

Práctica de Estructura y Tecnología de Computadores III

Nombre: Javier de Silóniz Sandino
Dirección: ***** (Cádiz).
DNI: 7577***6*
Número de teléfono: *****
e-mail: *****@****.com

Práctica obligatoria 1: Diseño algorítmico de un monoestable

El diseño realizado es el de un monoestable no redispensible, cuyo pulso de salida tiene un ancho de 10 ns. El código se haya dividido en dos archivos: la entidad (*monoestable.vhd*) y su arquitectura (*monoestable_arq1.vhd*). Así mismo, las pruebas para el simulador se hayan en los archivos *prueba.vhd* y *prueba_arq1.vhd*.

El diseño realizado ha sido de tipo algorítmico. A continuación expongo los listados de código de la entidad y la arquitectura realizados:

monoestable.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY monoestable IS
5	GENERIC (tw : time := 10 ns);
6	
7	PORT (disparo: IN std_logic;
8	Q: OUT std_logic := '1';
9	nQ: OUT std_logic := '0');
10	
11	END monoestable;

En las líneas 1 y 2, se carga la librería IEEE para poder usar los tipos de variable `std_logic` y `std_logic_vector`.

En la descripción de la entidad (líneas 4 a 11), se establece el ancho del pulso de salida del monoestable. Después se establece el puerto de entrada (`disparo`, de tipo `std_logic`), y los de salida (`Q` y `nQ` de tipo `std_logic`, con valores iniciales opuestos).

monoestable_arq1.vhd	
1	ARCHITECTURE arq1 OF monoestable IS
2	BEGIN
3	
4	PROCESS
5	BEGIN
6	
7	Q <= '1';
8	nQ <= '0';
9	
10	WAIT UNTIL (disparo'EVENT AND disparo='1');
11	
12	Q <= '0';
13	nQ <= '1';
14	

monoestable_arq1.vhd	
15	WAIT FOR tw;
16	
17	END PROCESS;
18	
19	END arq1;

El modelo algorítmico para el monoestable es muy sencillo. Se asigna a las salidas Q y nQ los valores iniciales '1' y '0' (líneas 7 y 8), y se detiene en cuánto hay un pulso de subida en la entrada "disparo" (línea 10). Entonces se asigna a las salidas los valores inversos (líneas 12 y 13), esperando a que pase el tiempo *tw* expresado en la definición de la entidad (10 ns.) para volver al estado inicial de las salidas.

A continuación paso a exponer el código fuente de la prueba para el simulador, y su cronograma correspondiente.

prueba.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY prueba_generica IS
5	END prueba_generica;

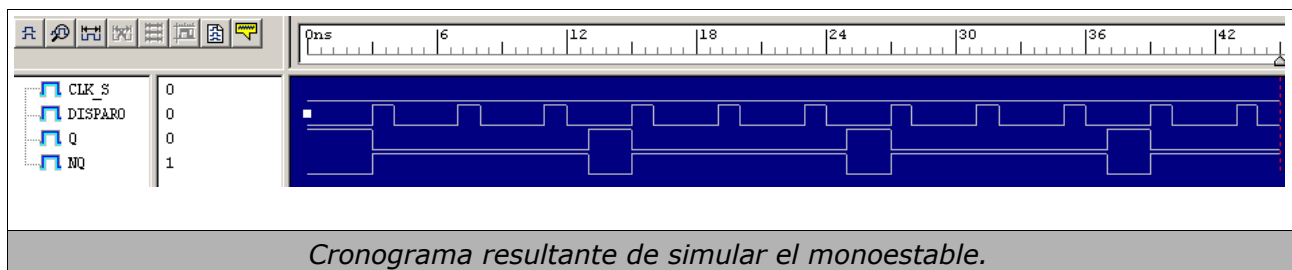
prueba_arq1.vhd	
1	ARCHITECTURE arq1 OF prueba_generica IS
2	
3	COMPONENT monoestable
4	GENERIC (tw: time:= 10 ns);
5	PORT (disparo: IN std_logic; q, nq: OUT std_logic);
6	END COMPONENT;
7	
8	CONSTANT duracion_ciclo: TIME := 1 us;
9	SIGNAL CLK_s: BIT := '0';
10	
11	SIGNAL disparo: std_logic;
12	SIGNAL q, nq: std_logic;
13	
14	BEGIN
15	
16	monoestable1: monoestable PORT MAP (disparo, q, nq);
17	reloj: PROCESS
18	BEGIN
19	WAIT FOR duracion_ciclo;

prueba_arq1.vhd	
20	CLK_s <= '1';
21	WAIT FOR duracion_ciclo;
22	CLK_s <= '0';
23	END PROCESS reloj;
24	
25	pruebas: PROCESS
26	BEGIN
27	disparo <= '0';
28	WAIT FOR 3 ns;
29	disparo <= '1';
30	WAIT FOR 1 ns;
31	
32	END PROCESS pruebas;
33	END arq1;

El componente a probar se declara usando la sentencia COMPONENT entre las líneas 3 y 6. Las señales usadas en la prueba se declaran en las líneas 11 y 12 (*disparo*, *q* y *nq* del tipo *std_logic*). El componente es entonces instanciado en la línea 16, asignándole a sus entradas y salidas las señales de prueba.

