S0	0	0	0	0	0	0	0	0	0	0	1
S1											
S2											
S3											
S4											
S5											
S6											

Cuadro 5.3: Plantilla de la tabla de señales de control. A completar por el alumno.

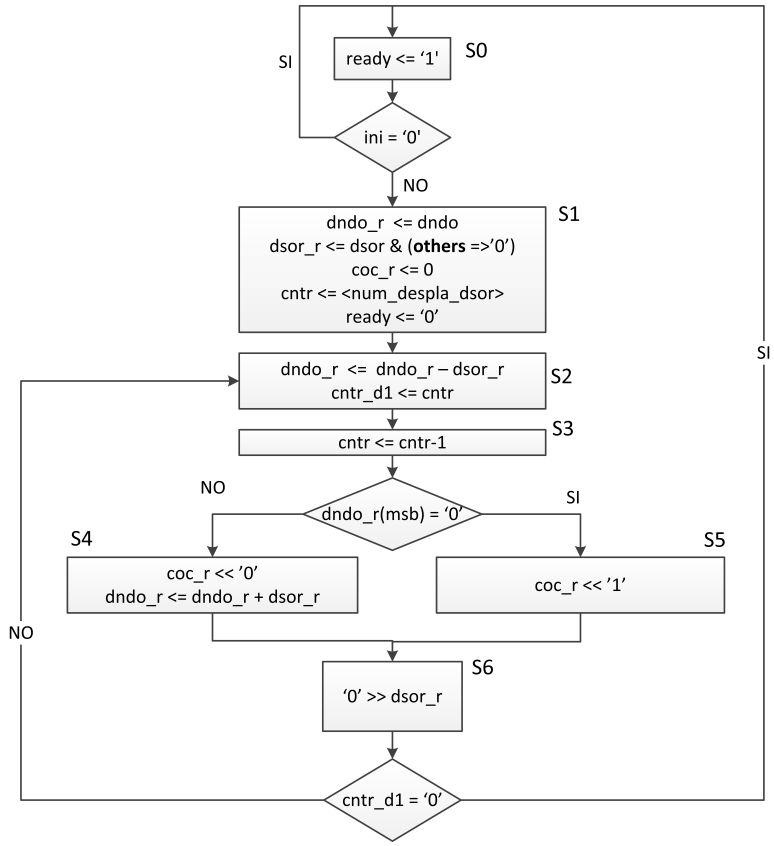

Figura 5.1: Diagrama ASM del Algoritmo 1.

5.1.3 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;
        rst_n  : in std_logic;
        dndo   : in  std_logic_vector(7 downto 0);
        dsor   : in  std_logic_vector(7 downto 0);
        coc    : out std_logic_vector(7 downto 0);
        res    : out std_logic_vector(7 downto 0);
        ctrl   : in  std_logic_vector(9 downto 0)
        status: out std_logic_vector(1 downto 0));
end cd;
```

Este módulo puede definirse de dos formas: estructural o RTL.

1. Descripción estructural. Con esta aproximación se definen entidades por separado para cada uno de los componentes necesarios: sumador, registros y comparadores, para a continuación instanciarlos y conectarlos dentro del camino de datos. Si se sigue esta aproximación entonces la arquitectura del camino de datos recibirá el nombre struc.

2. Definición RTL. Definimos cada uno de los componentes del camino de datos mediante un proceso, sin necesidad de instanciar componentes. Si se sigue esta aproximación entonces la arquitectura del camino de datos recibirá el nombre rtl.

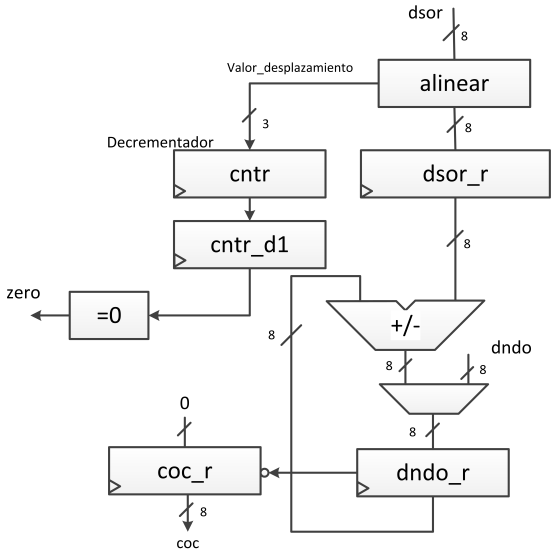

Figura 5.2: Camino de datos. La señal `zero` es una señal de estado.

Cualquiera de las dos formas es aceptable siempre y cuando la descripción respete la estructura del diseño presentada en la Figura 5.2.

La unidad de control tendrá la siguiente definición de entidad.

