

P4

Diseño e implementación de un contador

1.- Introducción y objetivos

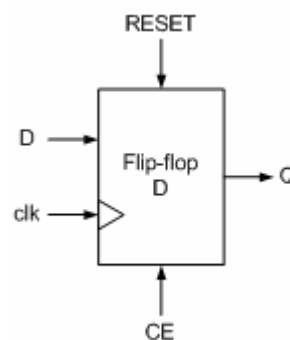
En esta práctica se van a implementar un contador cíclico de 0 a 9. Además de la descripción, la simulación y la síntesis, se completarán todas las fases del diseño digital en FPGA y se observará el resultado en una placa de desarrollo.

Posteriormente se añadirá al diseño el decodificador de 7 segmentos de la práctica P3 para poder observar la cuenta en un display de 7 segmentos. Con esto se persigue realizar un diseño con varios niveles jerárquicos familiarizando al alumno el empleo de las cláusulas COMPONENT y PORT MAP (que ya se han empleado en los test-bench).

2.- Descripción de un flip-flop tipo D en VHDL

En esta práctica vamos a emplear flip-flops tipo D con CE y RESET asíncrono para realizar el contador, así que habrá que crear un código VHDL de nombre **ffD_reset** para poder instanciar este componente en una entidad jerárquicamente superior. Crear un *VHDL Module* a tal fin con los puertos que se muestran en la figura. La arquitectura que describe la funcionalidad de este componente se muestra abajo.

```
38 architecture Behavioral of ffD_reset is
39
40 begin
41
42 process (clk, reset)
43 begin
44     if reset='1' then
45         q <= '0';
46     elsif (clk'event and clk='1') then
47         if ce = '1' then
48             q <= d;
49         end if;
50     end if;
51 end process;
52
53 end Behavioral;
```



Sin entrar en pormenores, las ecuaciones booleanas que se implementaron en las prácticas anteriores son procesos implícitos, aunque también pueden expresarse según la sintaxis de la figura de arriba (proceso explícito empleando VHDL comportamental). En la línea 42 aparece la denominada lista de sensibilidad del proceso. Éste no se ejecutará a menos que haya un cambio en alguna de las señales presentes en esta lista.

Los procesos explícitos son muy importantes, ya que podemos describir un sistema digital de una manera algorítmica, facilitando la vida a los ingenieros electrónicos y reduciendo considerablemente el tiempo de diseño. El sintetizador se encargará de *traducir* esta descripción en el hardware oportuno (en este caso, el biestable de la figura).

3.- Diseño del contador

El contador a implementar se especifica a través de la siguiente tabla de transiciones:

QA	QB	QC	QD	QA ⁺	QB ⁺	QC ⁺	QD ⁺
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Como se aprecia, son necesarios 4 flip-flops para implementar el contador. Por otra parte, el estado siguiente de los flip-flops tipo D se rige por esta ecuación: $Q^+ = D$. Así,

las entradas a los 4 biestables se obtienen directamente de la parte derecha de la tabla. Simplificando, las funciones lógicas que deben atacar las entradas a los biestables son:

$$D_A = Q_A \overline{Q_D} + Q_B \overline{Q_C} Q_D$$

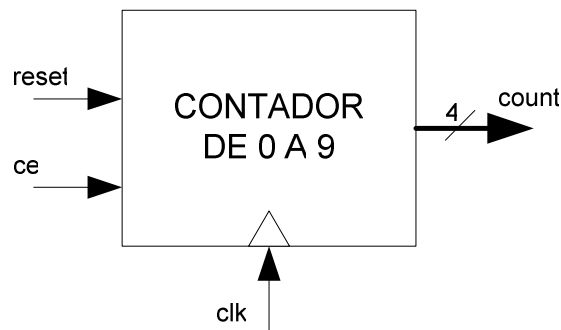
$$D_B = Q_B \overline{Q_C} + Q_B \overline{Q_D} + \overline{Q_B} Q_C Q_D = Q_B (\overline{Q_C} + \overline{Q_D}) + \overline{Q_B} Q_C Q_D$$

$$D_C = \overline{Q_A} \cdot \overline{Q_C} Q_D + \overline{Q_A} Q_C \overline{Q_D} = \overline{Q_A} (Q_C \oplus Q_D)$$

$$D_D = \overline{Q_D}$$

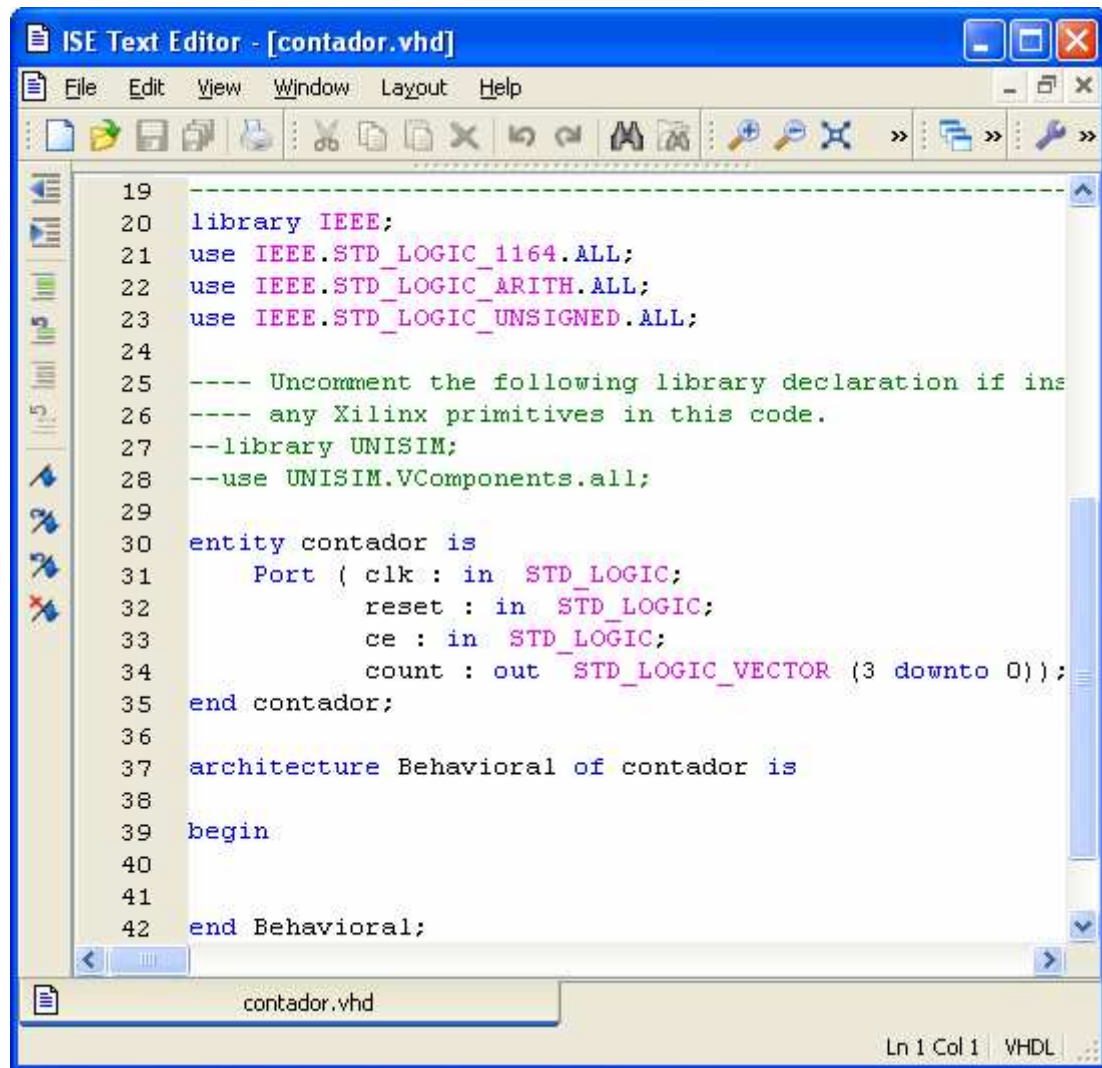
4.- Descripción VHDL del contador

Creemos un *VHDL Module*, llamado **contador**, con los puertos que aparecen en la figura:



Port Name	Direction	Bus	MSB	LSB
clk	in	<input type="checkbox"/>		
reset	in	<input type="checkbox"/>		
ce	in	<input type="checkbox"/>		
count	out	<input checked="" type="checkbox"/>	3	0
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

Una vez finalizados los pasos del asistente, la plantilla generada para **contador.vhd** tiene la pinta que se muestra a continuación:



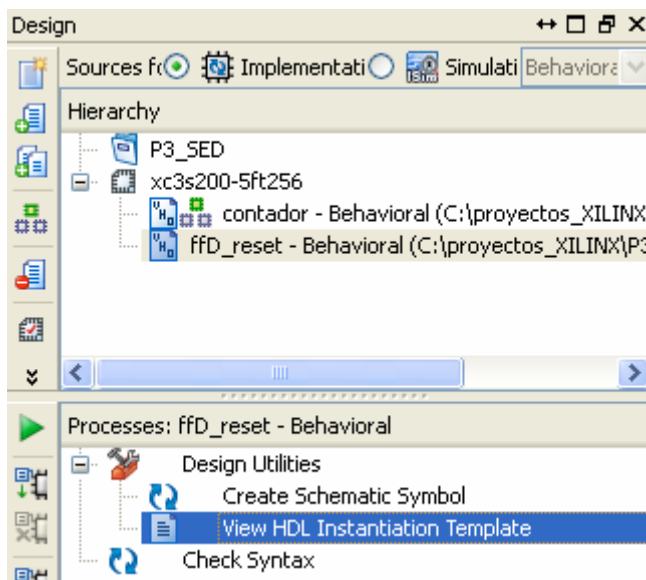
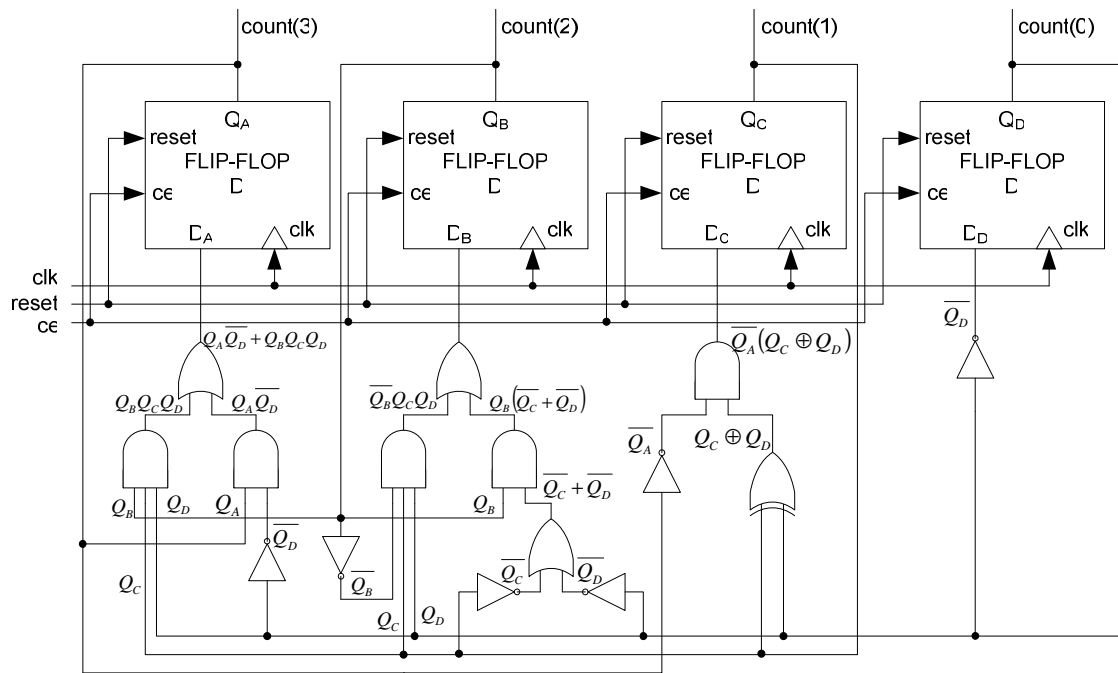
```
19
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if ins
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity contador is
31     Port ( clk : in  STD_LOGIC;
32           reset : in  STD_LOGIC;
33           ce : in  STD_LOGIC;
34           count : out  STD_LOGIC_VECTOR (3 downto 0));
35 end contador;
36
37 architecture Behavioral of contador is
38
39 begin
40
41
42 end Behavioral;
```

Aquí tendremos que describir el contador que deseamos implementar, y que se muestra en el circuito de la siguiente página.

Como puede observarse, debemos usar el flip-flop tipo D que creamos en el apartado 2 para implementar el contador. De hecho, son necesarios 4 componentes de este tipo. Para ello debemos emplear las cláusulas COMPONENT y PORT MAP que ya hemos usado en las prácticas anteriores de manera automática cuando generamos los test-bench. Para poder generar estas partes del código sin necesidad de escribir, operamos como se muestra a continuación.

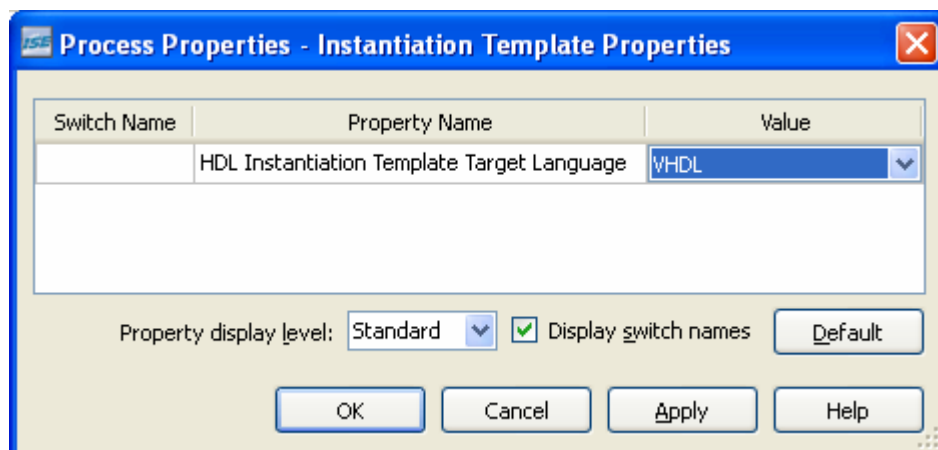
IMPORTANTE:

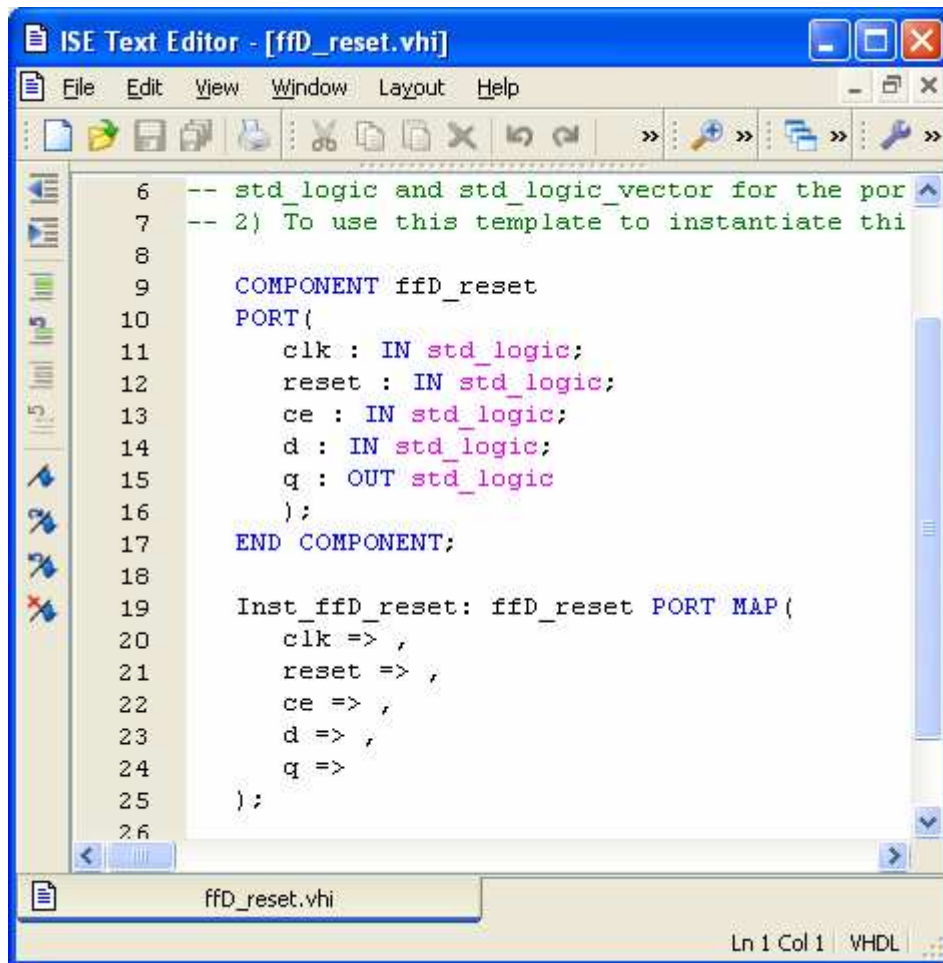
Hacer clic con el botón derecho del ratón sobre *View HDL Instantiation Template* y elegir *Process Properties* para asegurarnos que el *HDL Instantiation Template Target Language* sea **VHDL**. Si pone **Verilog**, hay que cambiarlo. Recordad que estamos describiendo hardware usando VHDL, no Verilog.