Tras el proceso reloj ya descrito en el enunciado de la práctica, se declara otro de pruebas, en el que se asignan los valores de la prueba a la señal de entrada (*disparo*).

El cronograma resultante de esta prueba es el siguiente:



Práctica obligatoria 2: Diseño estructural de un registro de desplazamiento con entrada serie y salida paralelo de 8 bits.

Se expone a continuación el diseño realizado para el registro de desplazamiento de 8 bits **SRG_SP8**.

Al tratarse de un diseño estructural, se han elaborado los componentes fundamentales del diseño (biestables RS), para después ser unidos aparte en la arquitectura final. Pasamos a describir primero el diseño realizado para los biestables RS y luego el conjunto final, así como la prueba realizada para el simulador.

biestableRS.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY biestableRS IS
5	PORT(R, S, clk, clr: IN std_logic;
6	Q, nQ: OUT std_logic);
7	END biestableRS;

Se describen en esta entidad los puertos de entrada (*R, S, reloj y clear*) y de salida (*Q y nQ*), siendo todos del tipo std_logic.

biestableRS_arq1.vhd	
1	ARCHITECTURE arq1 OF biestableRS IS
2	BEGIN
3	
4	PROCESS
5	VARIABLE estado: std_logic := '0';
6	VARIABLE n_estado: std_logic := '1';
7	
8	BEGIN
9	
10	IF (clr'EVENT AND clr='0') THEN
11	
12	estado := '0';
13	n_estado := '1';
14	
15	ELSIF (clk'EVENT AND clk = '0') THEN
16	IF (S='0' AND R='0') THEN
17	estado:=estado;
18	n_estado:=n_estado;
19	ELSIF (S='0' AND R = '1') THEN
20	estado:='0';

biestableRS_arq1.vhd	
21	n_estado:='1';
22	ELSIF (S='1' AND R='0') THEN
23	estado:='1';
24	n_estado:='0';
25	ELSE
26	estado:='-';
27	n_estado:='-';
28	END IF;
29	END IF;
30	
31	Q <= estado;
32	nQ <= n_estado;
33	
34	WAIT ON clk,clr;
35	
36	END PROCESS;
37	END arq1;

Usando de base el código para el biestable RS que hay en el enunciado de la práctica, he introducido ciertas modificaciones para ajustarlo al diseño final del registro de desplazamiento.

He codificado la condición para manipular la señal *CLR* de borrado asíncrono, introduciendo una nueva condición en la línea 10; de tal manera que el biestable sólo actúe cuando la condición de borrado no se cumpla. Así mismo, he anidado los *IF* que definen la conducta del biestable (líneas 16 a 28) bajo otro *IF* que define la condición en la que el biestable se dispara (línea 15). El proceso se renueva en la nueva sentencia *WAIT ON* de la línea 34, teniendo en cuenta las señales de reloj y borrado.

Una vez descrito el comportamiento del biestable RS, se puede pasar a crear el registro desplazador interconectando 8 instancias del componente entre sí. Ésto se realiza en los archivos *srg_sp8.vhd* y *srg_sp8_arq1.vhd* que paso a describir a continuación:

srg_sp8.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY srg_sp8 IS
5	PORT(a,b,clk,clr: IN std_logic;
6	q: OUT std_logic_vector (0 TO 7));
7	END srg_sp8;

srg_sp8_arq1.vhd	
1	ARCHITECTURE arq1 OF srg_sp8 IS
2	
3	COMPONENT biestableRS IS
4	PORT(R, S, clk, clr: IN std_logic;
5	Q, nQ: OUT std_logic);
6	END COMPONENT;
7	
8	SIGNAL entrada_serie, entrada_serie_negada: std_logic := '0';
9	SIGNAL salidas_q, salidas_nq : std_logic_vector (0 TO 7);
10	SIGNAL reloj_negado: std_logic;
11	
12	BEGIN
13	
14	entrada_serie <= A NAND B;
15	entrada_serie_negada <= NOT entrada_serie;
16	reloj_negado <= NOT clk;
17	
18	despl_8: FOR i IN 0 TO 7 GENERATE
19	
20	cero : IF i= 0 GENERATE
21	primero : biestableRS PORT MAP (entrada_serie,
22	entrada_serie_negada, reloj_negado, clr, salidas_q(0), salidas_nq(0));
23	END GENERATE cero;
24	
25	mayor_que_cero : IF i>0 GENERATE
26	otros : biestableRS PORT MAP (salidas_nq(i-1),
27	salidas_q(i-1), reloj_negado, clr, salidas_q(i), salidas_nq(i));
28	END GENERATE mayor_que_cero;
29	
30	END GENERATE displ_8;
31	
32	q <= salidas_q;
	END arq1;

Para crear y conectar entre sí los 8 biestables RS necesarios, tenemos que recurrir a las sentencias *GENERATE*. Primero declaramos el biestable usando la sentencia *COMPONENT* entre las líneas 3 a 6. Tras ello creamos las señales necesarias para la interconexión de los biestables (incluyendo las señales para las negaciones necesarias que se hayan en el diseño del desplazador).