```
entity uc is
  port (clk    : in  std_logic;
        rst_n  : in  std_logic;
        ini    : in  std_logic;
        ready  : out std_logic;
        ctrl   : out std_logic_vector(9 downto 0)
        status: in  std_logic_vector(1 downto 0));
end uc;
```

Su definición seguirá las indicaciones que se presentaron en la práctica 2 para el diseño de máquinas de estados finitos.

La forma más cómoda de definir las señales de control es utilizar constantes para indicar los valores que deben tomar cada uno de los bits del vector de control según la operación a realizar. Con esta aproximación, primero se define el significado de cada uno de los bits del vector de control. Una posible asignación seria:

bit	Señal de control
0	mux
1	coc_sh
2	coc_ld
3	dndo_ld
4	dsor_sh
5	dsor_ld
6	cntr_cu
7	cntr_ld
8	cntr_d1_ld
9	add_sub

Cuadro 5.4: Asignación de señales de control a cada bit del vector de control, `ctrl`.

A continuación se define tantas constantes como operaciones elementales puede realizar la unidad de control. Por ejemplo:

```
constant c_coc_sh  : std_logic_vector(9 downto 0) := "0000000010";
constant c_coc_ld  : std_logic_vector(9 downto 0) := "0000000100";
constant c_dndo_ld : std_logic_vector(9 downto 0) := "0000001000";
...
```

Finalmente para cada estado se definen los valores de las señales de control asignando la OR lógica de todas las operaciones que tienen que realizarse en un determinado paso de control. Por ejemplo, en el caso de tener que cargar el registo `coc_r` y `dndo_r` en un mismo estado de la unidad de control, la asignación de la señal de control sería la siguiente:

```
...
ctrl <= c_coc_ld or c_dndo_ld;
...
```

El camino de datos genera dos señales de estado: `zero` y `dndo_r[msb]`. Estas señales deben transmitirse a la unidad de control mediante la señal `status`. La asignación de estas señales al vector de estado será:

bit	Señal de estado
0	zero
1	dndo_r[msb]

Cuadro 5.5: Asignación de señales de estado a cada bit del vector de estado, `status`.

5.2 Cuestiones y resultados experimentales

La documentación a presentar en la memoria del apartado es:

1. Diagrama ASM con la optimización propuesta en la sección .
2. Diagrama de transición de estados.
3. Tabla de salidas de la unidad de control.
4. Indicar el número de FF, LUT y puertas básicas que ha inferido XST.
5. 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?
6. Realizar una simulación post-place&route y determinar si existen glitches a las salidas del circuito. Si existen indicar valores de las entradas con las que aparecen dichos glitches.

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”.

5.3 Diseños avanzados

En esta sección se presentan dos diseños avanzados que el alumno deberá completar una vez haya realizado el diseño de la Sección 5.2.

5.3.1 Divisor secuencial de tamaño fijo optimizado

En la Sección 5.2 se solicita que el alumno presente el diagrama ASM del diseño optimizado en el que se elimina la asignación sobre la variable `cntr_d1` y se comprueba la condición de salto directamente sobre `cntr` en el mismo estado que se decrementa su valor. En esta primera parte avanzada se solicita que implemente el diseño en VHDL.

5.3.2 Divisor secuencial paralelo

En este diseño vamos a implementar un divisor con un intervalo de inicialización de un ciclo: cada ciclo de reloj debe ser capaz de tomar dos nuevos operandos, y por lo tanto cada ciclo de reloj debe ser capaz de generar un nuevo conjunto de resultados –cociente y resto–.

Para ello, mediante la sentencia **generate** se deben generar ocho instancias del divisor de la Sección 5.2, de forma que cada uno de ellos reciba un nuevo par de operandos cada ciclo de reloj. Para que la asignación de datos a cada divisor se haga en orden se hará uso de un contador módulo 8 que seleccionará, siempre en el mismo orden, el divisor que ejecutará el algoritmo con los datos que en ese ciclo están en los puertos de entrada del nuevo divisor. El mismo contador puede usarse para ir tomando los resultados calculados por cada divisor y mostrarlos en el puerto de salida.

Las especificaciones del sistema son:

Spec 1. Su funcionamiento será síncrono y todos los registros serán activos por flanco de subida.

Spec 2. La señal de reloj, `clk`, deberá tener una frecuencia superior a 25 MHz.

Spec 3. La señal de reset, `rst_n`, estará activa a nivel bajo.

Spec 4. El divisor tiene dos puertos de entrada, `dndo` y `dsor`, cada uno de 8 bits de ancho: `dndo` será el dividendo y `dsor` el divisor.

Spec 5. El divisor tiene dos puertos de salida, `coc` y `rem`, de 8 bits de ancho: `coc` será el cociente y `res` el resto.

Spec 6. El sistema tiene un puerto de entrada, `ini`, de 1 bit de ancho que indica comienzo de la división. Cuando `ini`= 1 los valores a dividir deberán estar en los puertos de entrada, `dndo` y `dsor`. La señal `ini` debe permanecer en alta la menos durante 1 ciclo de reloj.

Spec 7. El sistema tiene un puerto de salida, `ready`, de 1 bit de ancho que indica final de división. Cuando `ini`= 1 el resultado de la división deberá estar en los puertos de salida, `coc` y `rem`. La señal `ready` permanecerá en alta hasta que que la señal `ini` tome el valor 1.

Spec 8. La entidad `divisor` viene definida por el siguiente código VHDL:

```
entity divisor_parallel is
  port (clk   : in std_logic;
        rst_n : in std_logic;
        ini   : in std_logic;
        ready : out std_logic;
        dndo  : in  std_logic_vector(7 downto 0);
        dsor  : in  std_logic_vector(7 downto 0);
        coc   : out std_logic_vector(7 downto 0);
        res   : out std_logic_vector(7 downto 0)
        );
end divisor_parallel;
```

Spec 9. El sistema debe ser capaz de tomar un dato cada ciclo de reloj.

5.3.3 Diseño

El divisor paralelo contendrá 8 divisores instanciados mediante la sentencia **generate**. Todos ellos recibirán las entradas dndo y dsor de divisor_parallel. Para seleccionar qué divisor tomará un dato en cada momento se creará un contador módulo 8 cuya salida será decodificada mediante un decodificador. Cada una de las salidas del decodificador tiene que ir hasta una puerta AND en la que la otra entrada será la señal `ini`. De esta forma un divisor sólo comenzará a realizar el algoritmo de la división con los datos en los puertos de entrada si se cumple que: (1) es el divisor seleccionado en ese ciclo de reloj y (2) si la señal `ini` toma el valor 1 en ese ciclo de reloj –indicando que los valores en los puertos dndo y dsor son correctos–.

De igual forma las salidas de cada divisor deben ir hasta tres multiplexores 8 a 1–un multiplexor para las salidas coc, otro para las salidas res y un último para las salidas ready–. Las señales de selección de estos multiplexores será generada por el contador presentado en el párrafo anterior.