Marcamos el fichero (o *source*) del cual queremos generar las instancias COMPONENT y PORT MAP (es decir, **ffd_reset.vhd**, ver figura). En la ventana de procesos, seleccionamos *View HDL Instantiation Template*.

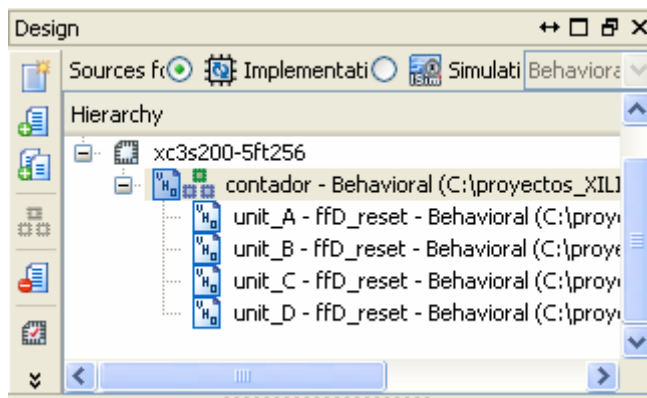
Una vez comprobado todo esto, hacemos doble clic en *View HDL Instantiation Template* y creamos el fichero **ffd_reset.vhi** donde *vhi* es el acrónimo de *VHDL Instantiation*. Si todo lo hemos realizado correctamente, el aspecto que presenta esta porción de código debe ser parecido a la que figura en la siguiente página.



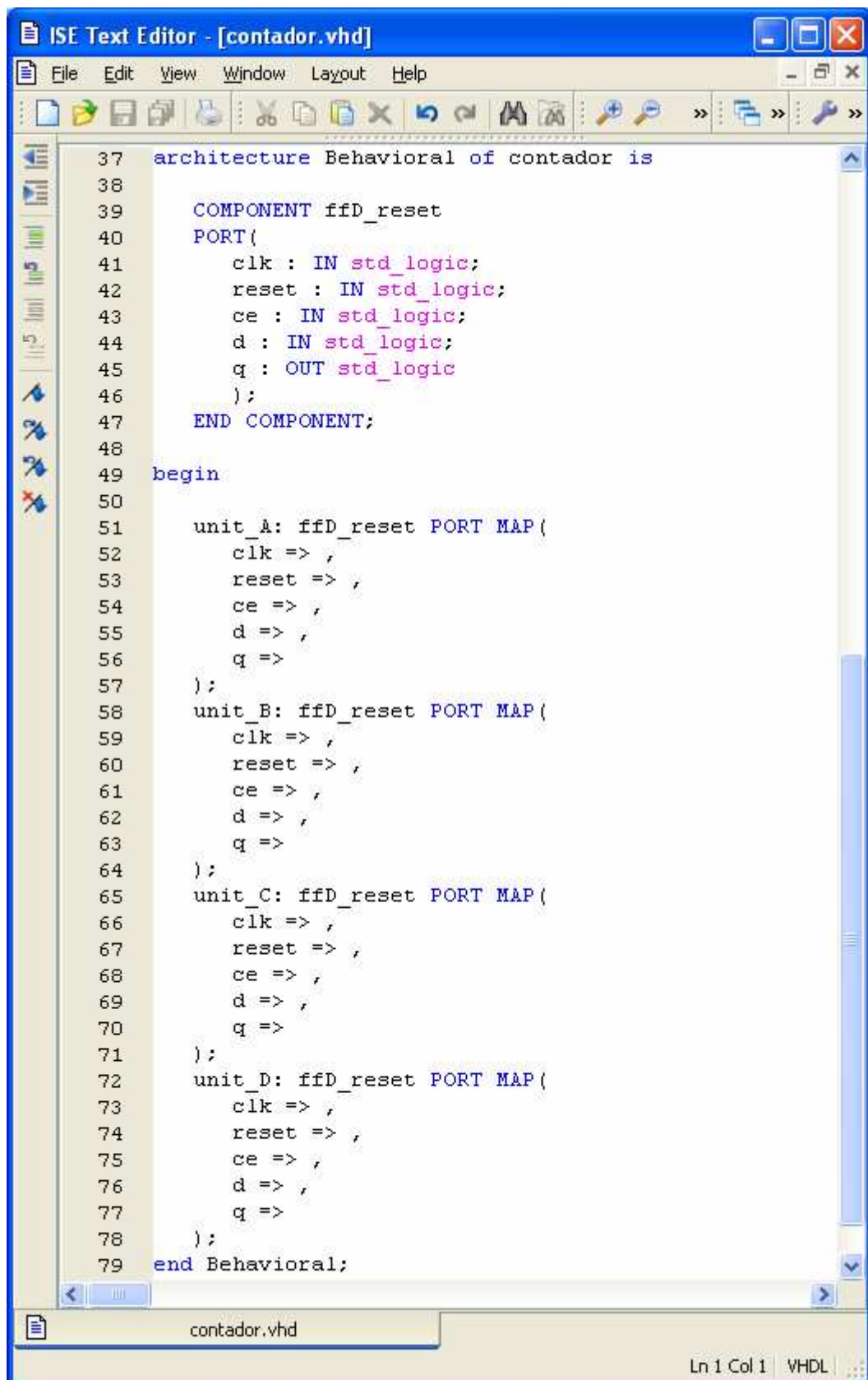


A continuación deberemos copiar la parte declarativa del componente (de la línea 9 a la 17 en la figura) y pegarlo en la zona declarativa del contador. Debemos hacer lo mismo con la instancia (de la línea 19 a la 25). Todo esto se especifica en los dos puntos que aparecen en los comentarios en verde del propio documento generado.

En realidad, el PORT MAP debemos pegarlo 4 veces, ya que vamos a necesitar 4 biestables tipo D. La arquitectura del fichero **contador.vhd** debe quedar como se presenta en la figura de la siguiente página. Nótese que el nombre de los PORT MAPs ha sido cambiado a los nombres arbitrarios *unit_A*, *unit_B*, *unit_C* y *unit_D* (correspondientes a los biestables A, B, C y D), ya que el nombre con el que se identifica cada PORT MAP debe ser distinto.