Llegados a este punto, se crea el componente *despl_8* (línea 18), que a su vez hemos subdividido en dos sentencias *GENERATE*. La primera describe el primer biestable (cuyas entradas deben ser la entrada y la entrada negada; y cuyas salidas deben asignarse a los bits menos significativos de las arrays *salidas_q* y *salidas_nq*).

La segunda sentencia *GENERATE* se compone de los otros 7 biestables, cuyas entradas corresponden con las salidas de los anteriores, y cuyas salidas se asignan a los bits correspondientes a sus posiciones en las arrays *salidas_q* y *salidas_nq*.

De este modo se consigue que los biestables se conecten entre sí en cascada, teniendo al final como salida paralelo la unión de las distintas salidas parciales de cada uno de ellos.

A continuación paso a exponer el código realizado para la prueba en el simulador y su correspondiente cronograma:

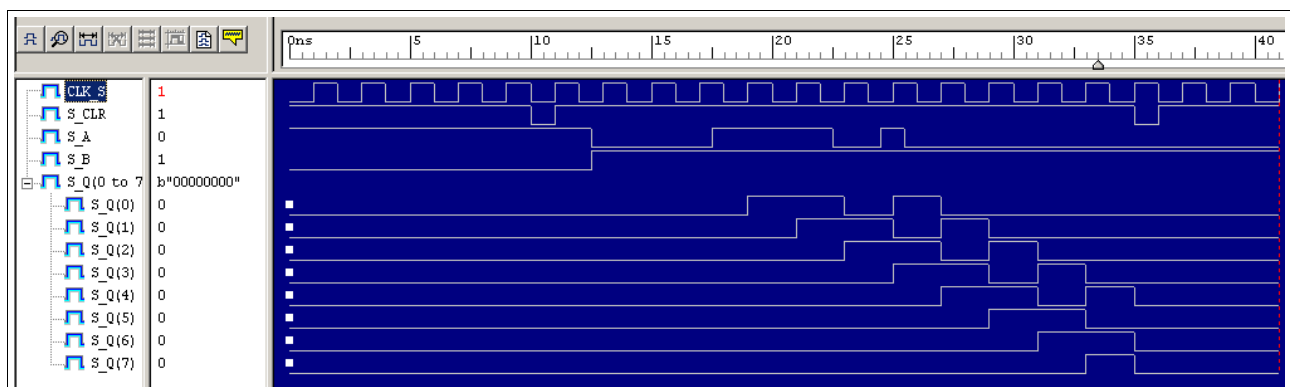
testSRG_SP8.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY prueba_generica IS
5	END prueba_generica;

testSRG_SP8_arq1.vhd	
1	ARCHITECTURE arq1 OF prueba_generica IS
2	
3	COMPONENT registro_desp
4	PORT(a,b,clk,clr: IN std_logic;
5	q: OUT std_logic_vector (0 TO 7));
6	END COMPONENT;
7	
8	CONSTANT duracion_ciclo: TIME := 1 ns;
9	SIGNAL CLK_s: std_logic := '0';
10	
11	FOR registro: registro_desp USE ENTITY WORK.SRG_SP8(arq1);
12	
13	SIGNAL s_CLR : std_logic := '1';
14	SIGNAL s_A : std_logic := '1';
15	SIGNAL s_B : std_logic := '0';
16	SIGNAL s_Q : std_logic_vector (0 TO 7);
17	
18	BEGIN
19	
20	registro: registro_desp PORT MAP (s_A, s_B, CLK_s, s_CLR, s_Q);
21	
22	reloj: PROCESS
23	BEGIN
24	WAIT FOR duracion_ciclo;
25	CLK_s <= '1';
26	WAIT FOR duracion_ciclo;

testSRG_SP8_arq1.vhd	
27	CLK_s <= '0';
28	END PROCESS reloj;
29	
30	pruebas: PROCESS
31	BEGIN
32	s_CLR <= '1' AFTER 0 ns,
33	'0' AFTER 10 ns,
34	'1' AFTER 11 ns,
35	'0' AFTER 35 ns,
36	'1' AFTER 36 ns;
37	
38	s_A <= '0' AFTER 12.5 ns,
39	'1' AFTER 17.5 ns,
40	'0' AFTER 22.5 ns,
41	'1' AFTER 24.5 ns,
42	'0' AFTER 25.5 ns;
43	
44	s_B <= '1' AFTER 12.5 ns;
45	WAIT;
46	
47	END PROCESS pruebas;
48	END arq1;

Como en la anterior prueba el proceso es similar. Declaro el componente entre las líneas 3 y 6, defino las señales apropiadas para el reloj de prueba y asigno a la instanciación que haré luego la arquitectura arq1 de la entidad SRG_SP8 en el espacio de trabajo actual. Luego defino las señales necesarias para la prueba (señales de entrada y salida del componente; líneas de 13 a 16).

Tras instanciar el componente en la línea 20, defino los dos procesos de la prueba: reloj y prueba. En el último defino las entradas para s_CLR, s_A y s_B en el tiempo tal como se indica en el enunciado de la práctica. El resultado es el siguiente cronograma:



Cronograma resultante de simular el registro desplazador de 8 bits.

Práctica obligatoria 3: Diseño estructural de un registro de desplazamiento de n bits con entrada y salida paralelo.

Se expone a continuación el diseño realizado para crear un registro de desplazamiento con entrada y salida paralelo, extensible a un número n de bits. En el ejemplo que vamos a probar al final, n será equivalente a 4 bits.

Tal como se define en el enunciado de la práctica, este registro de desplazamiento se compone de n biestables D. Éstos a su vez se componen cada uno de un biestable RS. He usado el diseño del biestable RS que creé en la práctica anterior para realizar el diseño del biestable D. Describo a continuación el desarrollo de este componente antes de centrarme en el del registro (obvio el listado de código del biestable RS, puesto que ya ha sido descrito con anterioridad).

biestableD.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY biestableD IS
5	PORT (D, clk: IN std_logic;
6	Q, nQ: OUT std_logic);
7	END biestableD;

biestableD_arq1.vhd	
1	ARCHITECTURE arq1 OF biestableD IS
2	
3	COMPONENT biestableRS IS
4	PORT(R, S, clk, clr: IN std_logic;
5	Q, nQ: OUT std_logic);
6	END COMPONENT;
7	
8	SIGNAL clk_negado, D_negado: std_logic := '1';
9	SIGNAL clr_anulado : std_logic := '1';
10	
11	FOR ALL : biestableRS USE ENTITY WORK.biestableRS(arq1);
12	
13	BEGIN
14	D_negado <= NOT D;
15	clk_negado <= NOT clk;
16	
17	biestable:biestableRS PORT MAP (D_negado, D, clk_negado,clr_anulado, q, nQ);
18	
19	END arq1;

Para realizar el biestable D, primero he declarado el biestable RS (lineas 3 a 6). Tras

ello, he creado señales intermedias necesarias para el diseño (señales para la negación de D y de la señal de reloj; y la señal *clr_anulado* que anula la entrada *clr* del biestable RS para que nunca se produzca un borrado asíncrono).

Tan sólo queda realizar las negaciones de las señales D y reloj (líneas 14 y 15), e instanciar el componente final asignando las señales convenientes.

Una vez descrito el componente base del registro desplazador, paso a exponer el código del componente final.

srg_ppn.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY srg_ppn IS
5	GENERIC (magnitud: natural := 4);
6	
7	PORT (D: IN std_logic_vector (0 TO magnitud-1);
8	clk: IN std_logic;
9	Q: OUT std_logic_vector (0 TO magnitud-1));
10	
11	END srg_ppn;

En la declaración de la entidad hace falta destacar la línea 5, donde describo con una sentencia *GENERIC* el número de bits que quiero para el registro desplazador. Uso ese valor genérico en la declaración de los puertos *D* y *Q* (*magnitud-1* en ambos casos). Modificando el valor de *magnitud* en la línea 5, podría obtener diferentes registros de diferentes tamaños.

srg_ppn_arq1.vhd	
1	ARCHITECTURE arq1 OF srg_ppn IS
2	
3	COMPONENT biestableD IS
4	PORT (D, clk: IN std_logic;
5	Q: OUT std_logic);
6	END COMPONENT;
7	
8	BEGIN
9	
10	gen: FOR i IN 0 TO magnitud-1 GENERATE
11	biestables : biestableD PORT MAP (D(i), clk, Q(i));
12	END GENERATE gen;
13	
14	END arq1;

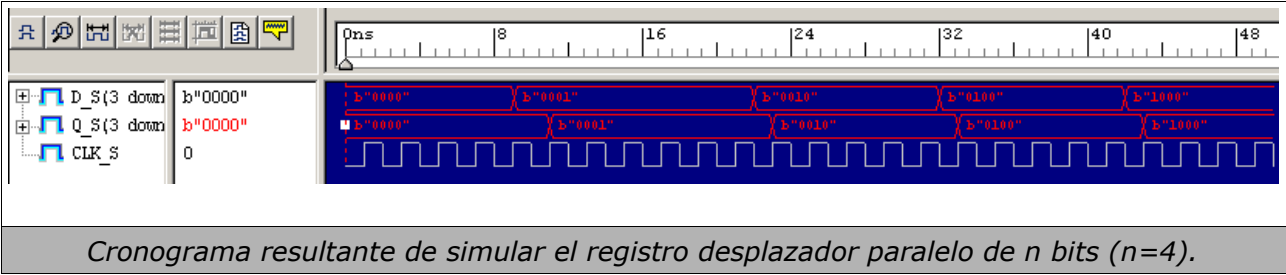
La arquitectura del registro de desplazamiento consiste en generar varios biestables D (tantos como se necesiten, basándonos en el valor del parámetro *magnitud* antes descrito). Cada uno de los biestables D recibe una de las entradas paralelo del registro, y se conecta a cada una de las salidas paralelo del mismo. A su vez están conectados a la misma fuente de reloj *CLK*. Con lo cual, basta una sentencia *FOR..GENERATE* para crear los biestables y conectarlos, creando así el registro desplazador final.