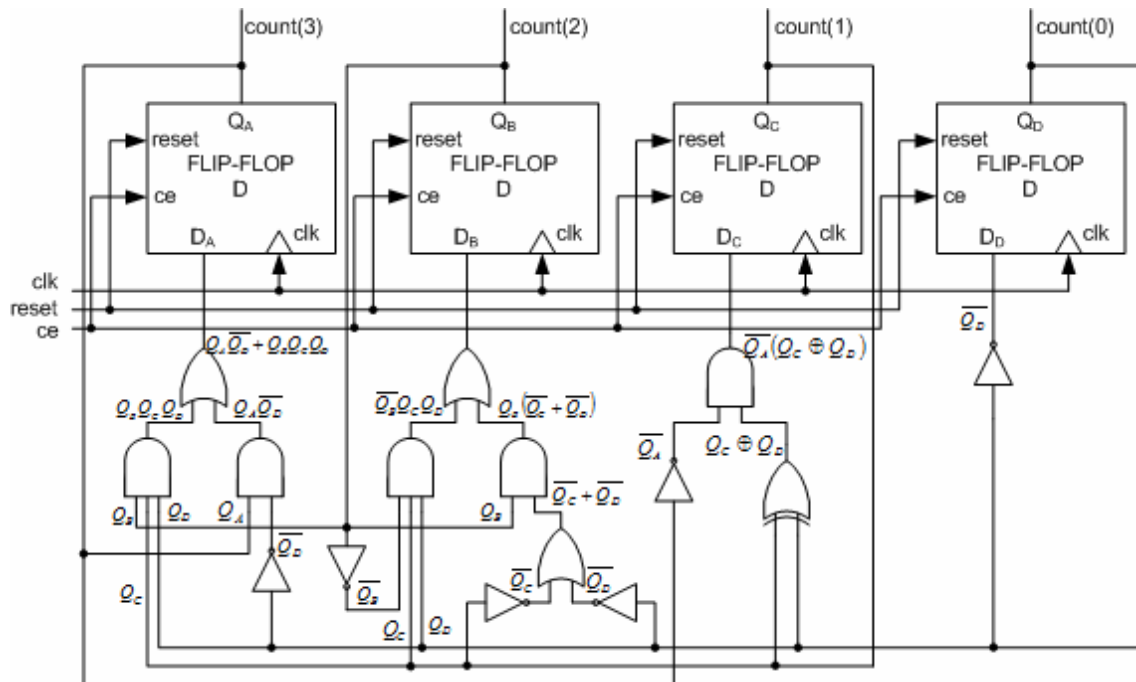
Se puede observar la ventana correspondiente a la jerarquía del diseño del sistema para ver cómo ha cambiado después de las últimas operaciones realizadas. Ahora, del *top* del diseño (**contador**) cuelgan los 4 biestables tipo D que se sitúan en un nivel jerárquicamente inferior (parecen 4 códigos, pero es el mismo repetido 4 veces).



```
37 architecture Behavioral of contador is
38
39     COMPONENT ffD_reset
40     PORT(
41         clk : IN std_logic;
42         reset : IN std_logic;
43         ce : IN std_logic;
44         d : IN std_logic;
45         q : OUT std_logic
46     );
47     END COMPONENT;
48
49 begin
50
51     unit_A: ffD_reset PORT MAP(
52         clk => ,
53         reset => ,
54         ce => ,
55         d => ,
56         q =>
57     );
58     unit_B: ffD_reset PORT MAP(
59         clk => ,
60         reset => ,
61         ce => ,
62         d => ,
63         q =>
64     );
65     unit_C: ffD_reset PORT MAP(
66         clk => ,
67         reset => ,
68         ce => ,
69         d => ,
70         q =>
71     );
72     unit_D: ffD_reset PORT MAP(
73         clk => ,
74         reset => ,
75         ce => ,
76         d => ,
77         q =>
78     );
79 end Behavioral;
```

Ahora sólo queda realizar las conexiones acordes al circuito que aparecía al principio del apartado y que se repite aquí por conveniencia. Empezamos por declarar las señales *qa*, *qb*, *qc*, *qd*, *da*, *db*, *dc*, *dd*, que serán conectadas a las entradas y salidas de datos de

los 4 biestables. Las entradas de *clk*, *reset* y *ce* son comunes a todos los biestables y quedan conectados directamente a los puertos de entrada con el mismo nombre.



```

48
49 signal qa, qb, qc, qd : std_logic;
50 signal da, db, dc, dd : std_logic;
51

```

Las entradas a los biestables (*da*, *db*, *dc* y *dd*) se expresan según se muestra a continuación, acorde con el circuito de arriba. También se muestra uno de los biestables (el D) con el mapeo ya realizado (hacer lo mismo con A, B, y C teniendo en cuenta que las entradas *d* y *q* son distintas en cada biestable).

```

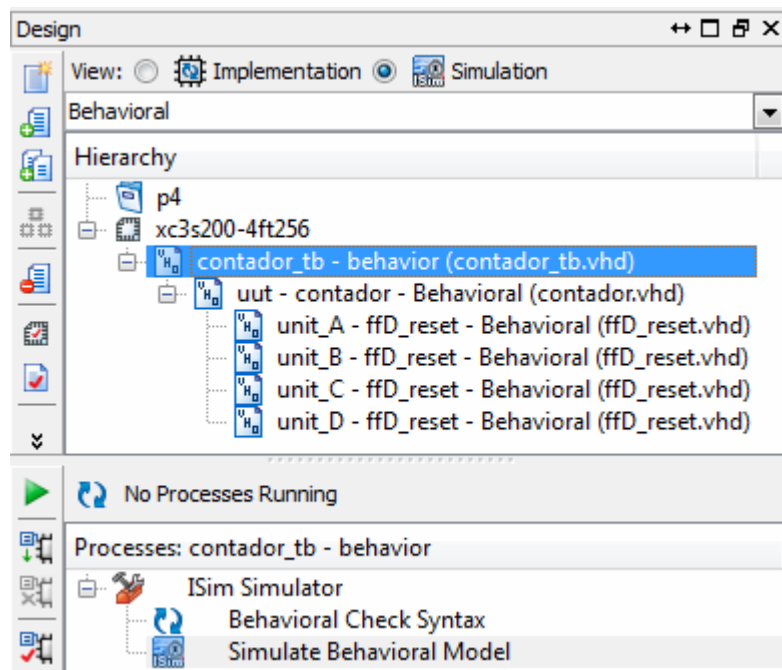
75 unit_D: ffD_reset PORT MAP(
76     clk => clk,
77     reset => reset,
78     ce => ce,
79     d => dd,
80     q => qd
81 );
82
83 dd <= not qd;
84 dc <= (not qa) and (qc xor qd);
85 db <= (qb and (not qc or not qd)) or (not qb and (qc and qd));
86 da <= (qa and not qd) or ((qb and qc) and qd);
87
88 count <= qa & qb & qc & qd;
89
90 -- count(0) <= qd;
91 -- count(1) <= qc;
92 -- count(2) <= qb;
93 -- count(3) <= qa;

```


Sólo nos queda conectar el puerto de salida del contador (el bus *count*) a las salidas de los biestables. Para ello usamos el operador concatenación (&), por ejemplo (aunque se puede hacer como se muestra comentado en color verde).

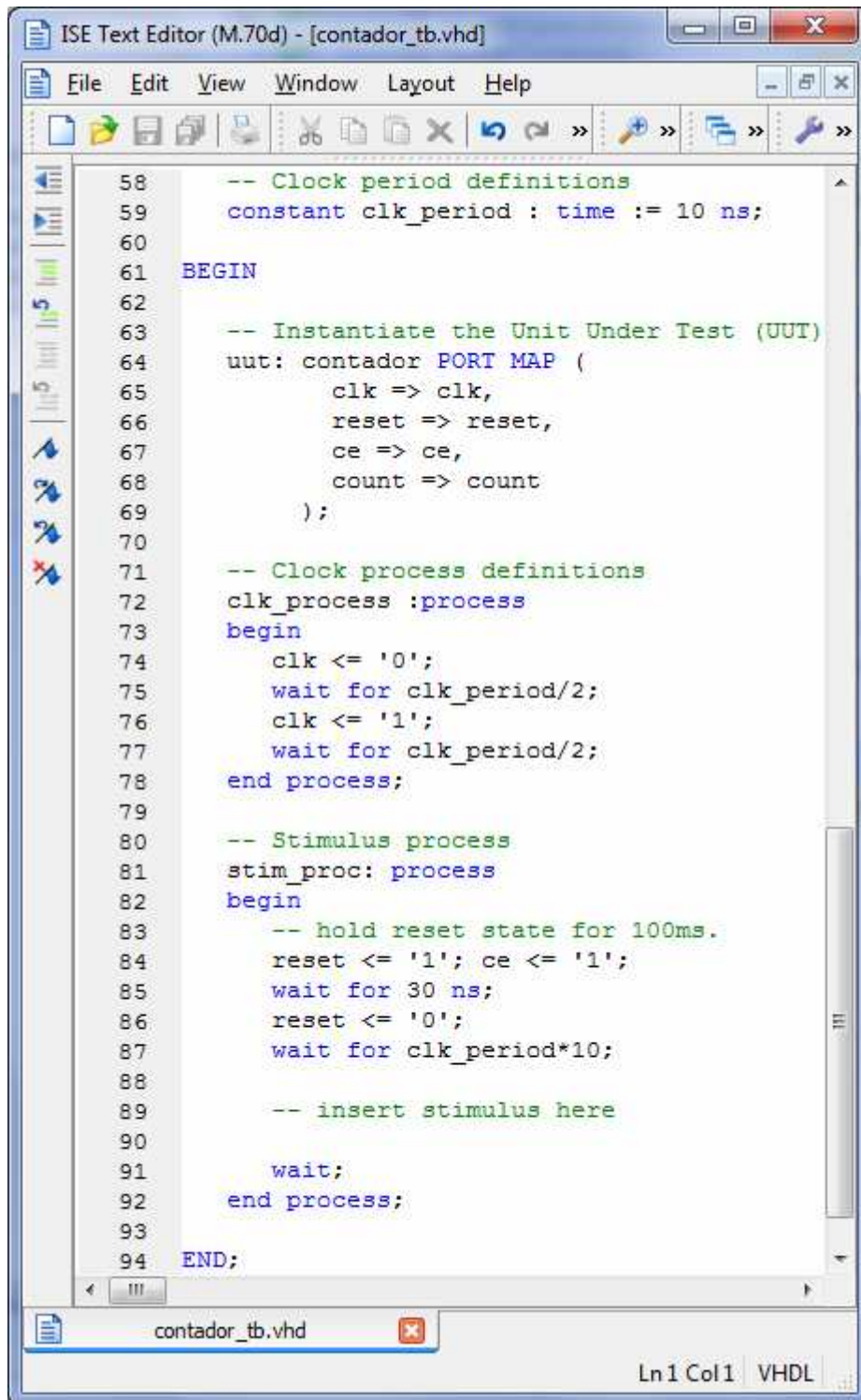
5.- Simulación del contador

Creamos una *New Source*, que será un *VHDL Test Bench* que llamaremos **contador_tb.vhd** y que asociaremos a **contador.vhd** en el asistente. Cuando seleccionemos la vista para simulación al terminar con el asistente, ésta presentará el aspecto que aparece en la figura.



La plantilla que genera el asistente es muy completa. El sintetizador ha detectado que es un diseño secuencial y que la señal de reloj es *clk*. Para poder visualizar mejor la simulación, cambiaremos el valor del periodo de la señal de reloj a 10 ns y rellenaremos el último proceso del código como se muestra en la figura (cambiando el valor del tiempo de la línea 85 a 30 ns, por ejemplo).



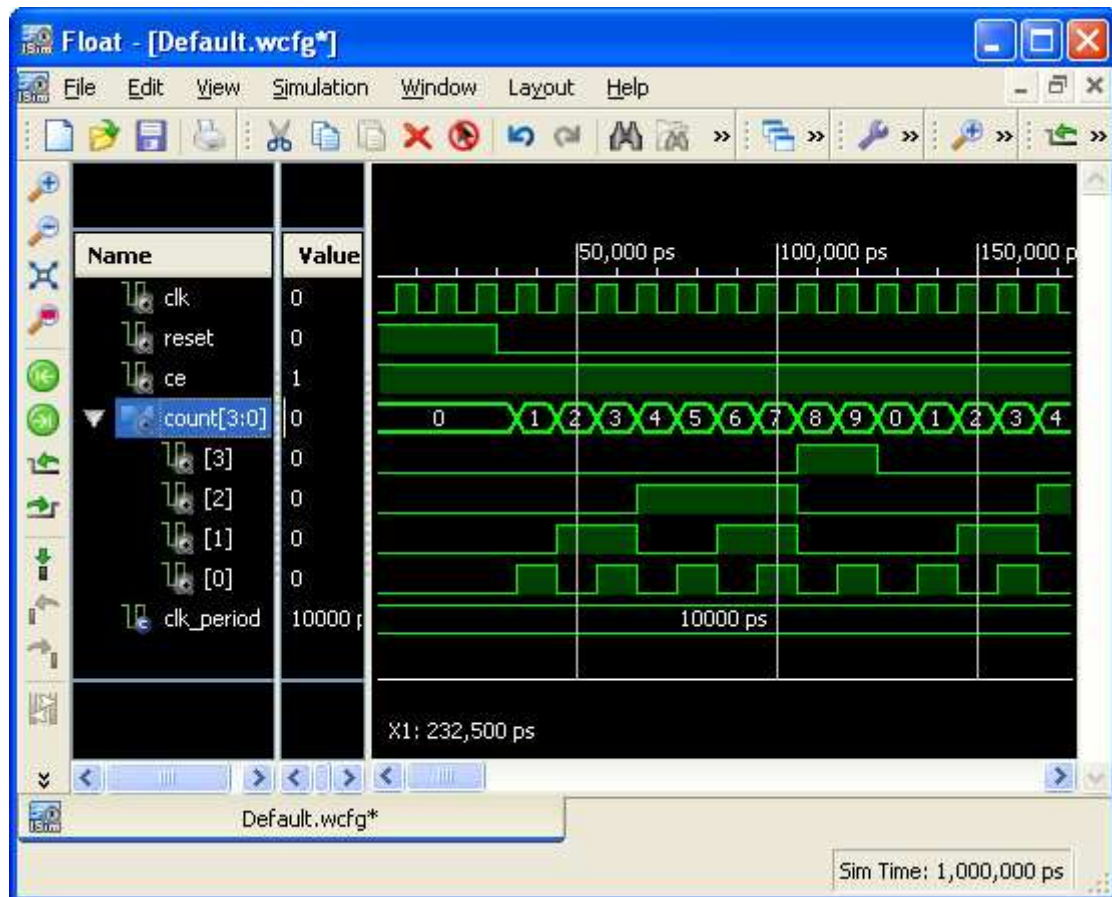
The image shows a screenshot of the ISE Text Editor window. The title bar reads "ISE Text Editor (M.70d) - [contador_tb.vhd]". The menu bar includes "File", "Edit", "View", "Window", "Layout", and "Help". The toolbar contains various icons for file operations, editing, and simulation. The main text area displays VHDL code for a testbench named "contador_tb.vhd". The code is as follows:

```
58  -- Clock period definitions
59  constant clk_period : time := 10 ns;
60
61  BEGIN
62
63  -- Instantiate the Unit Under Test (UUT)
64  uut: contador PORT MAP (
65      clk => clk,
66      reset => reset,
67      ce => ce,
68      count => count
69  );
70
71  -- Clock process definitions
72  clk_process :process
73  begin
74      clk <= '0';
75      wait for clk_period/2;
76      clk <= '1';
77      wait for clk_period/2;
78  end process;
79
80  -- Stimulus process
81  stim_proc: process
82  begin
83      -- hold reset state for 100ms.
84      reset <= '1'; ce <= '1';
85      wait for 30 ns;
86      reset <= '0';
87      wait for clk_period*10;
88
89      -- insert stimulus here
90
91      wait;
92  end process;
93
94  END;
```

The status bar at the bottom indicates "Ln 1 Col 1 VHDL".

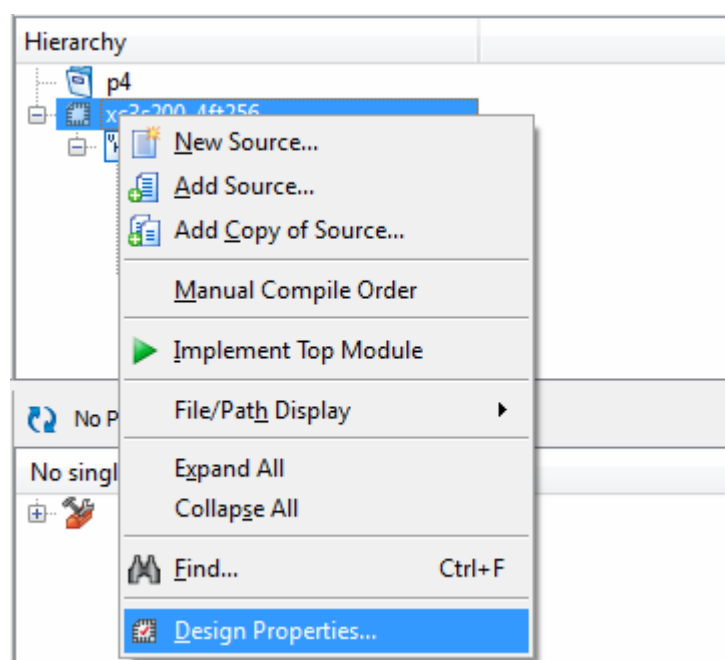
Al ejecutar la simulación y operar con el zoom, veremos un cronograma como el de la figura. Puede apreciarse que, tras un *reset* de 30 ns, el circuito empieza a contar cíclicamente de 0 a 9, como pretendíamos al inicio de la práctica.

RECORDATORIO: Una sólo simulación simultáneamente.



5.- Selección de la placa de desarrollo

Para los siguientes pasos de la fase de diseño es necesario especificar el tipo de dispositivo FPGA sobre el que se a implementar las funciones lógicas. Si no se indicó al crear el proyecto, habrá que seleccionar la FPGA ahora.



Usar el botón derecho en la descripción de la FPGA actual y seleccionar *Design Properties*. Seleccionar la FPGA de la figura de abajo (Spartan3, XC3S200 y FT256).

Property Name	Value
Product Category	All
Family	Spartan3
Device	XC3S200
Package	FT256
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>

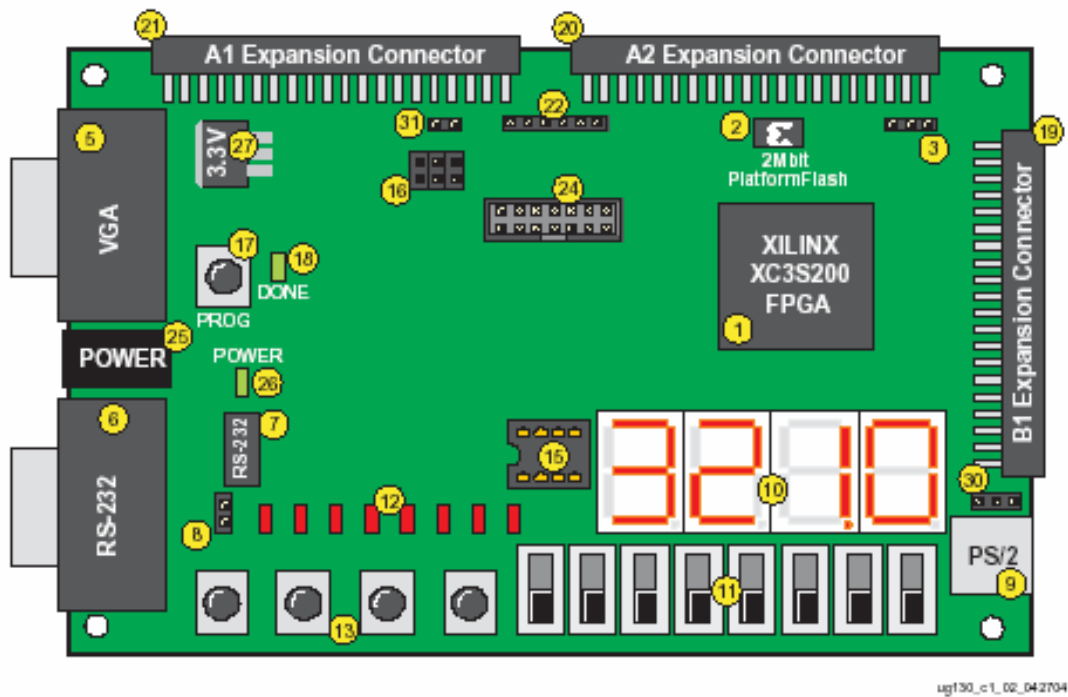
6.- Asignación de pines de la FPGA

En este punto el diseñador puede abstraerse de todos los procesos que tienen lugar a partir de ahora. No obstante, el entorno permite modificar y seleccionar parámetros y características en todos estos procesos, a conveniencia por parte del propio diseñador.

Una de las cosas que puede hacer es lo que se denomina **restricciones de usuario** (**user constraints** en inglés). Estas restricciones pueden ser temporales (especificar unos ciertos requerimientos de velocidad, por ejemplo) o de ubicación (en qué lugar físico de la FPGA implementaremos nuestro circuito lógico) o de asignación de pines (en qué pines físicos del dispositivo FPGA pondremos nuestros puertos de entrada y salida).

La placa de desarrollo suele constar de una FPGA y ciertos periféricos conectados a ella que son útiles para comprobar en el propio circuito nuestro diseño. Estos periféricos suelen ser interruptores, botones, leds, displays, conectores serie, VGA y otros muchos.

Un ejemplo se presenta en el siguiente esquema:



En nuestro diseño deseamos conectar las entradas **reset** y **ce** a 2 interruptores como los de la figura (11), la entrada de reloj, **clk**, a un botón (13), y la salida **count** a 4 leds como los de la figura (12). Normalmente los fabricantes proveen en las hojas de características de sus placas de una serie de tablas que asocian cada periférico con un pin de la FPGA:

Table 4-1: **Slider Switch Connections**

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Table 4-2: **Push Button Switch Connections**

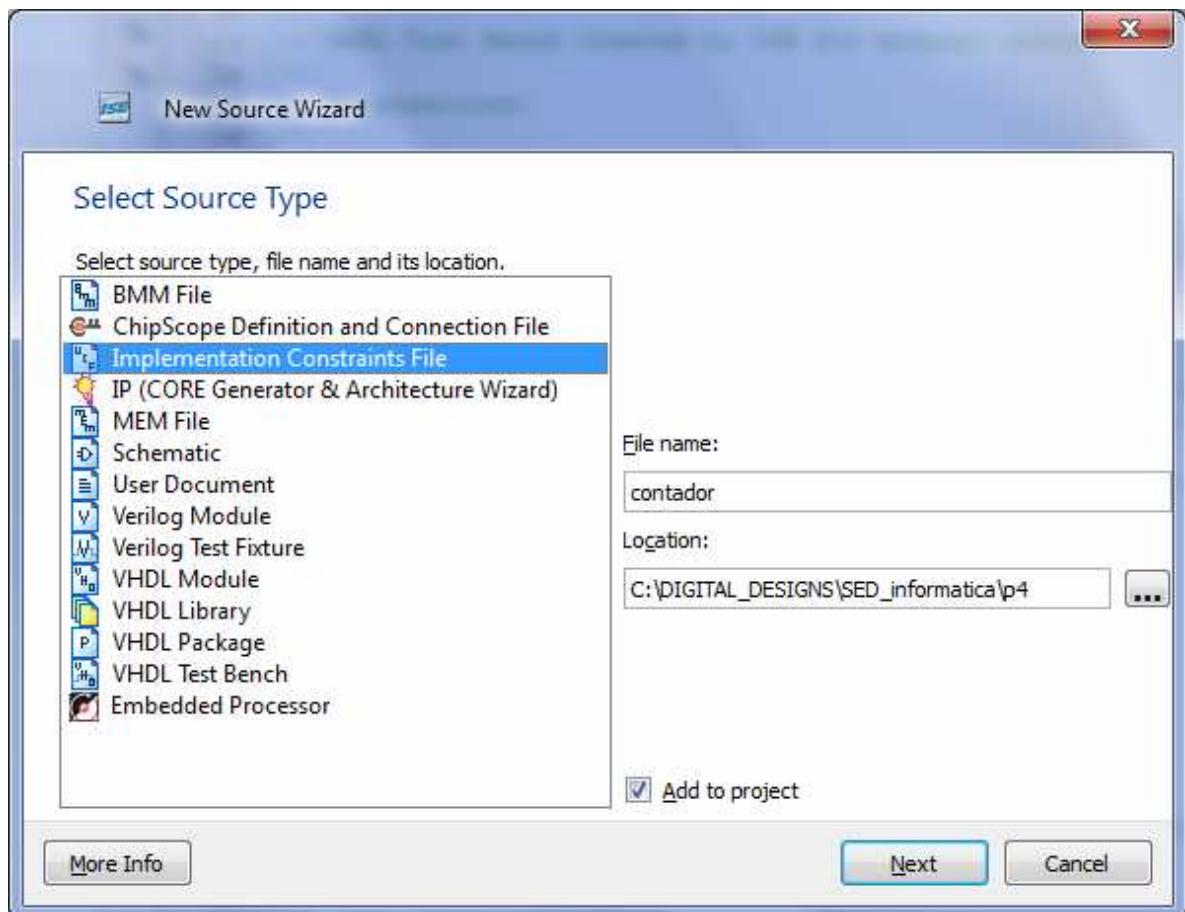
Push Button	BTN3	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Table 4-3: **LED Connections to the Spartan-3 FPGA**

LED	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

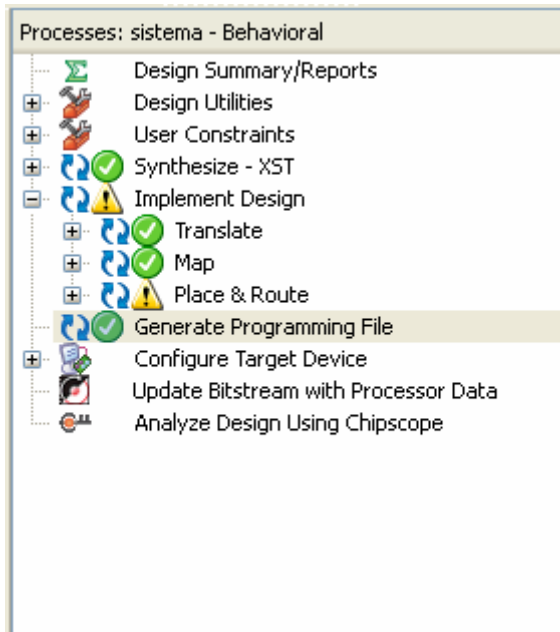
Para realizar la asignación debemos crear una nueva fuente (*New Source*). En el asistente seleccionamos *Implementation Constraints File*. Esto generará un fichero de texto con extensión UCF (acrónimo de *User Constraints File*) en el que introduciremos la asignación de pines a cada puerto de entrada o salida de nuestro diseño.

En la figura aparece un ejemplo de asignación de pines para una FPGA Spartan-3 XS200. El profesor proporcionará la información de los pines de la FPGA de la que se disponga ese día en el laboratorio.



En la última línea de **contador.ucf**, se especifica que no usaremos una línea de reloj dedicada a tal fin para nuestra señal de reloj (con ello podremos conectar nuestro puerto de reloj a cualquier pin, por ejemplo a un botón de la placa).

7.- Implementación y generación del fichero de configuración



Los últimos pasos del diseño se realizan ejecutando el comando *Generate Programming File*. Haciendo esto, se ejecutarán todos los pasos necesarios para implementar el diseño (*Implement Design: Translate, Map, y Place & Route*). Si todo va bien, aparecerá en cada una de estas fases un icono confirmativo de color verde, como en la figura, salvo un *warning*, que nos avisa que la línea de reloj no es dedicada (así lo impusimos) y el diseño no será óptimo en velocidad.

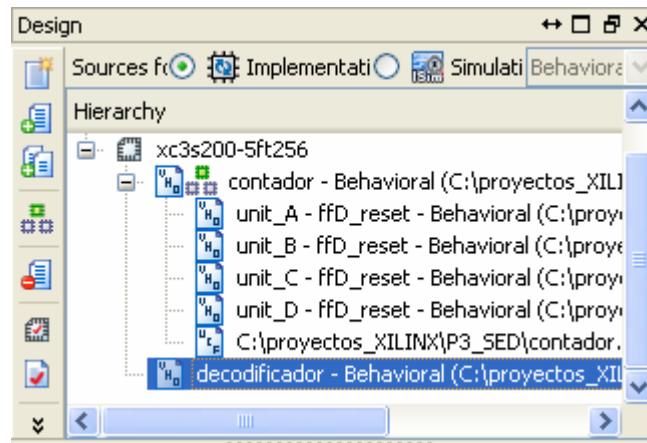
Si todo se ha ejecutado satisfactoriamente se generará el archivo **contador.bit**, con toda la información para configurar satisfactoriamente la FPGA. El profesor le indicará cómo.

Al configurar la FPGA correctamente, los leds seleccionados como salida empezarán a contar en binario a media que pulsamos el botón de reloj (siempre que *reset* esté a 0 y *ce* a 1)

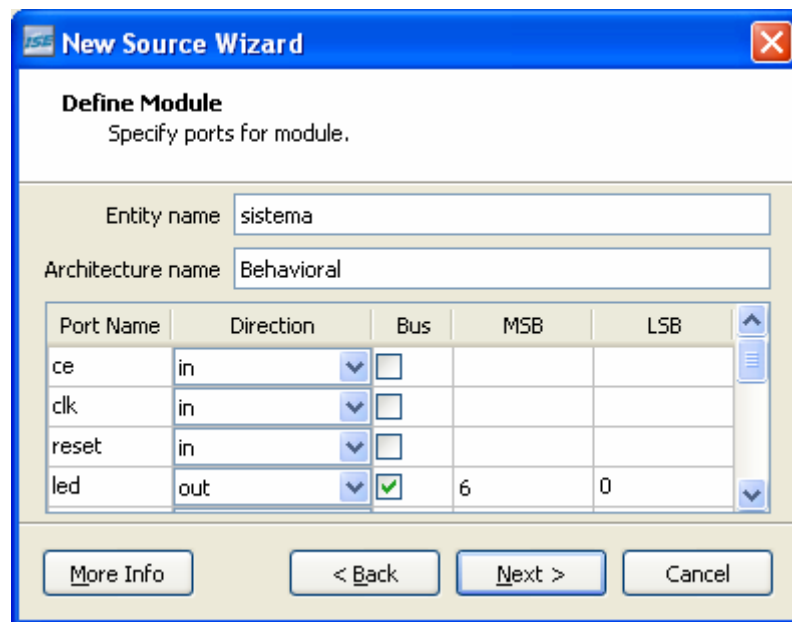
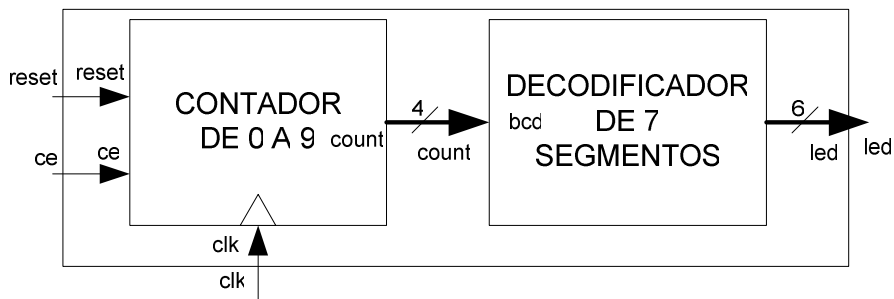
QA	QB	QC	QD
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1

8.- Visualización de la salida en un display de 7 segmentos

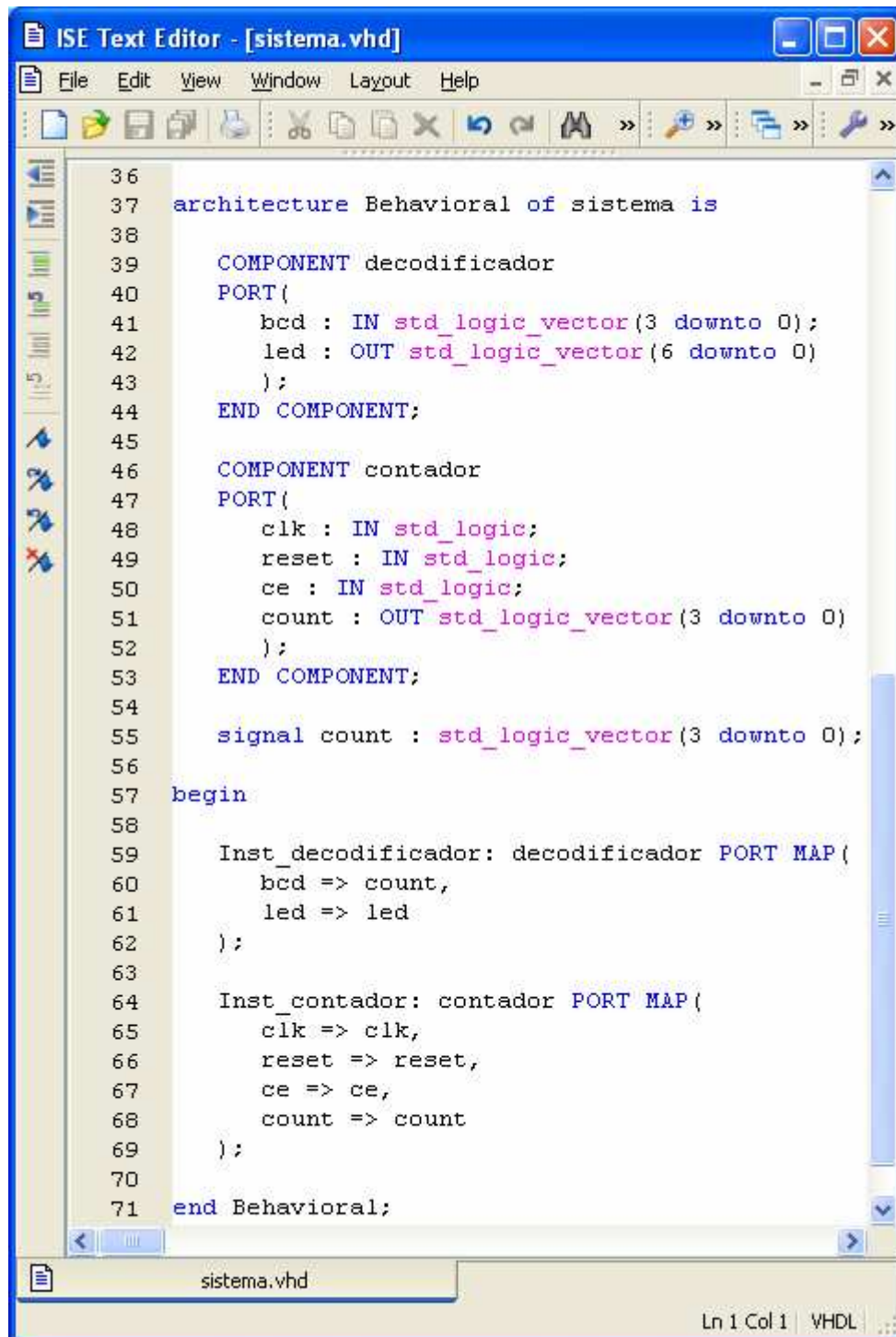
A continuación deseamos visualizar nuestra cuenta en un display de 7 segmentos, así que habrá que importar el código VHDL creado en la práctica anterior (se aconseja primero copiar el fichero en cuestión a la carpeta donde se localiza nuestro proyecto actual) usando el comando *Add Source*. Si hacemos esto correctamente, aparecerá el decodificador en la ventana de jerarquía:



Creamos un nuevo *VHDL Module* llamado por ejemplo **sistema.vhd**, en el que integraremos el contador y el decodificador tal como se aprecia en la figura.



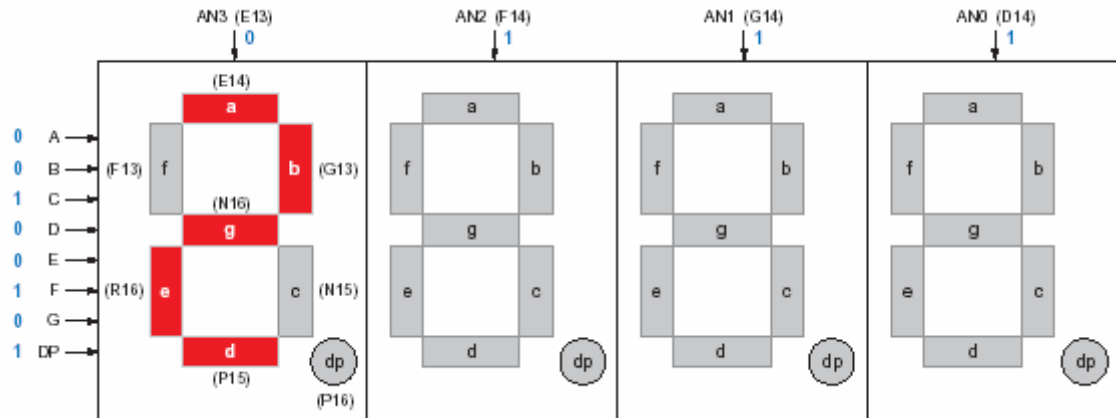
En la nueva plantilla generada, debemos colocar un COMPONENT y un PORT MAP, por cada elemento que queremos integrar. Para realizar este paso de una manera rápida hay que seguir los pasos especificados en el apartado 4. Sólo resta completar el conexionado rellenando la parte derecha de los PORT MAPs (habrá que crear una señal interna *std_logic_vector(3 downto 0)* para conectar los dos bloques. Haciendo esto, la descripción del sistema total queda como se aprecia en la figura:



```
36
37 architecture Behavioral of sistema is
38
39     COMPONENT decodificador
40     PORT(
41         bcd : IN std_logic_vector(3 downto 0);
42         led : OUT std_logic_vector(6 downto 0)
43     );
44     END COMPONENT;
45
46     COMPONENT contador
47     PORT(
48         clk : IN std_logic;
49         reset : IN std_logic;
50         ce : IN std_logic;
51         count : OUT std_logic_vector(3 downto 0)
52     );
53     END COMPONENT;
54
55     signal count : std_logic_vector(3 downto 0);
56
57 begin
58
59     Inst_decodificador: decodificador PORT MAP(
60         bcd => count,
61         led => led
62     );
63
64     Inst_contador: contador PORT MAP(
65         clk => clk,
66         reset => reset,
67         ce => ce,
68         count => count
69     );
70
71 end Behavioral;
```

El siguiente paso consistiría en realizar una simulación del sistema total. No obstante, ya hemos comprobado que el contador y el decodificador funcionan correctamente y en el nivel superior únicamente los conectamos (no generamos lógica). En este caso, estimamos que la simulación no es necesaria y no saltamos ese paso.

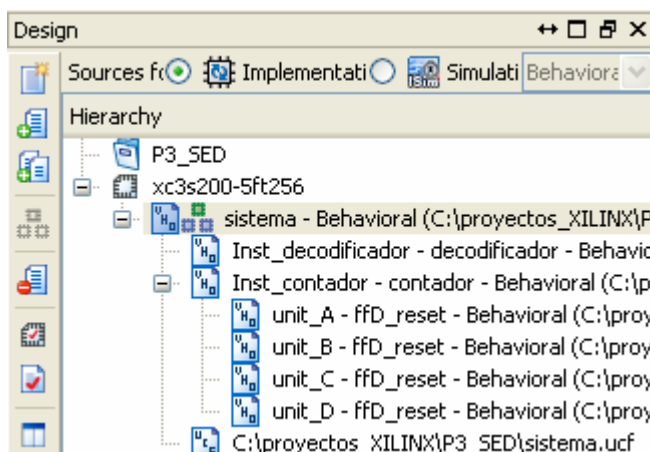
Por último, hay que crear un nuevo fichero UCF en el que se debe tener en cuenta que los pines de salida han cambiado. Borramos, pues, **contador.ucf** del proyecto y generamos uno nuevo (**sistema.ucf**), teniendo en cuenta para la salida los pines conectados al display:



```

ISE Text Editor - [sistema.ucf]
File Edit View Window Layout Help
1 NET "ce" LOC = K13;
2 NET "clk" LOC = L14;
3 NET "reset" LOC = K14;
4 NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
5 NET "led[0]" LOC = E14;
6 NET "led[1]" LOC = G13;
7 NET "led[2]" LOC = N15;
8 NET "led[3]" LOC = P15;
9 NET "led[4]" LOC = R16;
10 NET "led[5]" LOC = F13;
11 NET "led[6]" LOC = N16;
sistema.ucf
Ln 1 Col 1 UCF

```

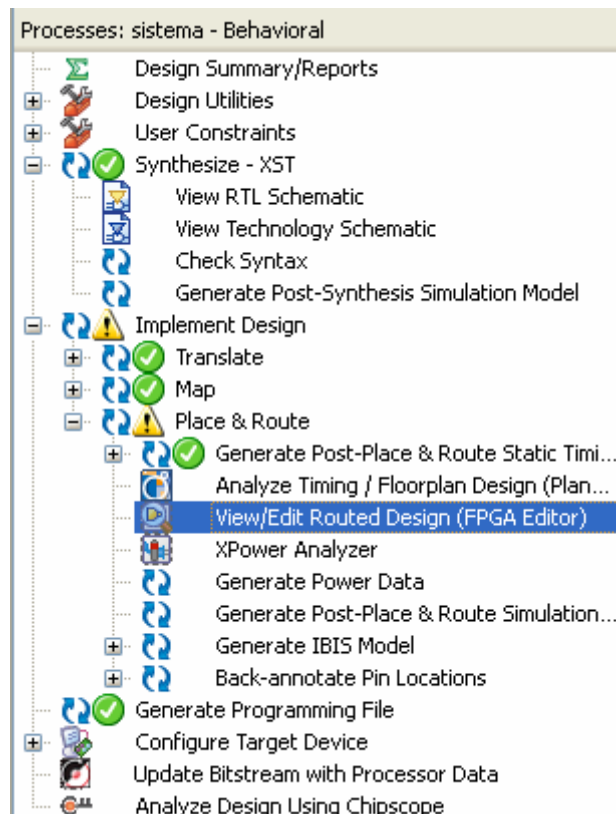


La ventana de jerarquía presenta ahora el aspecto que se muestra a la izquierda.

Ya estamos listos para realizar el resto de fases de diseño y configurar la FPGA con el fichero **sistema.bit**. Deberíamos observar en el display de la placa la cuenta cíclica que se muestra en la parte inferior.

00123456789

9.- Ejercicios



1. Ejecutar el comando *View RTL Schematic*. Observar los resultados que se obtienen.

2. Ejecutar el comando *View Technology Schematic*. Observar los resultados que se obtienen.

3. Ejecutar el *FPGA Editor*. Con este comando podemos observar cómo queda configurada la FPGA por dentro. Emplear el zoom para obtener un mayor nivel de detalle. Seleccionar una celda lógica o *slice* que esté usado y mirar su contenido haciendo doble clic en el mismo.

4. Repetir el contador con el código VHDL comportamental que se muestra. Es necesario usar las librerías que se muestran.

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
```

```
32 architecture Behavioral of contador2 is
33
34 signal int_count : STD_LOGIC_VECTOR (3 downto 0);
35
36 begin
37
38 process (clk, reset)
39 begin
40     if reset='1' then
41         int_count <= (others => '0');
42     elsif clk='1' and clk'event then
43         if ce='1' then
44             int_count <= int_count + 1;
45         end if;
46     end if;
47 end process;
48
49 count <= int_count;
50
51 end Behavioral;
```