Paso a describir ahora la prueba realizada para el simulador y el cronograma correspondiente:

prueba_srg_ppn.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY prueba_generica IS
5	END prueba_generica;

prueba_srg_ppn_arq1.vhd	
1	ARCHITECTURE arq OF prueba_srg_ppn IS
2	
3	COMPONENT srg_ppn IS
4	GENERIC (magnitud: natural := 4);
5	PORT (D: IN std_logic_vector (magnitud-1 DOWNT0 0);
6	clk: IN std_logic;
7	Q: OUT std_logic_vector (magnitud-1 DOWNT0 0));
8	END COMPONENT;
9	
10	FOR ALL : srg_ppn USE ENTITY WORK.srg_ppn(arq1);
11	
12	CONSTANT duracion_ciclo: TIME := 1 ns;
13	CONSTANT magnitud: natural := 4;
14	
15	SIGNAL D_s, Q_s: std_logic_vector (magnitud-1 DOWNT0 0) := "0000";
16	SIGNAL clk_s : std_logic := '0';
17	
18	BEGIN
19	
20	registro : srg_ppn PORT MAP (D_s, clk_s, Q_s);
21	
22	reloj: PROCESS
23	BEGIN
24	WAIT FOR duracion_ciclo;
25	CLK_s <= '1';
26	WAIT FOR duracion_ciclo;

prueba_srg_ppn_arq1.vhd	
27	CLK_s <= '0';
28	END PROCESS reloj;
29	
30	pruebas: PROCESS
31	BEGIN
32	D_s <= "0000" AFTER 0 ns,
33	"0001" AFTER 9.1 ns,
34	"0010" AFTER 22 ns,
35	"0100" AFTER 32 ns,
36	"1000" AFTER 42 ns;
37	
38	WAIT;
39	
40	END PROCESS;
41	END arq;



Práctica obligatoria 4: Diseño estructural de un circuito convertidor serie-paralelo.

Se expone a continuación el diseño de un circuito simplificado de un convertidor serie-paralelo. En él, he debido usar componentes de las anteriores prácticas, así como desarrollar otros nuevos. En el caso de los primeros, en algunos casos he debido de realizar algunos cambios para adaptarlos a lo que el enunciado de la práctica pide.

Enumero a continuación los diferentes componentes del circuito final, y una breve descripción de su procedencia: flip-flop de control (nuevo desarrollo), generador de reloj (nuevo desarrollo), contador de división por 8 (nuevo desarrollo), puerta lógica AND (nuevo desarrollo), registro de entrada de datos (basado en el registro desplazador de 8 bits `srg_sp_8`), registro de salida de datos (basado en el registro desplazador de entradas y salidas paralelo, escalado a 8 bits), y un monoestable (el mismo de la práctica 1).

Paso a describir a continuación los nuevos componentes que he realizado para el desarrollo del conversor serie-paralelo.

El flip-flop de control se realiza con un biestable JK. No difiere demasiado de un biestable RS, únicamente cambia que en el caso de $J=1$ y $K=1$ la entrada se invierte. Basándome en el prototipo del enunciado de la práctica, he creado el biestable JK siguiente:

biestableJK.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY biestableJK IS
5	PORT (J, K, clk, clr: IN std_logic;
6	Q, nQ: OUT std_logic);
7	END biestableJK;

biestableJK_arq1.vhd	
1	ARCHITECTURE arq1 OF biestableJK IS
2	BEGIN
3	
4	PROCESS
5	VARIABLE estado: std_logic := '0';
6	
7	BEGIN
8	
9	IF (clr'EVENT AND clr='0') THEN
10	estado := '0';
11	
12	ELSIF (clk'EVENT AND clk = '1') THEN

biestableJK_arq1.vhd	
13	IF (J='0' AND K='0') THEN
14	estado:=estado;
15	ELSIF (J='0' AND K='1') THEN
16	estado:='0';
17	ELSIF (J='1' AND K='0') THEN
18	estado:='1';
19	ELSIF (J='1' AND K='1') THEN
20	estado:= NOT estado;
21	END IF;
22	END IF;
23	
24	Q <= estado;
25	nQ <= NOT estado;
26	
27	WAIT ON clk,clr;
28	
29	END PROCESS;

Los cambios realizados con respecto al prototipo del enunciado son: contemplación de la señal asíncrona de puesta a cero *clr* (líneas 9 y 10), activación por flanco negativo (línea 12) y el uso del tipo de datos *std_logic*.

El generador de reloj lo he realizado usando un modelo estructural, con una puerta AND y un biestable D. Paso a describirlos:

puerta_and.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY puerta_and IS
5	PORT (a, b: IN std_logic;
6	c: OUT std_logic);
7	END puerta_and;

puerta_and_arq.vhd	
1	ARCHITECTURE arq OF puerta_and IS
2	
3	BEGIN
4	PROCESS(a,b)
5	BEGIN
6	c <= a AND b;
7	END PROCESS;
8	END arq;

Simplemente se trata de un componente con dos entradas *a* y *b* de tipo *std_logic* , y una salida *c* también de tipo *std_logic*. El componente hace una operación AND lógica entre las entradas y la devuelve al puerto de salida.

En el caso del biestable D, he decidido no usar el de la práctica anterior. En el caso de esta práctica, el retardo que hace el simulador a la hora de manejar los biestables RS daba siempre una situación '1'- '1', dando lugar a una indeterminación en la señal de salida del generador. Por ello, y puesto que la función del biestable D en este caso es retrasar un ciclo de reloj la entrada, creo que un planteamiento algorítmico es mucho más lógico y funcional. A continuación expongo el código fuente de dicho desarrollo:

biestableD.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY biestableD IS
5	PORT (D, clk: IN std_logic;
6	Q: OUT std_logic);
7	END biestableD;

biestableD_arql.vhd	
1	ARCHITECTURE arql OF biestableD IS
2	BEGIN
3	
4	PROCESS
5	BEGIN
6	WAIT UNTIL (clk'EVENT);
7	Q <= D;
8	END PROCESS;
9	
10	END arql;

Este código simplemente espera a que la señal CLK cambie para enviar la entrada D a la salida Q.

Con estos componentes desarrollados ya estamos en disposición de crear el generador de reloj:

generador_reloj.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY generador_reloj IS
5	PORT (Q, clk: IN std_logic;
6	s_CLK: OUT std_logic);
7	END generador_reloj;

generador_reloj_arq.vhd	
1	ARCHITECTURE arq1 OF generador_reloj IS
2	
3	COMPONENT puerta_and IS
4	PORT (a, b: IN std_logic;
5	c: OUT std_logic);
6	END COMPONENT;
7	
8	COMPONENT biestableD IS
9	PORT (D, clk: IN std_logic;
10	Q: OUT std_logic);
11	END COMPONENT;
12	
13	SIGNAL salida_and1, salida_biestable : std_logic;
14	FOR ALL : puerta_and USE ENTITY WORK.puerta_and(arq);
15	FOR ALL : biestableD USE ENTITY WORK.biestableD(arq1);
16	
17	BEGIN
18	
19	biestable : biestableD PORT MAP (Q, clk, salida_biestable);
20	and1 : puerta_and PORT MAP (salida_biestable, clk, salida_and1);
21	
22	s_CLK <= salida_and1;
23	
24	END arq1;

El generador de reloj no es más que un biestable D que recibe como entrada la salida Q del flip-flop de control, y sincronizado al reloj de entrada envía su salida a una puerta AND a la que también está conectada la señal de reloj de entrada. La salida de la puerta AND se envía como salida del generador de reloj. Como en todo diseño estructural, los pasos realizados han sido: declaración de componentes (*puerta_and* entre las líneas 3 y 6, y *biestableD* entre las líneas 8 y 11), declaración de señales intermedias (*salida_and1* y *salida_biestable* para las salidas de cada componente, de tipo *std_logic*; línea 13), configuración de componentes (líneas 14 y 15), instanciaciones de los componentes (líneas 19 y 20), y devolución de valor de salida (línea 22).

Para el contador de división por 8 he realizado un simple desarrollo algorítmico. Describo el código fuente a continuación:

ctr_div8.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY ctr_div8 IS
5	PORT (clr, clk : IN std_logic;
6	TC : OUT std_logic);
7	END ctr_div8;

ctr_div8_arq.vhd	
1	ARCHITECTURE arq1 OF ctr_div8 IS
2	
3	BEGIN
4	PROCESS
5	
6	VARIABLE contador : NATURAL := 0;
7	
8	BEGIN
9	IF (clr'EVENT AND clr='0') THEN
10	contador:= 0;
11	TC <= '0';
12	ELSIF (clk'EVENT AND clk='1') THEN
13	IF(contador<7) THEN
14	contador := contador +1;
15	TC <= '0';
16	ELSE
17	contador := 0;
18	
19	TC <= '1';
20	WAIT ON clk;
21	TC <= '0';
22	END IF;
23	END IF;
24	
25	WAIT ON clk, clr;
26	
27	END PROCESS;
28	END arq1;

El funcionamiento es el siguiente: cuando la señal de puesta a 0 asíncrona no está activa (*clr*='1'), cada vez que hay un flanco de subida en el reloj de entrada, una variable *contador* se incrementa en una unidad. En el caso de que el valor de dicha

variable supere las 8 unidades, se envía un pulso de salida *TC* con el ancho de un ciclo de reloj (líneas 19 a 21).

En el caso del registro de salida de datos, he adaptado varios aspectos del registro de desplazamiento paralelo-paralelo de 4 bits escalable de la práctica anterior. Paso a describir estos cambios, exponiendo además el código fuente del registro para esta práctica:

srg_ppn.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY srg_ppn IS
5	GENERIC (magnitud: natural := 8);
6	
7	PORT (D: IN std_logic_vector (0 TO magnitud-1);
8	clk: IN std_logic;
9	Q: OUT std_logic_vector (0 TO magnitud-1));
10	
11	END srg_ppn;

En el caso de la declaración de la entidad, el único cambio realizado es ampliar el valor de *magnitud* de 4 a 8, para adaptarlo a este diseño.

El código de la arquitectura es el mismo que en la anterior práctica. Debo resañar sin embargo, un cambio en el diseño final del conversor con respecto a como viene descrito en el enunciado de la práctica.

En las conexiones del registro de salida de datos, he debido negar la señal *TC_CLK* antes de conectarla a la entrada de reloj. Ésto es debido a que todo el conjunto se adelantaba medio ciclo de reloj de no hacerlo, dando un resultado no contemplado en el enunciado. Este cambio se observará mejor en el código del conversor algo más adelante.

El resto de componentes del diseño: el monoestable y el registro de entrada de datos (un registro de desplazamiento serie-paralelo de 8 bits) son iguales a los realizados en las prácticas 1 y 2 respectivamente, por lo que obvio tanto su código como su explicación.

Paso por tanto a describir el código del conversor, que no es más sino la interconexión de los componentes que acabo de ir describiendo en estas últimas páginas.

convorsor_sp.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY convorsor_sp IS
5	PORT (entrada, clock : IN std_logic;
6	salida : OUT std_logic_vector (0 TO 7));
7	END biestableD;

Para el conversor que estoy diseñando, la entrada en serie y la entrada de reloj es del tipo std_logic, y la salida un vector de std_logic de 8 bits.

convorsor_sp_arq1.vhd	
1	ARCHITECTURE arq1 OF convorsor_sp IS
2	
3	COMPONENT ctr_div8 IS
4	PORT (clr, clk : IN std_logic;
5	TC : OUT std_logic);
6	END COMPONENT;
7	
8	COMPONENT biestableJK IS
9	PORT (J, K, clk, clr: IN std_logic;
10	Q, nQ: OUT std_logic);
11	END COMPONENT;
12	
13	COMPONENT puerta_and IS
14	PORT (a, b: IN std_logic;
15	c: OUT std_logic);
16	END COMPONENT;
17	
18	COMPONENT generador_reloj IS
19	PORT (Q, clk: IN std_logic;
20	s_CLK: OUT std_logic);
21	END COMPONENT;
22	
23	COMPONENT srg_ppn IS
24	GENERIC (magnitud: natural := 8);
25	PORT (D: IN std_logic_vector (0 TO magnitud-1);
26	clk: IN std_logic;
27	Q: OUT std_logic_vector (0 TO magnitud-1));
28	END COMPONENT;
29	
30	COMPONENT srg_sp8 IS
31	PORT(a,clk,clr: IN std_logic;

conversor_sp_arq1.vhd	
32	q: OUT std_logic_vector (0 TO 7));
33	END COMPONENT;
34	
35	COMPONENT monoestable IS
36	GENERIC (tw : time := 1 ns);
37	
38	PORT (disparo: IN std_logic;
39	Q: OUT std_logic := '1';
40	nQ: OUT std_logic := '0');
41	END COMPONENT;
42	
43	SIGNAL salida_JK, salida_clk, salida_TC, salida_TC_CLK, salida_monoestable_nQ, salida_nula, n_entrada, n_monoestable, n_TC_CLK, n_salida_monoestable_Q : std_logic := '0';
44	SIGNAL salida_monoestable_Q : std_logic := '1';
45	SIGNAL salida_Q, salida_D : std_logic_vector (0 TO 7);
46	
47	FOR ALL : biestableJK USE ENTITY WORK.biestableJK(arq1);
48	FOR ALL : puerta_and USE ENTITY WORK.puerta_and(arq);
49	FOR ALL : generador_reloj USE ENTITY WORK.generador_reloj(arq1);
50	FOR ALL : srg_ppn USE ENTITY WORK.srg_ppn(arq1);
51	FOR ALL : srg_sp8 USE ENTITY WORK.srg_sp8(arq1);
52	FOR ALL : monoestable USE ENTITY WORK.monoestable(arq1);
53	
54	BEGIN
55	n_entrada <= NOT entrada;
56	n_monoestable <= NOT salida_monoestable_Q;
57	n_TC_CLK <= NOT salida_TC_CLK;
58	n_salida_monoestable_Q <= NOT salida_monoestable_Q;
59	
60	flipflop : biestableJK PORT MAP (J => '1', K => '0', clk => n_entrada, clr => n_monoestable, Q => salida_JK, nQ => salida_nula);
61	generador: generador_reloj PORT MAP (Q => salida_JK, clk => clock, s_CLK => salida_clk);
62	divisor : ctr_div8 PORT MAP (clr => n_salida_monoestable_Q, clk => salida_clk, TC => salida_TC);
63	and1 : puerta_and PORT MAP (a => salida_TC, b => salida_clk, c => salida_TC_CLK);
64	mono : monoestable PORT MAP (disparo => n_TC_CLK, Q => salida_monoestable_Q, nQ => salida_monoestable_nQ);
65	desplazador : srg_ppn PORT MAP (D => salida_Q, clk => n_TC_CLK, Q => salida_D);
66	conversor_sp : srg_sp8 PORT MAP (a => entrada, clk => salida_clk, clr => '0', q => salida_Q);
67	
68	salida <= salida_D;
69	END arq1;

Para empezar, se declaran los diferentes componentes necesarios para el conversor, en este orden: contador de divisor por 8 (*ctr_div8*), biestable JK, puerta lógica AND, generador de reloj, registro de salida de datos (*srg_ppn*), registro de entrada de datos (*srg_sp8*), y el monoestable; entre las líneas 3 y 41.

Después se declaran las señales intermedias necesarias para la interconexión de estos componentes (sus salidas y la negación de varias de ellas (de la entrada del conversor, de la salida del monoestable, y de la señal TC_CLK); entre las líneas 43 y 45.

Entre las líneas 47 y 52 se definen las configuraciones para todos los componentes. Luego se realizan las negaciones que son necesarias, entre las líneas 55 y 58. Tras ello, se procede a conectar los componentes entre sí. Cabe destacar la nomenclatura que he usado para las interconexiones (conector => señal), para evitar errores debido a la cierta complejidad y cantidad de componentes del diseño.

Para finalizar, se asigna la señal de salida del conversor serie-paralelo a la salida del registro de salida de datos, para tener el conversor en funcionamiento.

Tan solo falta describir la prueba para el simulador.

prueba_conversor_sp.vhd	
1	LIBRARY ieee;
2	USE ieee.std_logic_1164.ALL;
3	
4	ENTITY prueba_generica IS
5	END prueba_generica;

prueba_conversor_sp_arq.vhd	
1	ARCHITECTURE arq1 OF prueba_conversor_sp IS
2	
3	COMPONENT conversor
4	PORT (entrada, clock : IN std_logic;
5	salida : OUT std_logic_vector (7 DOWNT0 0));
6	END COMPONENT;
7	
8	FOR ALL : conversor USE ENTITY WORK.conversor_sp(arq1);
9	
10	CONSTANT duracion_ciclo: time := 1 ns;
11	SIGNAL CLK_s, entrada_S : std_logic := '0';
12	SIGNAL salida_s : std_logic_vector (0 TO 7);
13	
14	BEGIN
15	
16	conversor1 : conversor PORT MAP (entrada_s, CLK_s, salida_s);
17	
18	reloj: PROCESS

prueba_conversor_sp_arq.vhd	
19	BEGIN
20	WAIT FOR duracion_ciclo;
21	CLK_s <= '1';
22	WAIT FOR duracion_ciclo;
23	CLK_s <= '0';
24	END PROCESS reloj;
25	
26	prueba: PROCESS
27	BEGIN
28	
29	entrada_s <='1',
30	'0' AFTER 4 ns, -- bit de arranque
31	'1' AFTER 6 ns, -- D7
32	'0' AFTER 8 ns, -- D6
33	'0' AFTER 10 ns, -- D5
34	'1' AFTER 12 ns, -- D4
35	'1' AFTER 14 ns, -- D3
36	'0' AFTER 16 ns, -- D2
37	'1' AFTER 18 ns, -- D1
38	'0' AFTER 20 ns, -- D0
39	'1' AFTER 22 ns, -- bit de parada 1
40	'1' AFTER 24 ns; -- bit de parada 2
41	
42	WAIT;
43	END PROCESS prueba;
44	
45	END arq1;

Como en las anteriores pruebas, primero declaro el componente (lineas 3 a 6), le asigno una configuración (línea 8), declaro las señales tanto del reloj como las necesarias para la prueba (lineas 10 y 12) e instancio el componente (línea 16).

Obviando el funcionamiento del proceso reloj, las señales pasadas a la entrada del componente en el proceso de prueba son las correspondientes a las del cronograma que aparece en el enunciado en la figura 41 (un bit de arranque '0', una secuencia de 8 bits "10011010" y dos bits de parada "11"). Ésto da a lugar a un cronograma como éste:

