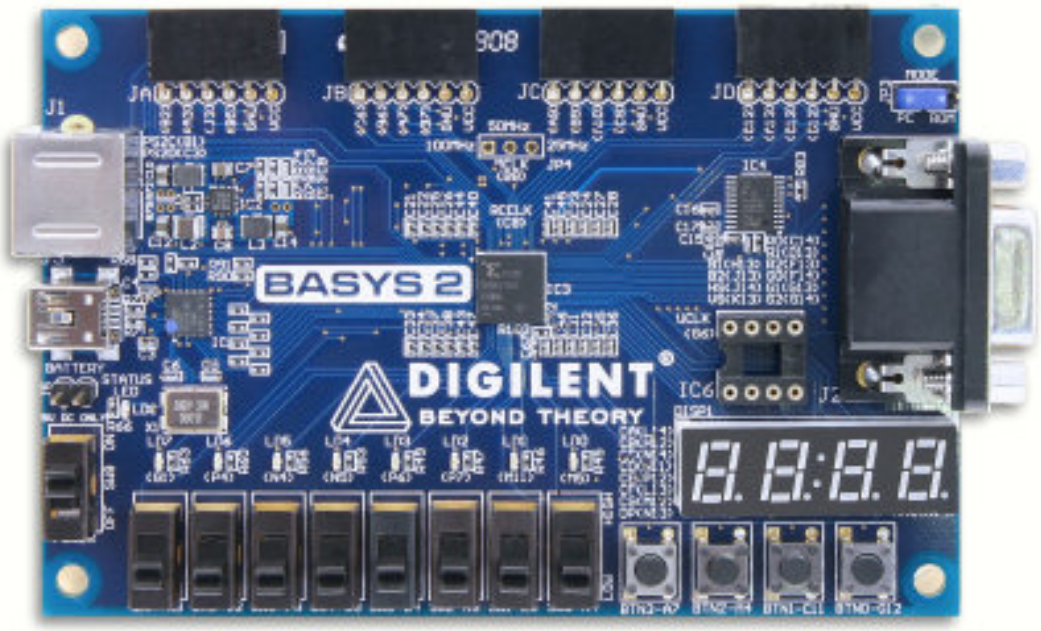
Universidad Politécnica de Madrid
ETSI de Telecomunicación
Departamento de Ingeniería Electrónica

Circuitos Electrónicos (CELT)

Manual de referencia de la tarjeta BASYS 2



Álvaro de Guzmán Fernández

Manual de referencia de la tarjeta BASYS 2

Índice general

Introducción

Esta guía pretende resumir los conceptos básicos de la utilización de la tarjeta BASYS2 y presentar unas directrices generales para el uso de la herramienta ISE, de tal forma que el alumno pueda empezar a utilizar las herramientas de desarrollo del laboratorio en un corto periodo de tiempo.

En ningún caso se pretende que este documento sea una descripción exhaustiva del lenguaje VHDL o del funcionamiento completo del ISE. Para ello remitimos al lector a los libros y manuales correspondientes que se detallan al final en la bibliografía adjunta.

Se recomienda que utilice los ejemplos que aparecen en esta guía para practicar en el manejo de las herramientas. También se recomienda sintetizar dichos ejemplos e incluso que pruebe a modificar su funcionamiento nominal. De esta manera podrá familiarizarse con el equipo de diseño del laboratorio y aumentar su eficiencia a la hora de desarrollar el prototipo que se exige en la asignatura.

La guía comienza con una descripción de la tarjeta BASYS2, de sus características y particularidades ilustradas a través de ejemplos. Por último se dan algunas ideas sobre el uso de la herramienta ISE y sobre la carga de los ficheros de configuración de la FPGA (“bit stream”).

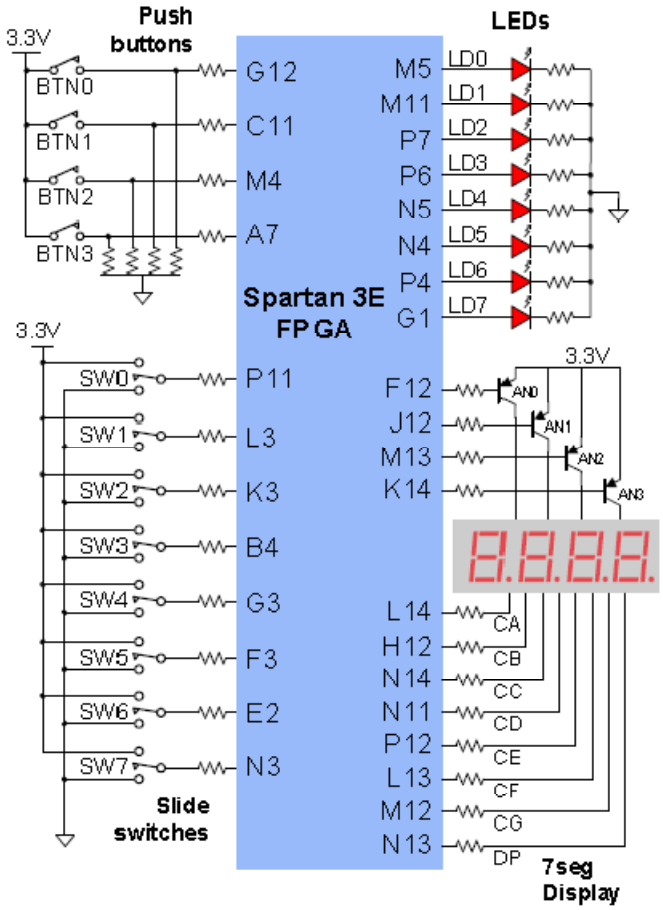
Descripción general de la tarjeta BASYS 2

La tarjeta BASYS 2 es una tarjeta de desarrollo fabricada por la compañía DIGILENT que contiene una FPGA modelo Spartan 3E de XILINX. Está diseñada para el aprendizaje de la síntesis de circuitos de complejidad media utilizando un entorno de desarrollo profesional.

Además de la citada FPGA, esta tarjeta contiene una serie de recursos que pueden ser utilizados en los diseños de los circuitos. Concretamente contiene:

- 4 pulsadores
- 8 conmutadores
- 8 LEDS
- 4 displays de 7 segmentos
- Un conector de teclado de PC
- Una salida de vídeo VGA

Todos estos recursos se encuentran conectados a las patillas de la FPGA de la forma que se indica en la siguiente figura:



Además la FPGA posee una entrada conectada a un reloj con una frecuencia de **50 MHz** que se corresponde con la patilla **M6**.

En dicha figura se han omitido intencionadamente el conector de teclado y la salida VGA por no ser necesarias para las aplicaciones que se pretenden realizar en esta asignatura.

De esta manera, el valor lógico proporcionado por el pulsador BTN0, se leerá en la patilla G12 de la FPGA. Del mismo modo, para activar el LED LD0, es necesario poner un valor lógico '1' en la patilla M5.

Además de los recursos integrados en la tarjeta, existe la posibilidad de utilizar entradas y salidas externas, las cuales también se encuentran conectadas a las patillas de la FPGA y que se encuentran disponibles en los conectores externos del entrenador presente en cada puesto del laboratorio (conectores marcados como ENTRADAS DIGITALES y SALIDAS DIGITALES). En el enunciado de cada práctica se indicarán las patillas concretas de la FPGA que se corresponden con estas entradas y salidas.

Síntesis de circuitos mediante la tarjeta BASYS 2

La síntesis de circuitos se realizará mediante el lenguaje de descripción hardware VHDL. Se compilará utilizando el entorno ISE de XILINX que se encontrará disponible en el ordenador de cada puesto del laboratorio. Este entorno es capaz de crear un archivo para la configuración de la FPGA a partir del código VHDL que se escriba (archivo de *"bit stream"* con extensión .bit). Dicho archivo debe ser cargado en la tarjeta BASYS 2. Esto hace que el hardware interno de la FPGA se configure para seguir las especificaciones hardware descritas.

Para volcar el contenido del archivo en la FPGA y configurarla es necesario utilizar el programa ADEPT de DIGILENT, el cual también estará disponible en el ordenador de cada puesto.

Por tanto, la secuencia de síntesis de un circuito deberá seguir necesariamente los siguientes pasos:

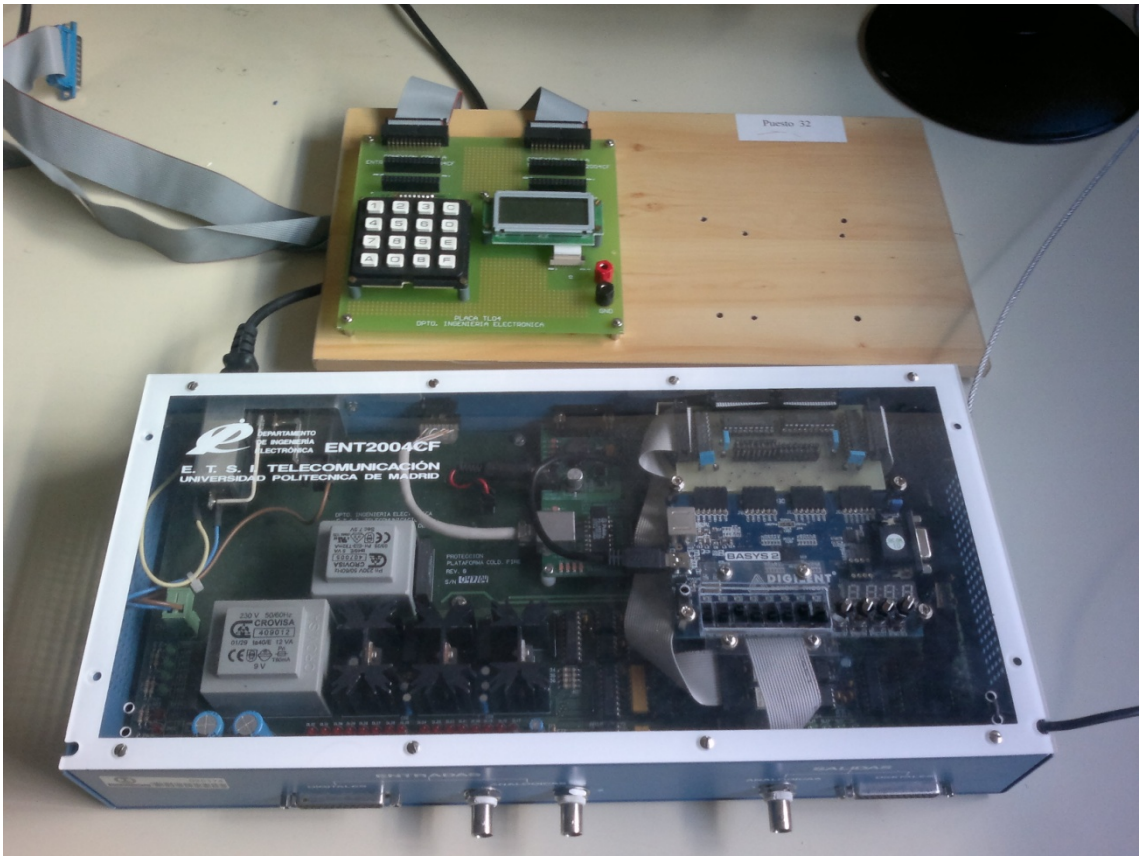
1. Escribir un código en VHDL que describa el hardware que queremos sintetizar. Recuerde que ESTAMOS SINTETIZANDO HARDWARE y que por tanto el VHDL NO ES UN LENGUAJE DE PROGRAMACIÓN, SINO UN LENGUAJE DE DESCRIPCIÓN HARDWARE
2. Simulación del circuito para estudiar su comportamiento antes de su síntesis en la FPGA. Esto le ayudará a depurar posibles errores que puedan existir en el código.
3. Compilar dicho código y generar el archivo de "bit stream". Evidentemente el archivo no se generará si el programa tiene errores.
4. Una vez generado el fichero de "bit stream", deberá utilizarse el programa ADEPT para volcarlo en la FPGA.
5. En este momento, la FPGA se convierte en un circuito que deberá realizar la tarea que haya sido descrita mediante el VHDL.

El equipo de desarrollo del laboratorio

En el laboratorio se dispone de un equipo de desarrollo que consiste en un armazón metálico con tapa transparente el cual contiene una tarjeta BASYS2 y algunos circuitos de protección para evitar que pueda ser dañada por causa de malas conexiones o cortocircuitos. Los recursos de la tarjeta pueden ser accedidos desde el exterior (microinterruptores y pulsadores).

El citado armazón está conectado a su vez a una tabla de madera que posee varios conectores, un teclado y un display. Los conectores se emplean para acceder a las entradas y salidas externas de la tarjeta BASYS 2 y serán descritos con detalle más adelante en este documento.

La fotografía siguiente muestra la disposición de los elementos que integran el equipo de desarrollo.



El equipo de desarrollo (armazón azul) posee un interruptor en la parte posterior. **Dicho interruptor deberá estar encendido para que el equipo funcione correctamente**. Asegúrese también que los cables planos se encuentran conectados a los conectores ENTRADAS y SALIDAS del armazón azul.

La estructura básica de un diseño VHDL

Un diseño VHDL consiste en la especificación de un hardware concreto que se quiere sintetizar en el interior de la FPGA. Para escribir el código VHDL se utilizará el editor del entorno ISE, y serán necesarios obligatoriamente los siguientes archivos:

1. Uno o varios archivos conteniendo el código VHDL, (dependiendo de su complejidad a veces es más sencillo separar el código en varios ficheros independientes). En estos archivos se declararán entidades ("entity") cuyas entradas y salidas tendrán nombres arbitrarios elegidos por el desarrollador.
2. Un archivo de asociaciones donde se le dice al compilador qué patillas de la FPGA se corresponden con cada entrada y salida declaradas en las entidades que componen el circuito. Este fichero es IMPRESCINDIBLE para generar el "bit stream". La sintaxis empleada en este fichero es muy sencilla y se describe en los ejemplos siguientes.

Ejemplo 1: Encendido de un LED mediante un conmutador.

El objetivo de este ejemplo es demostrar cómo se sintetiza un circuito que sea capaz de reflejar en un LED el valor lógico proporcionado por un conmutador. Se utilizarán los recursos de la tarjeta BASYS 2.

Se realizará con dos ficheros: uno de código VHDL y otro de asociaciones.

FICHERO PRUEBA_LED.VHD (fichero de código VHDL)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity circuito is
  port (
    CONMUT : in   STD_LOGIC;    -- Conmutador conectado a una entrada
    LED    : out   STD_LOGIC    -- LED conectado a una salida
        );
end circuito;

architecture a_circuito of circuito is

begin
LED<=CONMUT;
end a_circuito;
```

FICHERO DE ASOCIACIONES: ASOCIACIONES.UCF

```
NET "CONMUT" LOC = "P11"; # Conmutador 0
NET "LED" LOC = "M5";     # LED0
```

Con estos dos ficheros se puede ya compilar el código y obtener un "bit stream". En el fichero VHDL declaramos una entidad llamada "circuito" con una entrada llamada CONMUT y una salida llamada LED. En su descripción funcional, decimos que dicha salida se corresponde con la entrada (de esta manera el valor del LED variará según el valor digital del conmutador). Es por tanto un circuito combinacional. En el fichero de asociaciones indicamos cuáles son las patillas de la FPGA que se asocian físicamente con la entrada y la salida de la entidad. Fíjese

que se asocian exactamente con las patillas donde se encuentran conectados los recursos de la tarjeta BASYS 2.

Ejemplo 2: Utilización del reloj de la FPGA.

La FPGA posee una entrada conectada a un reloj de 50 MHz en la patilla M6. Este reloj puede ser utilizado para el diseño de circuitos secuenciales síncronos. En este ejemplo utilizaremos el reloj para dividir su frecuencia mediante un contador visualizando la salida en un LED.

Utilizaremos un proceso en cuya lista de sensibilidad se coloca la señal de reloj. Con cada flanco activo se incrementará un contador. Queremos que el LED parpadee con un periodo encendido/apagado de 500 ms cada uno.

Se utilizarán igualmente dos ficheros, uno VHDL y otro de asociaciones.

FICHERO DIV_RELOJ.VHD (fichero de código VHDL)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity circuito is
    Port ( CLK : in  STD_LOGIC;    -- Reloj de entrada a la FPGA
           SAL : out  STD_LOGIC);  -- salida del circuito para conectar al LED
end circuito;

architecture a_circuito of circuito is
signal contador : STD_LOGIC_VECTOR (31 downto 0);
signal flag : STD_LOGIC;
begin
  PROC_CONT : process (CLK)
    begin
    if CLK'event and CLK='1' then
      contador<=contador + '1';
      if contador=25000000 then   -- el reloj tiene un periodo de 20 ns (50 MHz)
        flag<=not flag;           -- tras 25e6 cuentas habrán transcurrido 500 ms
        contador<=(others=>'0');
      end if;
    end if;
    end process;

  SAL<=flag;
end a_circuito;
```

FICHERO DE ASOCIACIONES: ASOCIACIONES.UCF

```
NET "CLK" LOC = "M6"; # Reloj de la FPGA
NET "SAL" LOC = "M5"; # LED0
```

En este caso se trata de un circuito secuencial con un proceso que sincroniza el sistema.

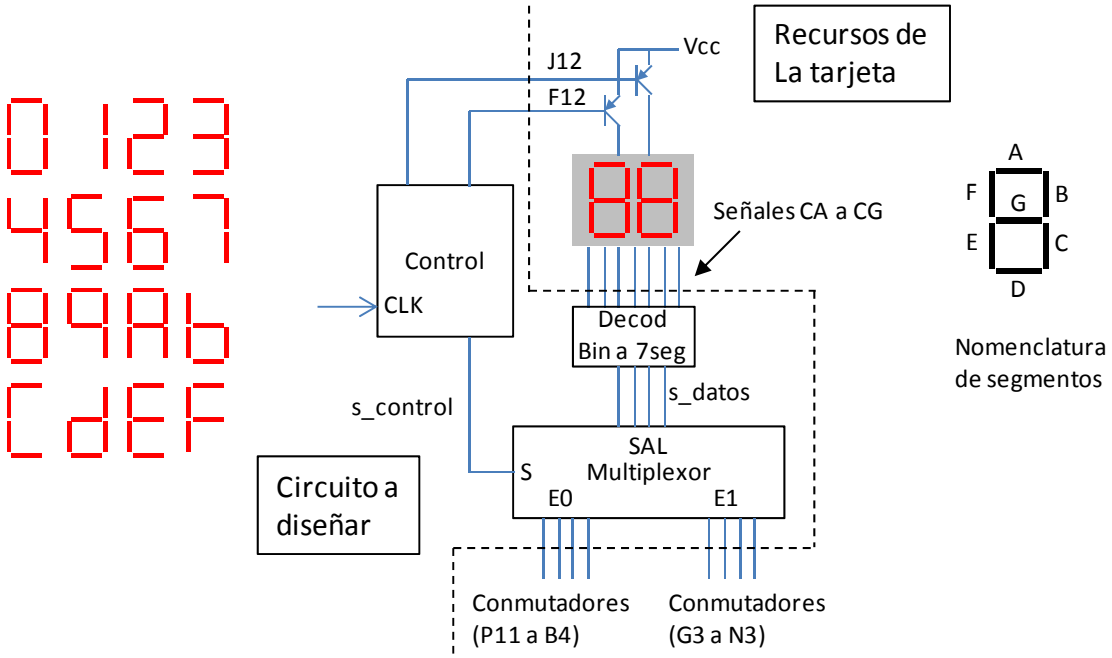
Esta es la estructura básica cuando se trata de un circuito de muy poca complejidad como los presentados en los ejemplos 1 y 2. A continuación se muestra un ejemplo más complejo con varios ficheros VHDL.

Ejemplo 3: Visualización de dos cifras diferentes en dos displays. Las cifras hexadecimales se introducen mediante los conmutadores.

En este caso queremos visualizar dos cifras diferentes en dos de los displays disponibles en la tarjeta BASYS 2. El problema de este ejemplo reside en el hecho de que los 7 valores binarios para excitar los distintos segmentos son compartidos por los 4 displays simultáneamente. Existen, no obstante, 4 señales (las correspondientes a las salidas F12, J12, M13 y K14) que controlan la activación independiente de cada uno de los displays. Para visualizar cifras diferentes en cada uno, es necesario activar los segmentos correspondientes a la cifra de uno de los displays junto con su señal de activación, a continuación hacer lo mismo con el siguiente y así sucesivamente. Si esta secuencia se repite más de 25 veces por segundo, el ojo no es capaz de percibir el parpadeo, pudiendo representarse varias cifras diferentes.

Para este ejemplo necesitaremos los siguientes elementos (ver figura):

- Un decodificador de binario a 7 segmentos.
- Un multiplexor de dos entradas de 4 bits para seleccionar la cifra a visualizar
- Un elemento de control que realice el refresco periódico de los displays.



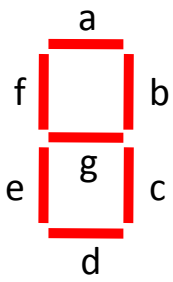
Visualizaremos todos los valores entre 0000 y 1111 utilizando para ello cifras hexadecimales tal como se muestra en la figura. Utilizaremos los recursos de la tarjeta: 8 conmutadores (4 para la primera cifra y 4 para la segunda), dos displays y el reloj para controlar la visualización.

En este caso, la manera más apropiada de describir este hardware consiste en realizar un fichero independiente por cada elemento, con descripciones funcionales de dichos elementos y un fichero maestro con una descripción estructural de interconexión entre ellos. Además también es necesario escribir el fichero de asociaciones.

El ejemplo constaría por tanto de 4 ficheros VHDL más uno de asociaciones.

FICHERO decod7s.vhd

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity decod7s is
    Port ( D : in  STD_LOGIC_VECTOR (3 downto 0); -- entrada de datos en binario
           S : out  STD_LOGIC_VECTOR (0 to 6));   -- salidas para los segmentos
end decod7s;

architecture a_decod7s of decod7s is

begin

with D select S <=
"0000001" when "0000",
"1001111" when "0001",
"0010010" when "0010",
"0000110" when "0011",
"1001100" when "0100",
"0100100" when "0101",
"0100000" when "0110",
"0001111" when "0111",
"0000000" when "1000",
"0001100" when "1001",
"0001000" when "1010",
"1100000" when "1011",
"0110001" when "1100",
"1000010" when "1101",
"0110000" when "1110",
"0111000" when "1111",
"1111111" when others;

end a_decod7s;
```



Este primer fichero contiene la descripción funcional de un decodificador de binario a 7 segmentos. Debe tenerse en cuenta que según las conexiones de la figura los segmentos se encienden con un “0” en la patilla correspondiente.

FICHERO MUX.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX is
    Port ( E0 : in  STD_LOGIC_VECTOR (3 downto 0);  -- Entrada de datos 0
           E1 : in  STD_LOGIC_VECTOR (3 downto 0);  -- Entrada de datos 1
           S : in  STD_LOGIC;                       -- Entrada de control
           SAL : out  STD_LOGIC_VECTOR (3 downto 0)); -- Salida
end MUX;

architecture a_MUX of MUX is

begin

with S select SAL<=
E0 when '0',
E1 when '1',
E0 when others;

end a_MUX;
```

Este fichero describe un multiplexor de 2 entradas de 4 bits y una salida. La señal de control es S.

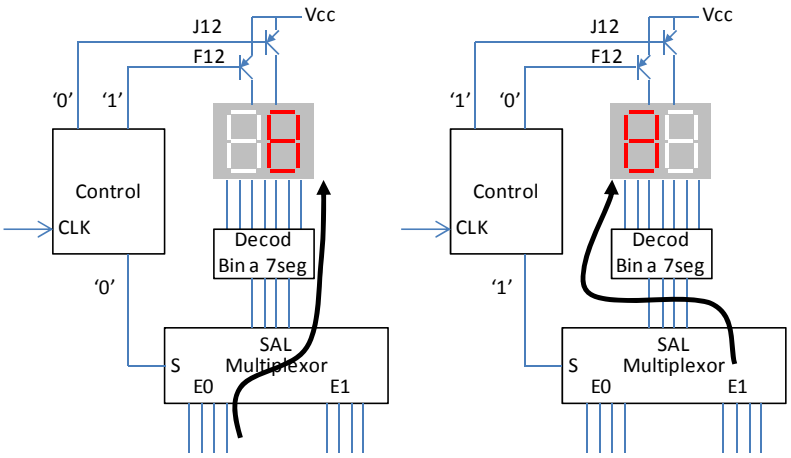
FICHERO control.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control is
    Port ( CLK : in  STD_LOGIC;          -- reloj de la FPGA
           DSP0 : out  STD_LOGIC;        -- señal para activar el display 0
           DSP1 : out  STD_LOGIC;        -- señal para activar el display 1
           SCONTROL : out  STD_LOGIC);   -- señal para controlar el mux.
end control;

architecture a_control of control is
signal contador : STD_LOGIC_VECTOR (31 downto 0);
signal salida : STD_LOGIC;
begin

process (CLK)
begin
if CLK'event and CLK='1' then    -- el periodo del reloj es de 20 ns
  contador<=contador + '1';      -- por tanto 500.000 cuentas se corresponden
  if contador=500000 then        -- con el transcurso de 10 ms.
    salida<=not salida;
        contador<=(others=>'0');
  end if;
end if;
end process;

SCONTROL<=salida;
DSP0<=salida;
DSP1<=not salida;

end a_control;
```

Este fichero describe un circuito secuencial de control. Un proceso divide la frecuencia del reloj utilizando un contador. Cada 10 ms invierte el valor lógico de la señal ‘salida’.

En paralelo con este proceso la salida SCONTROL toma el valor de la señal ‘salida’, mientras que las salidas DSP0 y DSP1 toman valores lógicos opuestos dependiendo de la señal ‘salida’.

Con este código se describe un elemento que alterna el valor de SCONTROL cada 10 ms. Al mismo tiempo invierte los valores de DSP0 y DSP1 que son las señales que activarán los displays. De esta manera uno de ellos está activo cuando se selecciona una entrada del multiplexor. El otro se activa cuando se selecciona la otra entrada. (Tenga en cuenta que según la figura, los displays se activan cuando la señal de control DSP correspondiente es ‘0’).



El circuito de control alterna
estas dos configuraciones cada 10 ms

FICHERO circuito.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity circuito is
    Port ( CIFRA0 : in  STD_LOGIC_VECTOR (3 downto 0); -- entrada de cifra 0
           CIFRA1 : in  STD_LOGIC_VECTOR (3 downto 0); -- entrada de cifra 1
           SEG7 : out  STD_LOGIC_VECTOR (0 to 6);  -- salidas para los displays
           DISP1 : out  STD_LOGIC;                 -- salida para activar display 1
           DISP2 : out  STD_LOGIC;                 -- salida para activar display 2
           CLK : in  STD_LOGIC);                   -- reloj de la FPGA
end circuito;

architecture a_circuito of circuito is             -- en este caso se trata de una

component decod7s                                  -- descripción estructural
  Port ( D : in  STD_LOGIC_VECTOR (3 downto 0);
         S : out  STD_LOGIC_VECTOR (0 to 6));
end component;

component MUX
  Port ( E0 : in  STD_LOGIC_VECTOR (3 downto 0);
         E1 : in  STD_LOGIC_VECTOR (3 downto 0);
         S : in  STD_LOGIC;
         SAL : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component control
  Port ( CLK : in  STD_LOGIC;
         DSP0 : out  STD_LOGIC;
         DSP1 : out  STD_LOGIC;
         SCONTROL : out  STD_LOGIC);
end component;

signal s_control : STD_LOGIC;
signal s_datos   : STD_LOGIC_VECTOR (3 downto 0);

begin

U1 : control
  port map (
         CLK => CLK,
         DSP0 => DISP1,
         DSP1 => DISP2,
         SCONTROL => s_control
         );

U2 : MUX
  port map (
         E0 => CIFRA0,
         E1 => CIFRA1,
         S => s_control,
         SAL => s_datos
         );

U3 : decod7s
  port map (
         D => s_datos,
         S => SEG7
         );

end a_circuito;
```

Este es el fichero maestro que describe la interconexión entre los distintos elementos. Se describe un circuito principal cuyas entradas son las dos cifras binarias de 4 bits y la señal de reloj. Sus salidas son las señales de activación de los displays, y los valores de los segmentos.

Cuando una entrada o salida de un componente se cablea directamente a una entrada o salida del circuito principal, se asigna directamente en el “port map”. Si se trata de interconexiones

entre elementos entonces deben utilizarse señales adicionales declaradas también en el fichero.

FICHERO DE ASOCIACIONES: asociaciones.ucf

```
NET "CLK" LOC = "M6"; # RELOJ DE LA FPGA

NET "SEG7<0>" LOC = "L14"; #Segmento a
NET "SEG7<1>" LOC = "H12"; #Segmento b
NET "SEG7<2>" LOC = "N14"; #Segmento c
NET "SEG7<3>" LOC = "N11"; #Segmento d
NET "SEG7<4>" LOC = "P12"; #Segmento e
NET "SEG7<5>" LOC = "L13"; #Segmento f
NET "SEG7<6>" LOC = "M12"; #Segmento g

NET "CIFRA0<0>" LOC = "P11"; # CONMUTADOR 0
NET "CIFRA0<1>" LOC = "L3";
NET "CIFRA0<2>" LOC = "K3";
NET "CIFRA0<3>" LOC = "B4";  # CONMUTADOR 3

NET "CIFRA1<0>" LOC = "G3"; # CONMUTADOR 4
NET "CIFRA1<1>" LOC = "F3";
NET "CIFRA1<2>" LOC = "E2";
NET "CIFRA1<3>" LOC = "N3";  # CONMUTADOR 7

NET "DISP1" LOC = "F12"; # ACTIVACIÓN DISP 1
NET "DISP2" LOC = "J12"; # ACTIVACIÓN DISP 2
```

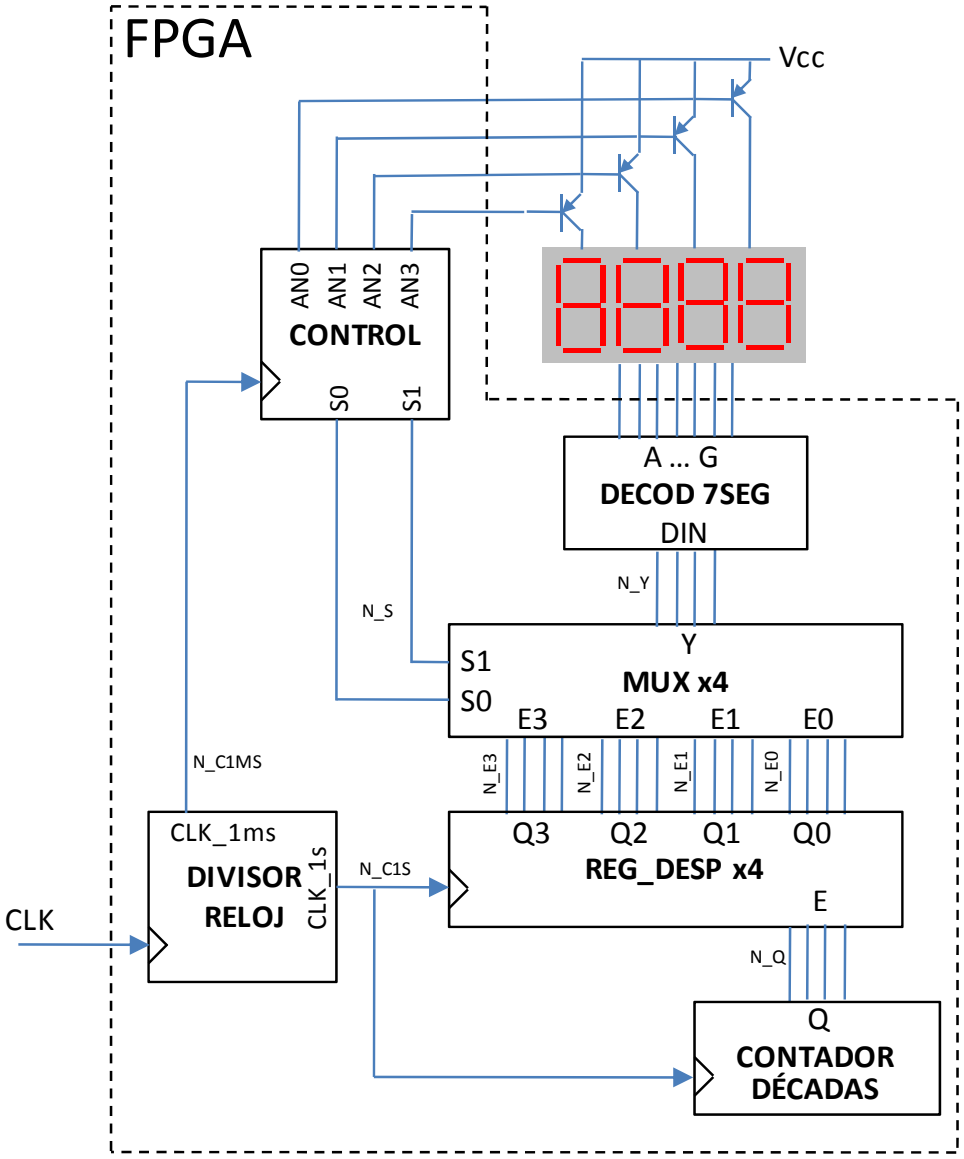
Por último, las entradas y salidas del circuito principal deben asociarse con los recursos de la tarjeta. Esto se hace mediante el correspondiente fichero de asociaciones que se muestra. La nomenclatura SEG7<N> se corresponde con el bit N de la salida SEG7. Lo mismo sucede con CIFRA0 y CIFRA1.

Con este circuito se representan las cifras hexadecimales correspondientes a los conmutadores en los dos displays de la derecha. Sin embargo los otros dos presentan también cifras como consecuencia de no estar desactivados. Un posible ejercicio consiste en modificar el archivo de descripción estructural (cricuito.vhd) y el fichero de asociaciones para que se mantengan desactivados los dos displays no utilizados.

Ejemplo 4: Contador rotatorio

Se muestra a continuación el desarrollo de un contador rotatorio. Se trata de un contador que utiliza los 4 displays y que genera la siguiente secuencia en los mismos: 0000, 0001, 0012, 0123, 1234, 2345, 3456, 4567, 5678, 6789, 7890, 8901, 9012, 0123, … repitiéndose la secuencia constantemente. Los cambios se producen cada segundo.

El desarrollo de este contador supone el empleo de varios módulos digitales: un contador de décadas, un registro de desplazamiento de 4 bits, un multiplexor paralelo con 4 entradas de 4 bits, un decodificador BCD a 7 segmentos y algunos elementos adicionales que se muestran en la figura siguiente:



Todo lo que se encuentra dibujado dentro de la línea punteada son los circuitos que debemos implementar en la FPGA. Lo que se encuentra fuera de dicha línea son los recursos de la tarjeta que emplearemos en el fichero de asociaciones.

El divisor del reloj genera dos relojes adicionales, uno con periodo de 1 ms y otro con periodo de 1 s. El primero se emplea para refrescar los displays periódicamente, puesto que como ya se ha mencionado todos ellos comparten las líneas A..G y por lo tanto es necesario presentar las cifras una a una de forma muy rápida. El segundo se utiliza para incrementar el contador y desplazar las cifras hacia la izquierda.

El refresco de los displays se realiza de una forma similar a la descrita en el ejemplo 3, aunque con 4 entradas en lugar de 2.

El diseño completo de este contador rotatorio está compuesto por varios módulos, los cuales se describen con detalle a continuación:

FICHERO contador.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity contador is
    Port ( CLK : in  STD_LOGIC;                          -- entrada de reloj
           Q   : out  STD_LOGIC_VECTOR (3 downto 0)); -- salida de datos
end contador;

architecture a_contador of contador is

signal QS : STD_LOGIC_VECTOR (3 downto 0); -- señal que almacena el valor del contador

begin
  SYNC : process (CLK)
    begin
      if (CLK'event and CLK='1') then
        QS<=QS+1;                        -- con cada flanco activo se incrementa
        if QS=9 then                     -- si llega a 9 se pone a 0
          QS<="0000";
        end if;
      end if;
    end process;

  Q<=QS;                     -- actualización de la salida
end a_contador;
```

Este fichero define un contador de décadas. Con cada flanco de reloj el contador se incrementa y cuando esta cuenta llega a 9, el contador vuelve a 0 en el siguiente ciclo de reloj.

FICHERO reg_desp4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg_desp4 is
    Port ( CLK : in   STD_LOGIC;                          -- entrada de reloj
           E   : in   STD_LOGIC_VECTOR (3 downto 0);   -- entrada de datos
           Q0  : out  STD_LOGIC_VECTOR (3 downto 0);   -- salida Q0
           Q1  : out  STD_LOGIC_VECTOR (3 downto 0);   -- salida Q1
           Q2  : out  STD_LOGIC_VECTOR (3 downto 0);   -- salida Q2
           Q3  : out  STD_LOGIC_VECTOR (3 downto 0));  -- salida Q3
end reg_desp4;

architecture a_reg_desp4 of reg_desp4 is

signal QS0 : STD_LOGIC_VECTOR (3 downto 0); -- señal que almacena el valor de Q0
signal QS1 : STD_LOGIC_VECTOR (3 downto 0); -- señal que almacena el valor de Q1
signal QS2 : STD_LOGIC_VECTOR (3 downto 0); -- señal que almacena el valor de Q2
signal QS3 : STD_LOGIC_VECTOR (3 downto 0); -- señal que almacena el valor de Q3

begin
  process (CLK)
    begin
      if (CLK'event and CLK='1') then   -- con cada flanco activo
        QS3<=QS2;                        -- se desplazan todas las salidas
        QS2<=QS1;
        QS1<=QS0;
        QS0<=E;                         -- y se copia el valor de la entrada en Q0
      end if;
  end process;

  Q0<=QS0;                              -- actualización de las salidas
  Q1<=QS1;
  Q2<=QS2;
  Q3<=QS3;

end a_reg_desp4;
```

Este fichero contiene la descripción de un registro de desplazamiento que desplaza palabras de 4 bits. Con cada flanco de reloj, los datos de la entrada E son copiados en la salida Q0, mientras el resto de las salidas son desplazadas hacia la izquierda.

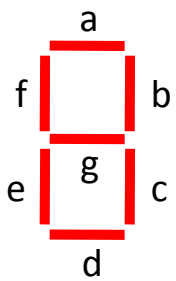
FICHERO MUX4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux4 is
    Port ( E0 : in   STD_LOGIC_VECTOR (3 downto 0);  -- entrada E0
           E1 : in   STD_LOGIC_VECTOR (3 downto 0);  -- entrada E1
           E2 : in   STD_LOGIC_VECTOR (3 downto 0);  -- entrada E2
           E3 : in   STD_LOGIC_VECTOR (3 downto 0);  -- entrada E3
           Y  : out  STD_LOGIC_VECTOR (3 downto 0);  -- salida Y
           S  : in   STD_LOGIC_VECTOR (1 downto 0)); -- entradas de control

end mux4;

architecture a_mux4 of mux4 is
begin

Y <= E0 when S="00" else  -- se selecciona la salida en función de las entradas
     E1 when S="01" else  -- de control
     E2 when S="10" else
     E3 when S="11";

end a_mux4;
```

Como puede verse, en este caso se trata de un multiplexor de 4 entradas de 4 bits (E0..E3) cuyas entradas de control son S0 y S1 y la salida es Y.

FICHERO decod7s.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decod7s is
  port ( DIN   : in  STD_LOGIC_VECTOR (3 downto 0);   -- entrada de datos
         S7SEG : out STD_LOGIC_VECTOR (0 to 6));   -- salidas 7seg (abcdefg)
end decod7s;

architecture a_decod7s of decod7s is
begin
   with DIN select S7SEG <=
      "0000001" when "0000",
      "1001111" when "0001",
      "0010010" when "0010",
      "0000110" when "0011",
      "1001100" when "0100",
      "0100100" when "0101",
      "0100000" when "0110",
      "0001111" when "0111",
      "0000000" when "1000",
      "0001100" when "1001",
      "1111111" when others;
end a_decod7s;
```



Este fichero define un decodificador de binario a 7 segmentos donde las entradas pueden tomar los valores decimales 0 a 9. En otro caso se presenta un valor que apaga todos los segmentos.

FICHERO control.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control is
    Port ( CLK : in  STD_LOGIC;                     -- entrada de reloj
           AN : out  STD_LOGIC_VECTOR (3 downto 0); -- activación displays
           S : out  STD_LOGIC_VECTOR (1 downto 0)); -- selección en el MUX
end control;

architecture a_control of control is

signal SS  : STD_LOGIC_VECTOR (1 downto 0);

begin

process (CLK)
  begin
  if (CLK'event and CLK='1') then
    SS<=SS+1;                      -- genera la secuencia 00,01,10 y 11
  end if;
  end process;

S<=SS;
AN<="0111" when SS="00" else   -- activa cada display en function del valor de SS
    "1011" when SS="01" else
    "1101" when SS="10" else
    "1110" when SS="11";

end a_control;
```

Este módulo secuencial utiliza una señal de reloj para incrementar un contador de 2 bits que repite la secuencia 00, 01, 10 y 11 indefinidamente. Con esta secuencia se genera una señal AN que activa cada uno de los displays (mediante un ‘0’). Esto se empleará para realizar el refresco de los displays. La secuencia de dos bits conmutará las entradas del multiplexor y las señales AN activarán los displays en consonancia, de un modo similar al del ejemplo 3 pero con 4 displays.

FICHERO div_reloj.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity div_reloj is
    Port ( CLK : in  STD_LOGIC;
           CLK_1S : out  STD_LOGIC;
           CLK_1MS : out  STD_LOGIC);
end div_reloj;

architecture a_div_reloj of div_reloj is

signal div1   : STD_LOGIC_VECTOR (31 downto 0); -- contador para dividir el reloj
signal div2   : STD_LOGIC_VECTOR (31 downto 0); -- contador para dividir el reloj
signal S_1S   : STD_LOGIC;                      -- señal de periodo 1s
signal S_1MS : STD_LOGIC;                       -- señal de period 1ms
```

```
begin
  DIV : process (CLK)
    begin
          if (CLK'event and CLK='1') then
            div1<=div1+1;            -- en cada flanco de CLK se incrementa div1
            div2<=div2+1;           -- en cada flanco de CLK se incrementa div2


            if div1=25000000 then   -- cuando llega a este valor han transcurrido 500ms
              S_1S<=not S_1S;       -- se conmuta el valor de S_1S
              div1<=(others=>'0');
            end if;

            if div2=25000 then      -- div2=2500 cuando han transcurrido 500 us
              S_1MS<=not S_1MS;     -- se conmuta el valor de S_1MS
              div2<=(others=>'0');
            end if;

          end if;
    end process;


  CLK_1S<=S_1S;
  CLK_1MS<=S_1MS;

end a_div_reloj;
```

Este es otro módulo secuencial en el que se utiliza el reloj del sistema de 50 MHz para generar los relojes de menor frecuencia que deben utilizarse en este circuito. Mediante dos contadores (div1 y div2) se realiza la división. Cuando uno de ellos cuenta 25.000.000 ciclos, habrán transcurrido 500 ms. En ese momento se conmuta el valor de la señal S_1S, por lo que la frecuencia que aparecerá en esta señal será de 1 Hz. En el otro caso, se conmuta el valor de S_1MS cada 25000 cuentas, por lo que la frecuencia de esta señal será de 1 KHz.

FICHERO circuito.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity circuito is
Port ( CLK  : in  STD_LOGIC;
       SEG7 : out  STD_LOGIC_VECTOR (0 to 6);
       AN   : out  STD_LOGIC_VECTOR (3 downto 0));
end circuito;

architecture a_circuito of circuito is

signal N_E0 : STD_LOGIC_VECTOR (3 downto 0);
signal N_E1 : STD_LOGIC_VECTOR (3 downto 0);
signal N_E2 : STD_LOGIC_VECTOR (3 downto 0);
signal N_E3 : STD_LOGIC_VECTOR (3 downto 0);
signal N_Y  : STD_LOGIC_VECTOR (3 downto 0);
signal N_Q  : STD_LOGIC_VECTOR (3 downto 0);
signal N_S  : STD_LOGIC_VECTOR (1 downto 0);
signal N_C1S   : STD_LOGIC;
signal N_C1MS : STD_LOGIC;

component contador
Port ( CLK : in  STD_LOGIC;
       Q   : out  STD_LOGIC_VECTOR (3 downto 0));
end component;
```

```
component reg_desp4
Port ( CLK : in STD_LOGIC;
               E   : in  STD_LOGIC_VECTOR (3 downto 0);
       Q0  : out  STD_LOGIC_VECTOR (3 downto 0);
       Q1  : out  STD_LOGIC_VECTOR (3 downto 0);
       Q2  : out  STD_LOGIC_VECTOR (3 downto 0);
       Q3  : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component mux4
Port ( E0  : in  STD_LOGIC_VECTOR (3 downto 0);
       E1  : in  STD_LOGIC_VECTOR (3 downto 0);
       E2  : in  STD_LOGIC_VECTOR (3 downto 0);
       E3  : in  STD_LOGIC_VECTOR (3 downto 0);
       Y   : out  STD_LOGIC_VECTOR (3 downto 0);
       S   : in  STD_LOGIC_VECTOR (1 downto 0));
end component;

component decod7s
port ( DIN   : in  STD_LOGIC_VECTOR (3 downto 0);
       S7SEG : out STD_LOGIC_VECTOR (0 to 6));
end component;

component control
Port ( CLK : in  STD_LOGIC;
       AN  : out  STD_LOGIC_VECTOR (3 downto 0);
       S   : out  STD_LOGIC_VECTOR (1 downto 0));
end component;

component div_reloj
Port ( CLK      : in  STD_LOGIC;
       CLK_1S   : out  STD_LOGIC;
       CLK_1MS  : out  STD_LOGIC);
end component;

begin

U1 : decod7s
      port map (
               DIN=>N_Y,
               S7SEG=>SEG7
               );

U2 : mux4
      port map (
              E0=>N_E0,
              E1=>N_E1,
              E2=>N_E2,
              E3=>N_E3,
              Y=>N_Y,
              S=>N_S
              );

U3 : reg_desp4
      port map (
              CLK=>N_C1S,
              E=>N_Q,
              Q0=>N_E0,
              Q1=>N_E1,
              Q2=>N_E2,
              Q3=>N_E3
              );

U4 : contador
       port map (
               CLK=>N_C1S,
               Q=>N_Q
               );

U5 : control
       port map (
               CLK=>N_C1MS,
               AN=>AN,
               S=>N_S
               );
```

```
U6 : div_reloj
       port map (
                CLK=>CLK,
                CLK_1S=>N_C1S,
                CLK_1MS=>N_C1MS
                );

end a_circuito;
```

Por último, en este código VHDL se realiza la descripción estructural del circuito completo, detallando la interconexión entre los distintos componentes. Las señales declaradas al principio de la arquitectura se corresponden con los nodos internos del circuito que no están directamente conectados a las entradas o salidas del mismo.

FICHERO DE ASOCIACIONES asociaciones.ucf

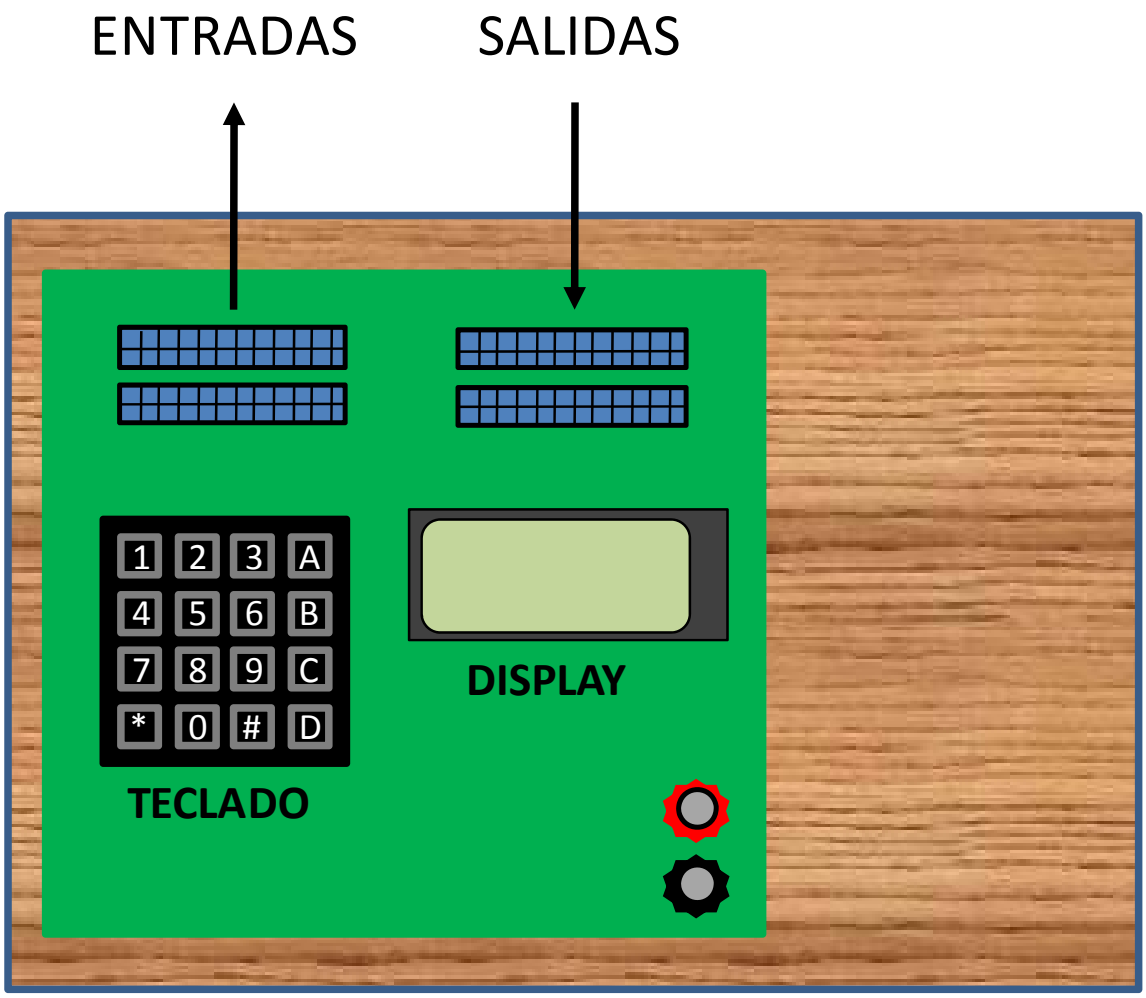
```
# Señal de reloj del sistema

NET "CLK" LOC = "M6"; # Reloj del sistema de 50 MHz

# DISPLAY DE 7 SEGMENTOS

NET "SEG7<0>" LOC = "L14"; # SEÑAL = CA
NET "SEG7<1>" LOC = "H12"; # SEÑAL = CB
NET "SEG7<2>" LOC = "N14"; # SEÑAL = CC
NET "SEG7<3>" LOC = "N11"; # SEÑAL = CD
NET "SEG7<4>" LOC = "P12"; # SEÑAL = CE
NET "SEG7<5>" LOC = "L13"; # SEÑAL = CF
NET "SEG7<6>" LOC = "M12"; # SEÑAL = CG

# SEÑALES DE ACTIVACIÓN DE LOS DISPLAYS

NET "AN<0>" LOC = "K14"; # SEÑAL = AN0
NET "AN<1>" LOC = "M13"; # SEÑAL = AN1
NET "AN<2>" LOC = "J12"; # SEÑAL = AN2
NET "AN<3>" LOC = "F12"; # SEÑAL = AN3
```

Este es el fichero de asociaciones correspondiente al circuito anterior donde se asignan las entradas y salidas a los diferentes recursos de la tarjeta. Solamente nos hacen falta en este caso el reloj del sistema y los displays.

Entradas y salidas externas

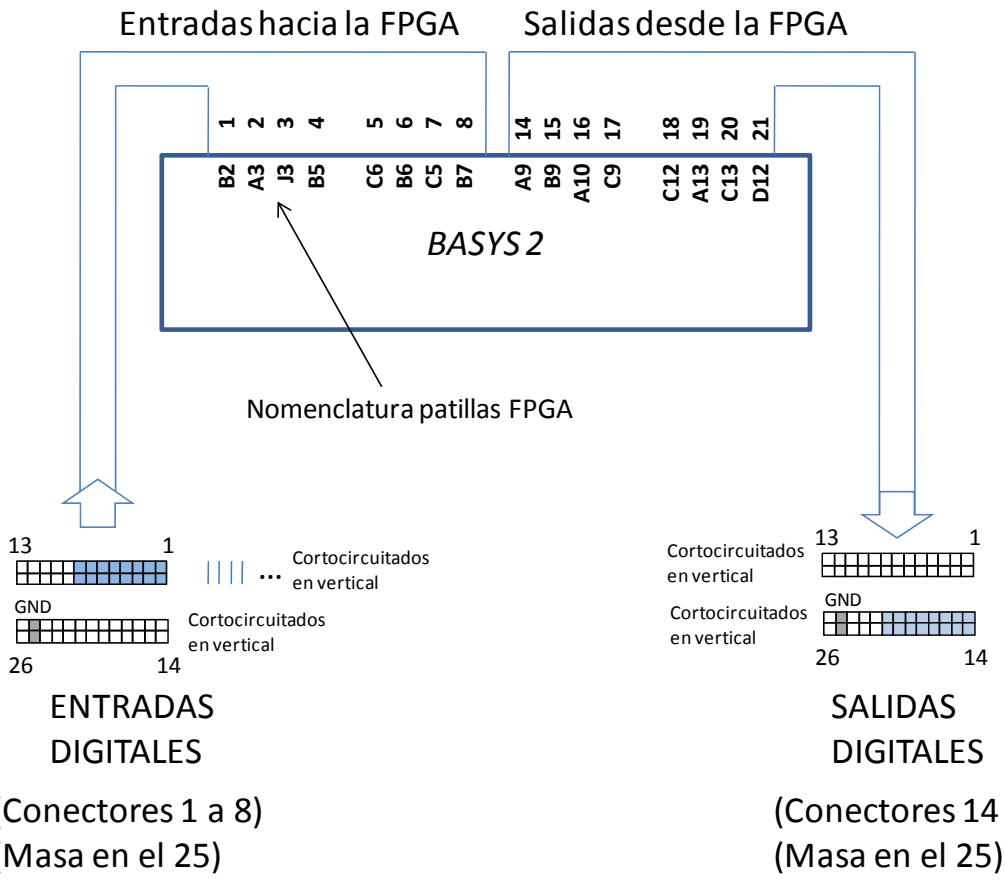
Además de los recursos conectados a algunas de las salidas de la FPGA, en la tarjeta BASYS2 existen también entradas y salidas externas que pueden utilizarse para conectar la tarjeta con circuitos exteriores.

Estas entradas y salidas se encuentran disponibles en el tablero de madera conectado a la caja azul que contiene la tarjeta. La figura siguiente muestra el esquema del citado tablero. Está formado por 4 conectores de 13 patillas más un teclado y un display. EL TECLADO Y EL DISPLAY NO SERÁN UTILIZADOS, POR LO TANTO NO TENGA EN CUENTA SU PRESENCIA.



Las entradas se refieren a señales del exterior que entran hacia la FPGA. Las salidas son señales que proporciona la FPGA hacia el exterior. Se dispone de 8 ENTRADAS y 8 SALIDAS que se encuentran conectadas a las patillas de la FPGA. Por lo tanto, para emplear estas entradas y salidas, será necesario incluir su definición en el fichero de asociaciones.

Concretamente, el cableado de estas entradas y salidas, junto con las asociaciones a las patillas de la FPGA se encuentran detallados en la figura de la página siguiente.



Como puede verse en la figura, las entradas están disponibles en el conector superior izquierdo, siendo las patillas 1 a 8 las líneas de salida y la masa está situada en el terminal 25 del conector inferior.

Las salidas se encuentran disponibles en el conector inferior derecho en las patillas 14 a 21 con la masa en el terminal 25 del citado conector.

La correspondencia de las líneas de salida y entrada con las patillas de la FPGA es la siguiente:

Entrada (terminal conector)	Patilla FPGA
1	B2
2	A3
3	J3
4	B5
5	C6
6	B6
7	C5
8	B7

Salida (terminal conector)	Patilla FPGA
14	A9
15	B9
16	A10
17	C9
18	C12
19	A13
20	C13
21	D12

ATENCIÓN: EN LAS ENTRADAS DEBERÁN SIEMPRE INTRODUCIRSE TENSIONES DE 0V Y 5V PARA EL "0" Y EL "1" RESPECTIVAMENTE. IGUALMENTE, LAS SALIDAS PROPORCIONAN TENSIONES ENTRE 0 Y 5 V.

Cada conector individual está formado por 13 columnas de 2 terminales. Los terminales verticales están cortocircuitados internamente para facilitar la interconexión con circuitos exteriores.

Una posible asignación en el fichero de asociaciones podría ser la siguiente:

```
# ENTRADAS
# --------------
NET “E1” LOC = “B2”;
NET “E2” LOC = “A3”; …
…

# SALIDAS
# --------------
NET “S9” LOC = “A9”;
NET “S10” LOC = “B9”; …
…
```

Donde E1, E2, S9 y S10 serían nodos del circuito definido en VHDL.

Ejemplo 5: Observación en el osciloscopio de señales rápidas del circuito VHDL.

A veces puede ser necesario observar una señal interna del circuito VHDL. Para ello es necesario volcar dicha señal hacia uno de los terminales de salida exterior y medirla con el osciloscopio.

En el siguiente ejemplo se define un contador binario de 4 bits con salida RCO (“*Ripple Carry Out*”) que indica el final de cuenta (se activa cuando el contador alcanza el valor binario “1111”). La señal de reloj que gobierna el contador es de 10 KHz y se genera a partir del reloj principal del sistema (50 MHz) por división de frecuencia.

Dado que la frecuencia del reloj es alta, en este caso no podemos volcar las salidas del contador ni la señal RCO hacia ninguno de los recursos de la tarjeta (LEDS o DISPLAYS), ya que la alternancia entre 0 y 1 haría que se viesen constantemente encendidos con una luminosidad intermedia.

Si queremos medir con precisión las frecuencias del reloj, las señales de cada una de las salidas del contador y la señal RCO, es necesario volcarlas hacia las salidas externas y conectar el osciloscopio en los terminales correspondientes para visualizarlas.

FICHERO contador.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador is
    Port ( CLK : in   STD_LOGIC;        -- Entrada de reloj
           Q   : out  STD_LOGIC_VECTOR (3 downto 0); -- Salidas
           RCO : out  STD_LOGIC);  -- Salida RCO activa si Q=15
end contador;

architecture a_contador of contador is

signal flag : STD_LOGIC :='0';
signal divisor : STD_LOGIC_VECTOR (15 downto 0) := "0000000000000000";
signal QS : STD_LOGIC_VECTOR (3 downto 0) := "0000";

begin

SYNC : process (CLK)               -- proceso que divide el reloj
begin
  if (CLK'event and CLK='1') then
    divisor<=divisor+1;
    if divisor=2500 then           -- tras 2500 cuentas han transcurrido
      divisor<=(others=>'0');      -- 50 us, luego la frec. de la señal
      flag <= not flag;            -- flag será de 10 KHz.
    end if;
  end if;
end process;

CONT : process (flag)
begin
  if (flag'event and flag='1') then
    QS<=QS+1;
  end if;
end process;

RCO<= QS(3) and  QS(2) and QS(1) and QS(0); -- RCO es 1 cuando Q=15
Q <= QS;                           -- asignamos la señal QS a la salida real
end a_contador;
```

Este fichero describe un contador binario de 4 bits. Con un primer proceso (SYNC), se divide la frecuencia del reloj de la FPGA de tal manera que genera una señal (flag) con una frecuencia de 10 KHz. Un segundo proceso (CONT), utiliza esta última para incrementar el contador. La señal RCO será 1 cuando todas las salidas del contador sean 1 simultáneamente.

FICHERO DE ASOCIACIONES: asociaciones.ucf

```
NET "CLK" LOC = "M6";   # RELOJ DE LA FPGA

NET "Q<0>" LOC = "A9";  #Salida terminal 14
NET "Q<1>" LOC = "B9";  #Salida terminal 15
NET "Q<2>" LOC = "A10"; #Salida terminal 16
NET "Q<3>" LOC = "C9";  #Salida terminal 17
NET "RCO" LOC = "C12";  #Salida terminal 18
```

En este caso, volcamos las salidas del contador (salidas Q y RCO) hacia los terminales del tablero. Conectando la sonda entre los terminales 14 y 25 (SALIDA) podrá observar la señal Q(0). El resto de las señales podrán medirse en los terminales 15, 16, 17 y 18.

Ejemplo 6: Observación de señales internas del circuito.

Imagine que en el ejemplo anterior quisiera observar en el osciloscopio la forma de onda de la señal “flag”. Dicha señal es interna al circuito y por lo tanto no está definida como salida o entrada en la declaración “entity”, ni puede por tanto asignarse a ningún terminal en el fichero de asociaciones.

Para visualizar esta señal será necesario modificar el fichero contador.vhd de tal manera que defina una nueva salida (que llamaremos TP1 “terminal de prueba 1”). Esta salida se asignará a la señal “flag”. Posteriormente se modificará el fichero asociaciones.ucf para volcar dicha salida hacia uno de los terminales externos de la FPGA.

FICHERO contador.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador is
    Port ( CLK : in   STD_LOGIC;
           Q   : out  STD_LOGIC_VECTOR (3 downto 0);
           RCO : out  STD_LOGIC;
           TP1 : out  STD_LOGIC );  -- nueva salida (terminal de prueba)
end contador;

architecture a_contador of contador is

signal flag : STD_LOGIC :='0';
signal divisor : STD_LOGIC_VECTOR (15 downto 0) := "0000000000000000";
signal QS : STD_LOGIC_VECTOR (3 downto 0) := "0000";

begin

SYNC : process (CLK)               -- proceso que divide el reloj
begin
  if (CLK'event and CLK='1') then
    divisor<=divisor+1;
    if divisor=2500 then           -- tras 2500 cuentas han transcurrido
      flag<=not flag;              -- 50 us, luego la frec. de la señal
      divisor<=(others=>'0');      -- flag será de 10 KHz.
    end if;
  end if;
end process;

CONT : process (flag)
begin
  if (flag'event and flag='1') then
    QS<=QS+1;
  end if;
end process;

RCO<= QS(3) and  QS(2) and QS(1) and QS(0); -- RCO es 1 cuando Q=15
Q <= QS;                           -- asignamos la señal QS a la salida real
TP1<=flag;                         -- asignamos la señal flag a TP1 para medirla
end a_contador;
```

Modificación del fichero del ejemplo 5 para poder visualizar la señal interna “flag”. Las líneas subrayadas son las añadidas respecto al fichero del ejemplo 5.

FICHERO DE ASOCIACIONES: asociaciones.ucf

```
NET "CLK" LOC = "M6";   # RELOJ DE LA FPGA

NET "Q<0>" LOC = "A9";  #Salida terminal 14
NET "Q<1>" LOC = "B9";  #Salida terminal 15
NET "Q<2>" LOC = "A10"; #Salida terminal 16
NET "Q<3>" LOC = "C9";  #Salida terminal 17
NET "RCO" LOC = "C12";  #Salida terminal 18
NET "TP1" LOC = "A13";  #Salida terminal 19
```

Ahora, volcamos las salidas del contador (salidas Q, TP1 y RCO) hacia los terminales del tablero. Conectando la sonda entre los terminales 19 y 25 (SALIDA) podrá observar la señal "flag". El resto de las señales podrán medirse en los terminales 14, 15, 16, 17 y 18.

Empleo de señales de reloj externas (diferentes de CLK = 50 MHz)

En ocasiones puede ser necesario excitar algunos de los componentes secuenciales diseñados en la FPGA mediante el empleo de señales de reloj diferentes a la señal principal de la tarjeta BASYS2 (CLK=50 MHz). Por ejemplo, se podría definir un biestable tipo JK donde las entradas J y K estuviesen conectadas a los conmutadores de la tarjeta y la señal de reloj del biestable estuviese conectada a uno de los pulsadores.

En estos casos, el compilador VHDL del ISE nos dará un error de implementación puesto que esta es una práctica poco recomendable que puede dar lugar a fallos en el sistema si el reloj que se utiliza no es suficientemente fiable (por ejemplo posee skew, rebotes o retardos).

Es posible, no obstante, forzar al compilador para que admita dicha línea de reloj. En este caso deberá añadirse una *directiva de compilación* al fichero de asociaciones antes de la definición de la línea de reloj. Suponga que la línea de reloj ha sido definida como CLK en los ficheros VHDL. Para escribir su asociación, por ejemplo, a uno de los pulsadores externos (BTN0, patilla G12 de la FPGA), deberá escribirse:

```
NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE;
NET "CLK" LOC = "G12"; # Pulsador BTN0
```

La directiva **CLOCK_DEDICATED_ROUTE = FALSE** fuerza al compilador a admitir que la línea CLK sea asignada al pulsador, en lugar de estar conectada al reloj de la FPGA.

Ejemplo 7: Diseño de un flip-flop JK con su entrada de reloj conectada a un pulsador

Ya que se ha citado el ejemplo en el párrafo anterior, a continuación se detalla el diseño de un flip flop de tipo JK cuyas entradas J y K están conectadas a dos de los conmutadores de la placa, mientras que la entrada de reloj está conectada a un pulsador. Las dos salidas (Q y QN, negada de la anterior) se visualizan a través de dos LEDs. Sintetizando este ejemplo podrá probar el funcionamiento del flip flop produciendo flancos activos de reloj mediante el pulsador.

FICHERO flip_flop_JK.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity flip_flop_JK is
    Port ( J    : in   STD_LOGIC;  -- Entrada J del flip flop
           K    : in   STD_LOGIC;  -- Entrada K del flip flop
           CLK  : in   STD_LOGIC;  -- Entrada CLK del flip flop
           Q    : out  STD_LOGIC;  -- Salida del flip flop
           QN   : out  STD_LOGIC); -- Salida negada
end flip_flop_JK;

architecture a_flip_flop_JK of flip_flop_JK is

signal QS : STD_LOGIC := '0';

begin
  process (CLK)
    begin
    if (CLK'event and CLK='1') then
      if    (J='0' and K='0') then   -- funcionamiento del JK
        QS<=QS;
      elsif (J='1' and K='0') then
        QS<='1';
      elsif (J='0' and K='1') then
        QS<='0';
      elsif (J='1' and K='1') then
        QS<=not QS;
      end if;
    end if;
  end process;

Q<=QS;         -- se asignan las señales a las salidas reales
QN<= not QS;

end a_flip_flop_JK;
```

En este caso se trata de un proceso con CLK en su lista de sensibilidad. En cada flanco activo de CLK se comprueban los valores de J y K y se genera la salida correspondiente del biestable en la variable QS. En paralelo, dicha señal es empleada para obtener las salidas complementarias Q y QN.

FICHERO DE ASOCIACIONES: asociaciones.ucf

```
#Pulsadores

NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE; #Forzar al compilador a admitir el
                                         #pulsador como línea de reloj.
NET "CLK" LOC = "G12"; # Pulsador BTN0

# Conmutadores

NET "J"  LOC = "N3";  # Conmutador SW7
NET "K"  LOC = "E2";  # Conmutador SW6

#LEDs

NET "Q"  LOC = "G1";  # LED 7
NET "QN" LOC = "P4";  # LED 6
```

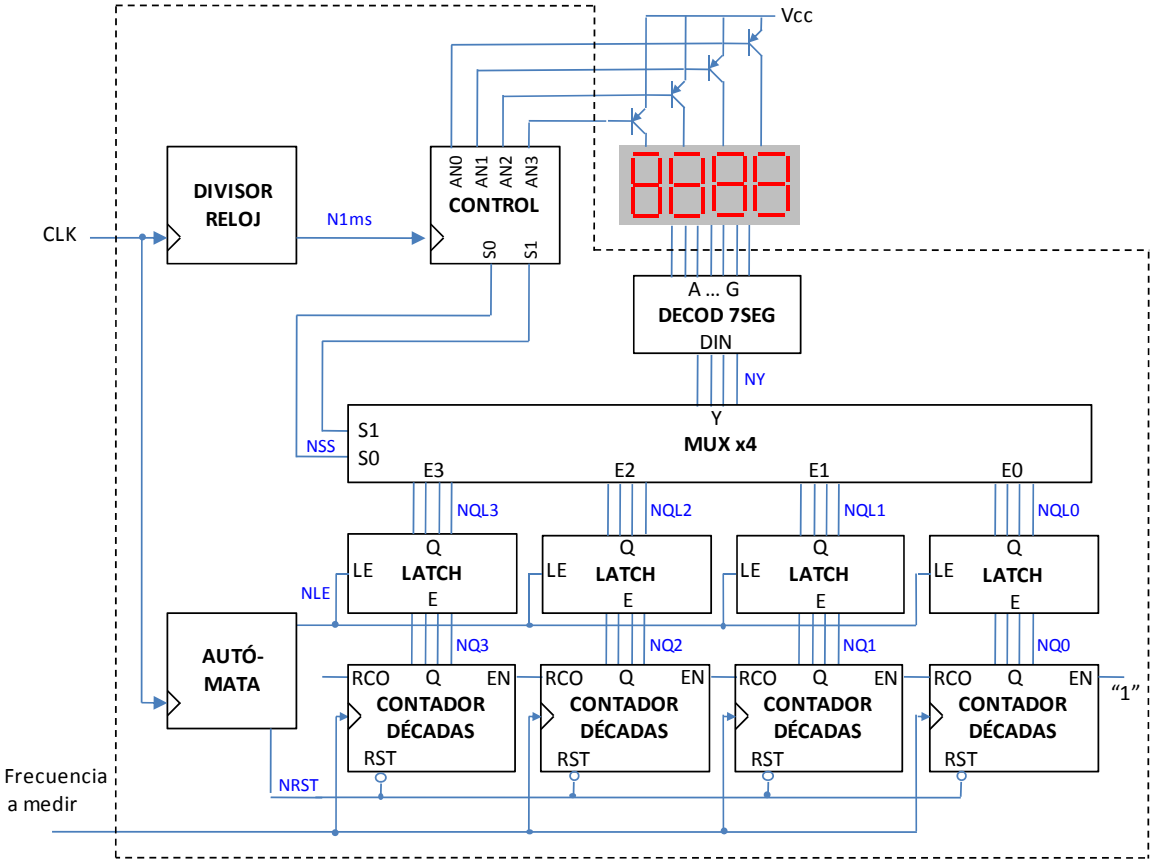
Observe que se incluye la directiva de compilación para forzar la utilización del pulsador como línea de reloj.

Ejemplo 8: Utilización de una entrada externa para el desarrollo de un frecuencímetro.

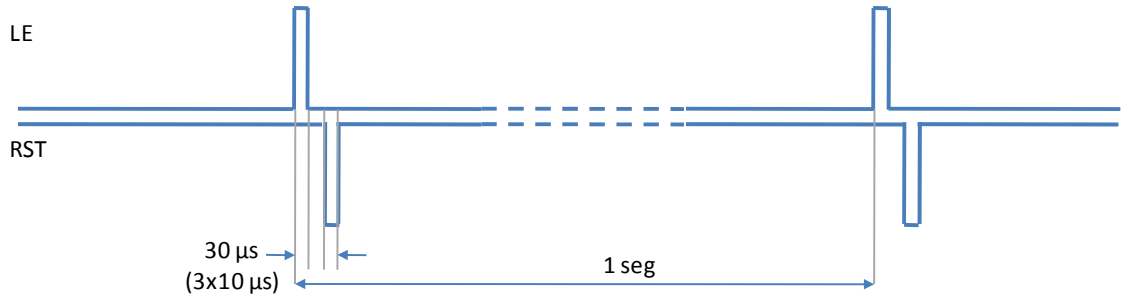
Queremos desarrollar un frecuencímetro que sea capaz de medir frecuencias entre 1 Hz y 9999 Hz y que presente la lectura en los displays de la tarjeta BASYS2. La señal entrante deberá ser digital TTL (es decir con valores de 0V y 5V para el "0" y el "1" respectivamente), de tal manera que sea compatible con la lógica de la tarjeta BASYS2. Esta señal es sencilla de obtener, puesto que está disponible en el conector "TTL" del generador de funciones.

El funcionamiento interno de un frecuencímetro se basa en contar los flancos activos de la señal entrante durante un periodo de tiempo conocido T. Puesto que el resultado de dicha cuenta se presenta en los displays, es importante diseñar adecuadamente el valor del tiempo T para que el número visualizado se corresponda realmente con la frecuencia expresada en Hz. En nuestro caso, cuando el valor presentado sea 9999 habrán transcurrido realmente 10000·100 µs = 1s , por lo tanto el valor del intervalo T se corresponderá con esta cifra. Variando el valor de T se pueden obtener diferentes precisiones y diferentes escalas de medida.

La siguiente figura, resume de forma simplificada el funcionamiento del frecuencímetro.



En esencia, un autómata controla el funcionamiento del frecuencímetro. Cada segundo, deberá generar un pulso en las patillas LE ("latch enable") para que el valor de la cuenta se quede retenido y a continuación deberá generar otro pulso en RST que ponga a 0 los contadores (ver figura siguiente). La anchura de los pulsos es de 10 µs separados entre sí también 10 µs.



Por otro lado, un sistema de control actualiza la visualización de los displays de un modo similar a los ejemplos 3 y 4.

Como la entrada externa por la que se suministrará la frecuencia a medir se va a utilizar como señal de reloj para los contadores, es necesario forzar al compilador para que admita dicha línea como señal externa de reloj (fíjese en el fichero de asociaciones).

Al igual que en ejemplos anteriores, lo que se encuentra dentro de la zona punteada es el circuito que debemos sintetizar en la FPGA, mientras que lo que se encuentra fuera de dicha línea se corresponde con los diferentes recursos de la tarjeta y las líneas exteriores.

Este proyecto, al ser de gran envergadura, estará también dividido en varios ficheros, los cuales se describen detalladamente a continuación:

FICHERO contador_decadas.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador_decadas is
    Port ( CLK    : in  STD_LOGIC;          -- entrada de reloj
           EN     : in  STD_LOGIC;          -- entrada de ENABLE
           RST_L  : in  STD_LOGIC;          -- entrada de RESET asíncrona
           Q      : out  STD_LOGIC_VECTOR (3 downto 0); -- Salida
           RCO    : out  STD_LOGIC); -- Salida de fin de cuenta
end contador_decadas;

architecture a_contador_decadas of contador_decadas is

signal QS : STD_LOGIC_VECTOR (3 downto 0) := "0000";

begin
  process (CLK,RST_L)        -- RESET asíncrono en lista de sensibilidad
    begin
       if (RST_L='0') then                       -- Si RST=0, el contador se pone a 0
         QS<="0000";
       elsif (CLK'event and CLK='1') then
         if (EN='1') then                -- en cada ciclo de reloj
           QS<=QS+1;                     -- si EN=1 el contador se incrementa
           if (QS=9) then     -- si llega a 9 vuelve a 0 en el sig. ciclo
             QS<="0000";
           end if;
         else
           QS<=QS;
         end if;
       end if;
    end process;

Q<=QS;

RCO<='1' when QS="1001" and EN='1' else '0';-- RCO es 1 cuando Q=9 y EN=1
end a_contador_decadas;
```

Este contador de décadas tiene una entrada de reloj (CLK) y una entrada de ENABLE. Cuando el ENABLE está activo ("1"), el contador se incrementa con cada ciclo de reloj. Si la cuenta llega a 9, se vuelve a 0 en el siguiente ciclo. La salida RCO se utiliza para encadenar contadores, por lo tanto será 1 cuando la salida es igual a 9 y cuando el ENABLE está activo. Además también tiene una entrada de RESET (RST) asíncrona, activa a nivel bajo, que pone el contador a 0 independientemente del reloj.

FICHERO latch.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity latch is
    Port ( E : in  STD_LOGIC_VECTOR (3 downto 0);  -- entrada del latch
           Q : out  STD_LOGIC_VECTOR (3 downto 0); -- salida del latch
           LE : in  STD_LOGIC);                    -- activación del latch
end latch;

architecture a_latch of latch is

signal QS : STD_LOGIC_VECTOR (3 downto 0);

begin

process (LE,E)
  begin
    if (LE='1') then
      QS<=E;
    end if;
  end process;

Q<=QS;

end a_latch;
```

Este módulo define un latch de 4 bits. Tiene una entrada de 4 bits y una salida de 4 bits. La señal de activación es LE. Mientras LE es 1, se copia el valor de la entrada en la salida. Cuando LE es igual a 0, la salida se queda congelada en el último valor que adquirió.

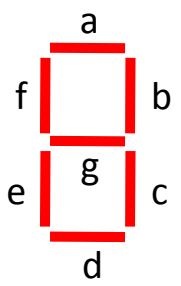
FICHERO MUX4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux4 is
    Port ( E0 : in  STD_LOGIC_VECTOR (3 downto 0);
           E1 : in  STD_LOGIC_VECTOR (3 downto 0);
           E2 : in  STD_LOGIC_VECTOR (3 downto 0);
           E3 : in  STD_LOGIC_VECTOR (3 downto 0);
           Y  : out STD_LOGIC_VECTOR (3 downto 0);
           S  : in  STD_LOGIC_VECTOR (1 downto 0));

end mux4;

architecture a_mux4 of mux4 is
begin

Y <= E0 when S="00" else
     E1 when S="01" else
     E2 when S="10" else
     E3 when S="11";

end a_mux4;
```

Al igual que en el ejemplo 4, este multiplexor se emplea para la visualización en los displays. Posee 4 entradas de 4 bits, una salida de 4 bits y dos entradas de control.

FICHERO decod7s.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decod7s is
  port ( DIN   : in  STD_LOGIC_VECTOR (3 downto 0);   -- entrada de datos
         S7SEG : out STD_LOGIC_VECTOR (0 to 6));   -- salidas 7seg (abcdefg)
end decod7s;

architecture a_decod7s of decod7s is
begin
   with DIN select S7SEG <=
      "0000001" when "0000",
      "1001111" when "0001",
      "0010010" when "0010",
      "0000110" when "0011",
      "1001100" when "0100",
      "0100100" when "0101",
      "0100000" when "0110",
      "0001111" when "0111",
      "0000000" when "1000",
      "0001100" when "1001",
      "1111111" when others;
end a_decod7s;
```



Nuevamente, como en el ejemplo 4, este decodificador de binario a 7 segmentos se emplea para la visualización en los displays. Posee una entrada de datos de 4 bits y 7 salidas para cada uno de los segmentos (vea ejemplos 3 y 4).

FICHERO control.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control is
    Port ( CLK : in  STD_LOGIC;
           AN : out  STD_LOGIC_VECTOR (3 downto 0);
           S : out  STD_LOGIC_VECTOR (1 downto 0));
end control;

architecture a_control of control is

signal SS  : STD_LOGIC_VECTOR (1 downto 0);

begin

process (CLK)
  begin
  if (CLK'event and CLK='1') then
    SS<=SS+1;
  end if;
  end process;

S<=SS;
AN<="0111" when SS="00" else
    "1011" when SS="01" else
    "1101" when SS="10" else
    "1110" when SS="11";

end a_control;
```

Este código es el mismo que el utilizado en el ejemplo 4. Es un módulo secuencial que utiliza una señal de reloj para incrementar un contador de 2 bits que repite la secuencia 00, 01, 10 y 11 indefinidamente. Con esta secuencia se genera una señal AN que activa cada uno de los displays (mediante un '0'). Esto se empleará para realizar el refresco de los displays. La secuencia de dos bits conmutará las entradas del multiplexor y las señales AN activarán los displays en consonancia, de un modo similar al del ejemplo 3 pero con 4 displays.

FICHERO div_reloj.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity div_reloj is
    Port ( CLK : in  STD_LOGIC;        -- entrada de reloj del sistema
           CLK_1MS : out  STD_LOGIC);  -- salida de reloj de frecuencia menor
end div_reloj;

architecture a_div_reloj of div_reloj is

signal div    : STD_LOGIC_VECTOR (15 downto 0);
signal S_1MS  : STD_LOGIC;

begin
  process (CLK)
    begin
      if (CLK'event and CLK='1') then
        div<=div+1;                    -- con cada ciclo de reloj se incrementa div
        if div=25000 then              -- tras 25000 cuentas han transcurrido 500 us
          S_1MS<=not S_1MS;            -- la frecuencia de S_1MS es de 1 KHz
          div<=(others=>'0');
        end if;
      end if;
  end process;

CLK_1MS<=S_1MS;

end a_div_reloj;
```

También como en el ejemplo 4, este código realiza la división de frecuencia del reloj del sistema proporcionando a la salida un reloj de frecuencia 1 KHz para el refresco periódico de los displays.

FICHERO automata.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity automata is
  port ( CLK   : in  STD_LOGIC;   -- reloj de la FPGA
         RST_L : out STD_LOGIC;   -- Salida de reset para los contadores
         LE    : out STD_LOGIC);  -- salida de latch enable
end automata;
```

```
architecture a_automata of automata is

type STATE_TYPE is (CUENTA,LE_ON,LE_OFF,RESET_CNT); -- autómata de 4 estados

signal NWS : STATE_TYPE := CUENTA ; --estado actual
signal DivCLK : STD_LOGIC_VECTOR (31 downto 0) := "00000000000000000000000000000000";
--contador para dividir la frecuencia del reloj.

begin
  process (CLK)
  begin
    if (CLK'event and CLK='1') then
      DivCLK<=DivCLK+1;
      case NWS is
        when CUENTA =>
          if (DivCLK=50000000) then  -- El estado CUENTA dura 1s
            NWS <= LE_ON;
            DivCLK<=(others=>'0');
          else
            NWS <= CUENTA;
          end if;

        when LE_ON =>
          if (DivCLK=500) then  -- El estado LE_ON dura 10 us
            NWS <= LE_OFF;
            DivCLK<=(others=>'0');
          else
            NWS <= LE_ON;
          end if;

        when LE_OFF =>
          if (DivCLK=500) then  -- El estado LE_OFF dura 10 us
            NWS <= RESET_CNT;
            DivCLK<=(others=>'0');
          else
            NWS <= LE_OFF;
          end if;

        when RESET_CNT =>
          if (DivCLK=500) then  -- El estado RESET_CNT dura 10 us
            NWS <= CUENTA;
            DivCLK<=(others=>'0');
          else
            NWS <= RESET_CNT;
          end if;
      end case;
    end if;
  end process;

LE <=    '1' when NWS=LE_ON else '0';      -- Las salidas dependen del estado
RST_L <= '0' when NWS=RESET_CNT else '1';  -- Es un autómata de Moore

end a_automata;
```

Este autómata es el núcleo del frecuencímetro, pues proporciona las señales que activan la captura de la cuenta y la puesta a 0 de los contadores en los instantes de tiempo determinados. El autómata pasa cíclicamente por 4 estados:

- **CUENTA**: El autómata permanece 1 segundo en este estado con las dos salidas desactivadas (LE=0 y RST=1)
- **LE_ON**: El autómata permanece 10 µs en este estado con la salida LE activa (LE=1 y RST=1)
- **LE_OFF**: El autómata permanece 10 µs en este estado con ambas salidas desactivadas (LE=0 y RST=1)
- **RESET_CNT**: autómata permanece 10 µs en este estado con la salida RST activa (LE=0 y RST=0)

FICHERO circuito.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity circuito is
    Port ( CLK         : in  STD_LOGIC;
           FREQ_INPUT  : in  STD_LOGIC;
           AN          : out  STD_LOGIC_VECTOR (3 downto 0);
           SEG7        : out  STD_LOGIC_VECTOR (0 to 6));
end circuito;

architecture a_circuito of circuito is

component contador_decadas
  Port   ( CLK    : in  STD_LOGIC;
           EN     : in  STD_LOGIC;
           RST_L  : in  STD_LOGIC;
           Q      : out  STD_LOGIC_VECTOR (3 downto 0);
           RCO    : out  STD_LOGIC);
end component;

component latch
  Port   ( E : in  STD_LOGIC_VECTOR (3 downto 0);
           Q : out  STD_LOGIC_VECTOR (3 downto 0);
           LE : in  STD_LOGIC);
end component;

component MUX4
  Port   ( E0 : in  STD_LOGIC_VECTOR (3 downto 0);
           E1 : in  STD_LOGIC_VECTOR (3 downto 0);
           E2 : in  STD_LOGIC_VECTOR (3 downto 0);
           E3 : in  STD_LOGIC_VECTOR (3 downto 0);
           Y : out  STD_LOGIC_VECTOR (3 downto 0);
           S : in  STD_LOGIC_VECTOR (1 downto 0));
end component;

component decod7s
  port   ( DIN   : in  STD_LOGIC_VECTOR (3 downto 0);
           S7SEG : out STD_LOGIC_VECTOR (0 to 6));
end component;

component control
  Port   ( CLK : in  STD_LOGIC;
           AN : out  STD_LOGIC_VECTOR (3 downto 0);
           S : out  STD_LOGIC_VECTOR (1 downto 0));
end component;

component div_reloj
  Port   ( CLK : in  STD_LOGIC;
           CLK_1MS : out  STD_LOGIC);
end component;

component automata
  port  ( CLK   : in  STD_LOGIC;
          RST_L : out STD_LOGIC;
          LE    : out STD_LOGIC);
end component;

signal NQ0 : STD_LOGIC_VECTOR (3 downto 0); -- nodos del circuito
signal NQ1 : STD_LOGIC_VECTOR (3 downto 0);
signal NQ2 : STD_LOGIC_VECTOR (3 downto 0);
signal NQ3 : STD_LOGIC_VECTOR (3 downto 0);
signal NQL0 : STD_LOGIC_VECTOR (3 downto 0);
signal NQL1 : STD_LOGIC_VECTOR (3 downto 0);
signal NQL2 : STD_LOGIC_VECTOR (3 downto 0);
signal NQL3 : STD_LOGIC_VECTOR (3 downto 0);
signal NY : STD_LOGIC_VECTOR (3 downto 0);
signal NSS : STD_LOGIC_VECTOR (1 downto 0);
signal NLE : STD_LOGIC;
signal NRST : STD_LOGIC;
signal N1MS : STD_LOGIC;
signal NRCO0 : STD_LOGIC;
signal NRCO1 : STD_LOGIC;
signal NRCO2 : STD_LOGIC;
```

```
signal NRCO3 : STD_LOGIC;

begin

U1 : contador_decadas     -- 4 contadores
       port map (
         CLK=>FREQ_INPUT,
         EN=>'1',
         RST_L=>NRST,
         Q=>NQ0,
         RCO=>NRCO0
         );

U2 : contador_decadas
       port map (
         CLK=>FREQ_INPUT,
         EN=>NRCO0,
         RST_L=>NRST,
         Q=>NQ1,
         RCO=>NRCO1
         );

U3 : contador_decadas
       port map (
         CLK=>FREQ_INPUT,
         EN=>NRCO1,
         RST_L=>NRST,
         Q=>NQ2,
         RCO=>NRCO2
         );

U4 : contador_decadas
       port map (
         CLK=>FREQ_INPUT,
         EN=>NRCO2,
         RST_L=>NRST,
         Q=>NQ3,
         RCO=>NRCO3
         );

U5 : latch                 -- 4 latches
       port map (
         E=>NQ0,
         Q=>NQL0,
         LE=>NLE
         );

U6 : latch
       port map (
         E=>NQ1,
         Q=>NQL1,
         LE=>NLE
         );

U7 : latch
       port map (
         E=>NQ2,
         Q=>NQL2,
         LE=>NLE
         );

U8 : latch
       port map (
         E=>NQ3,
         Q=>NQL3,
         LE=>NLE
         );

U9 : MUX4
       port map (
         E0=>NQL0,
         E1=>NQL1,
         E2=>NQL2,
         E3=>NQL3,
         Y=>NY,
         S=>NSS
         );
```

```
U10 : decod7s
       port map (
         DIN=>NY,
         S7SEG=>SEG7
         );

U11 : control
       port map (
         CLK=>N1MS,
         AN=>AN,
         S=>NSS
         );

U12 : div_reloj
       port map (
         CLK=>CLK,
         CLK_1MS=>N1MS
         );

U13 : automata
       port map (
         CLK=>CLK,
         RST_L=>NRST,
         LE=>NLE
         );

end a_circuito;
```

En este fichero se realiza la interconexión de todos los elementos mediante una descripción estructural que responde al diagrama de bloques presente al principio de este ejemplo. Las señales definidas son los nodos internos del circuito.

FICHERO DE ASOCIACIONES: asociaciones.ucf

```
# Reloj principal del sistema
NET "CLK" LOC = "M6"; # Señal de reloj del sistema


# Conexiones de los DISPLAYS
NET "SEG7<0>" LOC = "L14"; # señal = CA
NET "SEG7<1>" LOC = "H12"; # Señal = CB
NET "SEG7<2>" LOC = "N14"; # Señal = CC
NET "SEG7<3>" LOC = "N11"; # Señal = CD
NET "SEG7<4>" LOC = "P12"; # Señal = CE
NET "SEG7<5>" LOC = "L13"; # Señal = CF
NET "SEG7<6>" LOC = "M12"; # Señal = CG

# Señales de activación de los displays
NET "AN<0>" LOC = "K14"; # Activación del display 0 = AN0
NET "AN<1>" LOC = "M13"; # Activación del display 1 = AN1
NET "AN<2>" LOC = "J12"; # Activación del display 2 = AN2
NET "AN<3>" LOC = "F12"; # Activación del display 3 = AN3

#Entrada externa donde se conecta la señal entrante
NET "FREQ_INPUT" CLOCK_DEDICATED_ROUTE = FALSE; #Forzar al compilador a admitir esta
                                                #línea como entrada de reloj.

NET "FREQ_INPUT" LOC = "B2"; #Entrada terminal 1
```

Finalmente, el fichero de asociaciones asigna cada una de las entradas y salidas del circuito a los recursos físicos de la tarjeta y también a la entrada externa por la que se introduce la señal cuya frecuencia se desea medir.

Por último, para probar este circuito conecte la salida TTL del generador de funciones a la entrada externa (cable rojo al terminal 1 y cable negro (masa) al terminal 25). Ajuste el mando de frecuencia entre 1 Hz y 9999 Hz y dicha frecuencia aparecerá en los displays de la tarjeta.

Utilización del entorno ISE

Para realizar una síntesis mediante el entorno ISE debe crearse un proyecto. El proyecto incluirá todos los ficheros correspondientes al diseño incluido el fichero de asociaciones.

Vaya al menú FILE -> NEW PROJECT. Aparecerá una ventana donde tendrá que escribir el nombre del nuevo proyecto. El entorno creará una carpeta dentro de la ubicación "Location" que aparece en dicha ventana. En dicha carpeta almacenará toda la información del proyecto, incluidos los ficheros VHDL, el fichero de asociaciones y el "bit stream" una vez obtenido. Asegúrese también que en la casilla de selección "Top level Source Type" aparece "HDL".

En la siguiente pantalla deberá ajustar los siguientes valores:

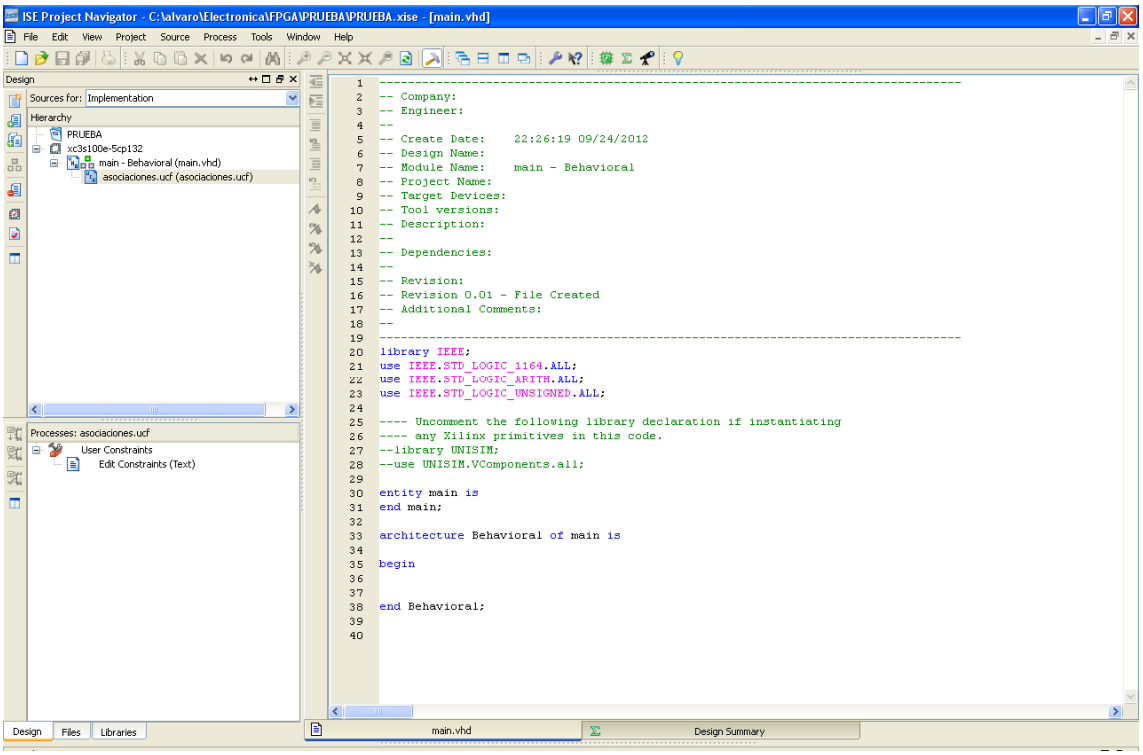
Family: **Spartan 3E**
Device: **XC3S100E**
Package: **CP132**
Speed: **-5**
Synthesis tool: **XST (VHDL/Verilog)**
Simulator: **ISim (VHDL/Verilog)**
Preferred Language: **VHDL**

Por último sáltese las siguientes pantallas pulsando en "Next". En la última aparecerá el botón "Finish".

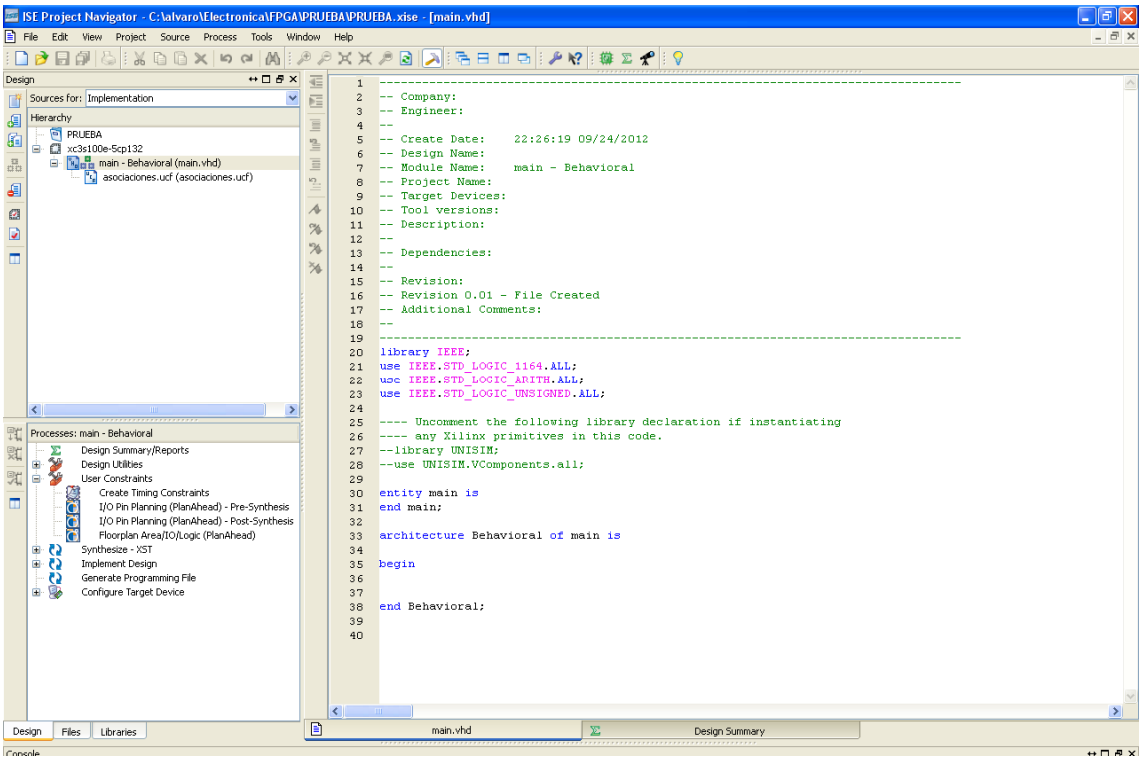
Una vez creado el proyecto pueden añadirse ficheros nuevos pulsando con el botón derecho del ratón sobre el icono "xc3s100e-5cp132" que aparece en la parte izquierda de la pantalla. En el menú que se despliega seleccione "New Source". Si el fichero que pretende añadir ya ha sido creado previamente seleccione entonces "Add Source".

Elija VHDL module para crear un nuevo archivo VHDL. Escriba un nombre para el archivo y pase a la siguiente pantalla. En ella puede declarar las entradas y las salidas del módulo (sección port dentro de la declaración entity). Si lo desea puede pasar esta pantalla y declarar las entradas y salidas manualmente más tarde. Tras esta pantalla aparecerá un editor con el esqueleto principal del código VHDL ya escrito. Puede escribir en el archivo y grabar los cambios con el icono de un diskette en la parte superior de la pantalla.

Para el archivo de asociaciones elija "Implementation Constraints File" y escriba un nombre. Se creará un archivo con extensión UCF. Para acceder a él, selecciónelo en la parte superior izquierda de la pantalla, luego pulse con el ratón el icono "Edit Constraints (Text)" en la parte inferior izquierda (ver figura).



Para compilar el código y generar el “bit stream” , seleccione el fichero principal del proyecto y haga doble click en “Generate Programming File” en la parte inferior izquierda. Vea la figura siguiente:



Evidentemente tendrá que depurar todos los errores existentes en el código hasta que se pueda generar un fichero “bit stream”. En el momento en que todo esté correcto aparecerá un icono verde a la izquierda de la etiqueta “Generate Programming File”.

Simulación de circuitos mediante ISIM

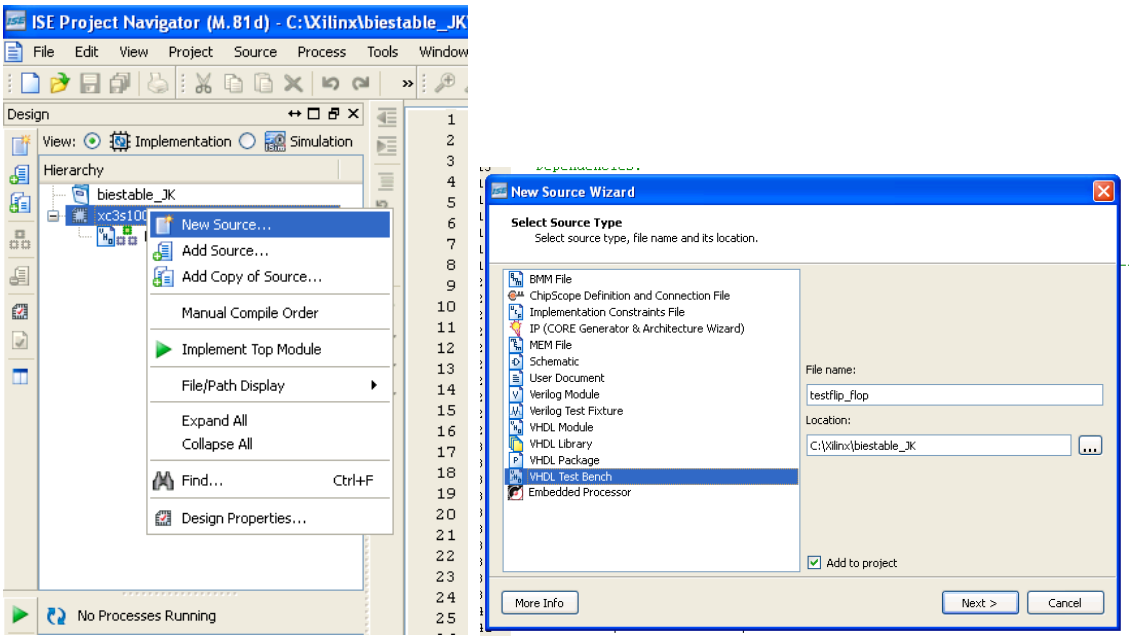
ISIM es una herramienta integrada en el entorno ISE para la simulación de circuitos. Con esta herramienta es posible probar el funcionamiento de los mismos antes de su síntesis en la tarjeta BASYS2. Es muy recomendable simular los circuitos antes de realizar la síntesis. Aunque un circuito que funciona bien en la simulación no tiene por qué ser necesariamente sintetizable, la simulación es un proceso que puede ahorrar tiempo de depuración sobre el circuito real y que además puede ser realizado fuera del laboratorio.

La simulación se lleva a cabo creando un nuevo fichero de test ("*test bench*") donde tendremos que introducir una secuencia lógica dependiente del tiempo en las variables de entrada. Como resultado, el simulador nos representará de forma gráfica los valores de las salidas en función de las citadas entradas.

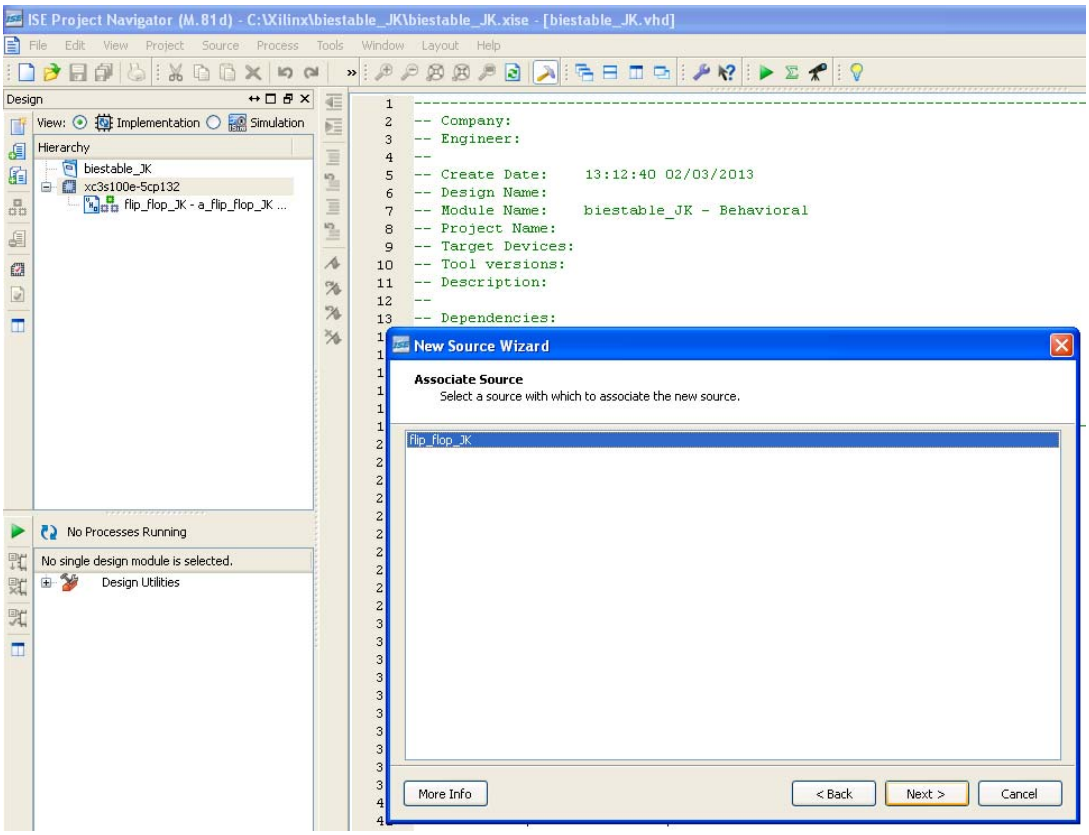
1. Simulación de circuitos secuenciales (síncronos con una señal de entrada de reloj)

Para ilustrar el funcionamiento del simulador ISIM utilizaremos el código VHDL correspondiente al flip flop JK definido en el ejemplo 7 de este documento. Por tanto debemos cargar el código del citado ejemplo en primer lugar.

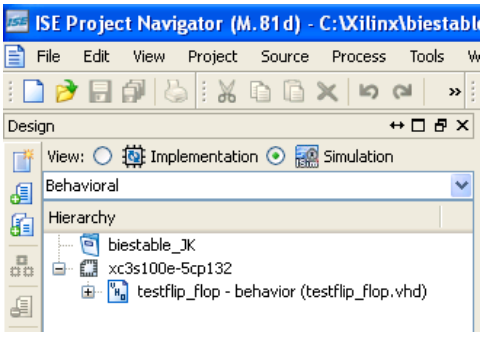
Para añadir al diseño del flip flop el fichero de test haga click con el botón derecho sobre el icono “xc3s100e-5cp132” y elija la opción "New Source". Seleccione en este caso "VHDL test bench" para el tipo e introduzca un nombre para el archivo (por ejemplo "test_flip_flop").



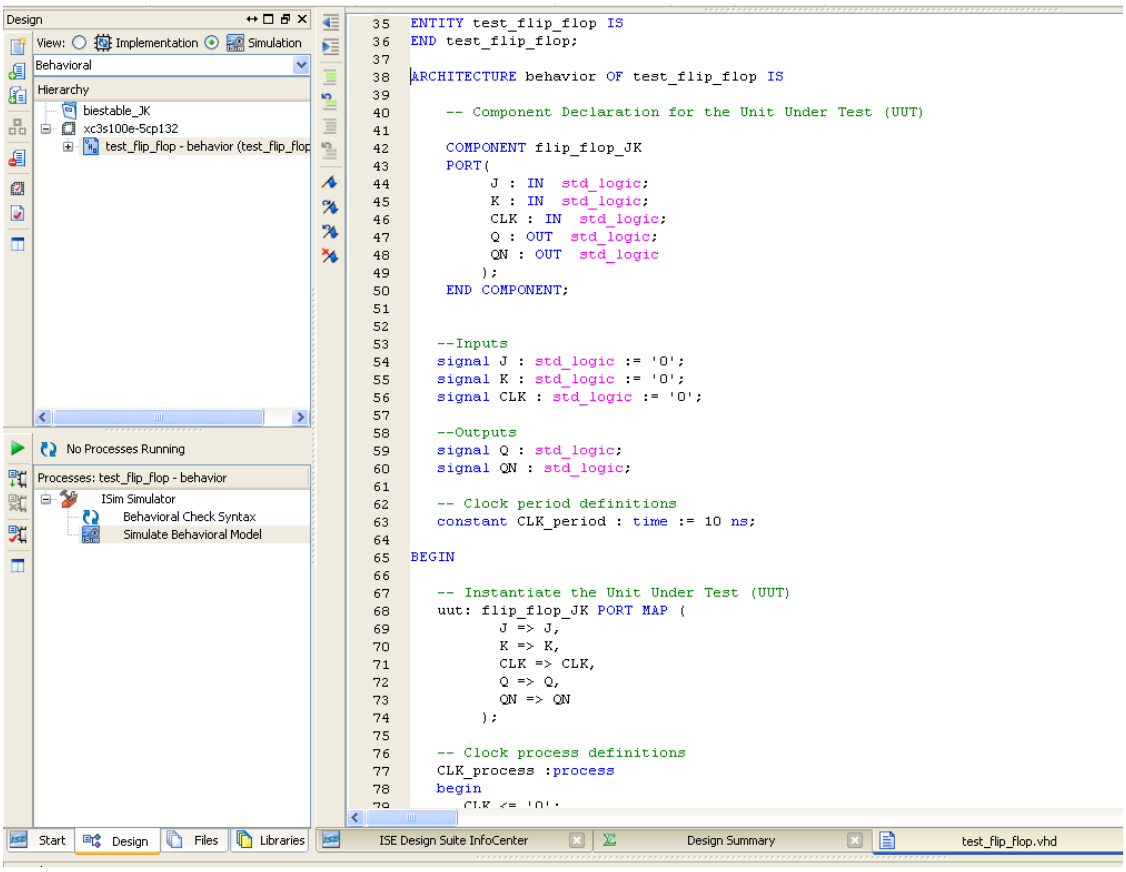
En la siguiente ventana se le preguntará por el fichero al cual quiere asociar el *test bench*. Debe asociarse al fichero principal. En este caso es sencillo porque solamente existe un fichero VHDL, pero en los diseños complejos donde existe más de un fichero, deberá asignarse al fichero principal que constituye la interconexión de los módulos. La simulación se realizará siempre tomando como entradas y salidas las correspondientes al fichero seleccionado (vea la siguiente figura).



Después de hacer click en "Next" y "Finish" el fichero se añadirá al proyecto, aunque aún no aparece. Para ver su contenido deberá pasar del modo "implementation" al modo "Behavioral Simulation" seleccionando la opción en la parte superior de la ventana que contiene los nombres de fichero:



El simulador creará un nuevo fichero con código VHDL. Abra este fichero haciendo doble click sobre la línea correspondiente. Podrá observar que se trata de un código VHDL especial que contiene la declaración de una entidad y una arquitectura. En la figura siguiente se puede observar el test bench creado por el ISE para el ejemplo del flip-flop. Entre las líneas 42 y 50 se define un componente correspondiente al circuito sobre el que se va a realizar la simulación. También se definen señales (líneas 54 a 60) por cada una de las entradas y salidas y se les asigna un valor inicial a las entradas igual a '0' (se puede cambiar por '1' si es necesario). Por último, se asignan dichas señales a las entradas y salidas del componente (líneas 68 a 74).



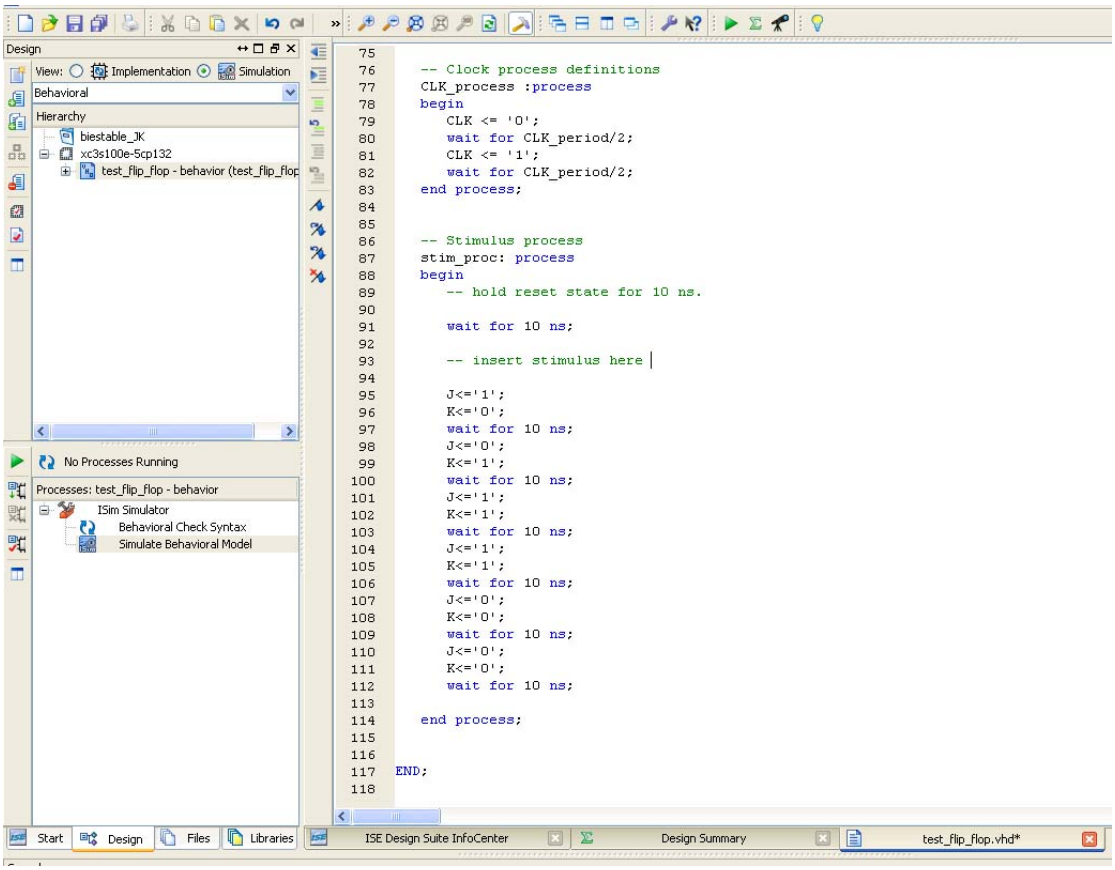
Los dos procesos que siguen a continuación crean la secuencia temporal de las señales de entrada que serán empleadas por el simulador para generar las salidas. Se trata en este caso de dos procesos:

- **CLK_process** crea una secuencia de '0' y '1' con periodo de CLK_period/2 cada uno. Esta secuencia ha sido creada por el ISE tras interpretar que la señal CLK es una señal de reloj. La secuencia se repite indefinidamente.
- **stim_proc** es un proceso en el que debemos detallar la secuencia temporal que queremos analizar para las señales de entrada (en nuestro caso J y K).

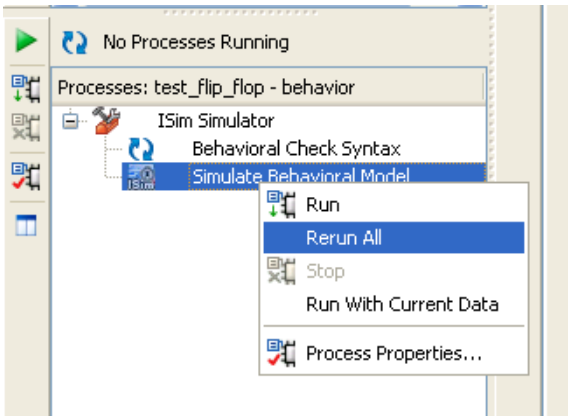
En nuestro caso (en la figura siguiente), hemos escrito algunas líneas con diferentes valores para las entradas (J y K). Tenga en cuenta que se trata solamente de un ejemplo reducido, no un banco de prueba completo para comprobar el funcionamiento exhaustivo del flip flop.

Se inicia el proceso esperando 10 ns (un periodo de reloj). Entre las líneas 95 y 112 damos valores a las entradas y esperamos otros 10 ns para que se produzca un flanco activo de reloj. Nuevamente se cambian los valores de las entradas, y así sucesivamente.

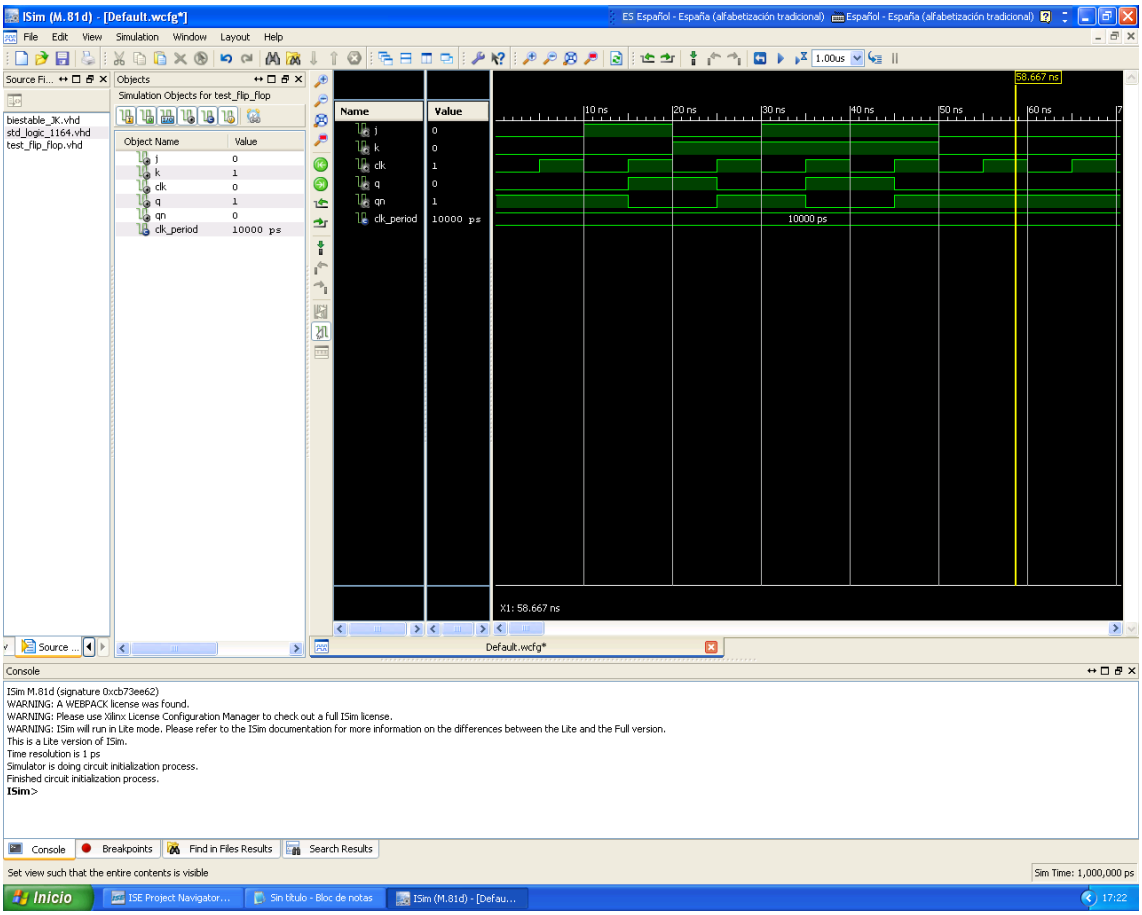
Este proceso se ejecuta en paralelo con el anterior, por lo que es posible sincronizar los valores de las entradas con la señal de reloj. Esta característica es muy importante, por ejemplo, a la hora de simular autómatas.



Para realizar la simulación, haga click con el botón derecho sobre la línea "Simulate Behavioral Model" en la parte inferior izquierda de la pantalla y seleccione "Rerun All".



Esto recompilará el código VHDL junto con el fichero correspondiente al "test bench" creando la simulación. Al terminar se arrancará el entorno de visualización ISIM donde aparecerá el cronograma con las señales de entrada y salida. En dicho entorno tiene varias opciones para poder realizar zoom en el cronograma, observar los valores lógicos de las señales, cambiar el tiempo total de simulación, etc.



2. Simulación de circuitos combinacionales

Los circuitos combinacionales no son dependientes de la señal de un reloj. No obstante, como veremos a continuación, el simulador sigue añadiendo las líneas correspondientes al proceso de sincronización, aunque nos dirá que no identifica la señal de reloj. En este caso el simulador hará referencia a una señal <clock> no existente en el circuito.

Cuando estamos simulando circuitos combinacionales se trabaja únicamente con el proceso stim_proc, y deben eliminarse todas las líneas correspondientes al reloj si no son necesarias. Para comprobar el funcionamiento, se asignarán valores a las entradas y se añadirán tiempos de espera entre variaciones en las mismas.

Como ejemplo para ilustrar esta parte vamos a utilizar el código VHDL de un circuito combinacional que convierte código binario de 5 bits a BCD (decenas y unidades). Su entrada (BIN) será un vector binario de 5 bits y sus salidas serán dos vectores de 4 bits para las unidades y las decenas BCD respectivamente.

Por ejemplo, ante la entrada “11000” (24 decimal), el circuito deberá obtener en sus salidas los valores: “0010” para las decenas y “0100” para las unidades. Ante la entrada “11111” (31 decimal), deberá obtener los valores “0011” para las decenas y “0001” para las unidades. En este ejemplo utilizaremos estos dos valores en la simulación, aunque para comprobar exhaustivamente su funcionamiento deberíamos probar todos los valores posibles de las entradas.

El código correspondiente al convertidor se muestra a continuación:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BIN2BCD is
    Port ( BIN    : in  STD_LOGIC_VECTOR (4 downto 0);   -- entrada binaria
           DECBCD : out  STD_LOGIC_VECTOR (3 downto 0);  -- decenas BCD
           UNIBCD : out  STD_LOGIC_VECTOR (3 downto 0)); -- unidades BCD
end BIN2BCD;

architecture a_BIN2BCD of BIN2BCD is

begin

with BIN select UNIBCD<=
"0000" when "00000",
"0001" when "00001",
"0010" when "00010",
"0011" when "00011",
"0100" when "00100",
"0101" when "00101",
"0110" when "00110",
"0111" when "00111",
"1000" when "01000",
"1001" when "01001",
"0000" when "01010",
"0001" when "01011",
"0010" when "01100",
"0011" when "01101",
"0100" when "01110",
"0101" when "01111",
"0110" when "10000",
"0111" when "10001",
"1000" when "10010",
"1001" when "10011",
"0000" when "10100",
"0001" when "10101",
"0010" when "10110",
"0011" when "10111",
"0100" when "11000",
"0101" when "11001",
"0110" when "11010",
"0111" when "11011",
"1000" when "11100",
"1001" when "11101",
"0000" when "11110",
"0001" when "11111",
"0000" when others;

with BIN select DECBCD<=
"0000" when "00000",
"0000" when "00001",
"0000" when "00010",
"0000" when "00011",
"0000" when "00100",
"0000" when "00101",
"0000" when "00110",
"0000" when "00111",
"0000" when "01000",
"0000" when "01001",
"0001" when "01010",
"0001" when "01011",
"0001" when "01100",
"0001" when "01101",
"0001" when "01110",
"0001" when "01111",
"0001" when "10000",
"0001" when "10001",
"0001" when "10010",
"0001" when "10011",
"0010" when "10100",
"0010" when "10101",
"0010" when "10110",
"0010" when "10111",
"0010" when "11000",
```

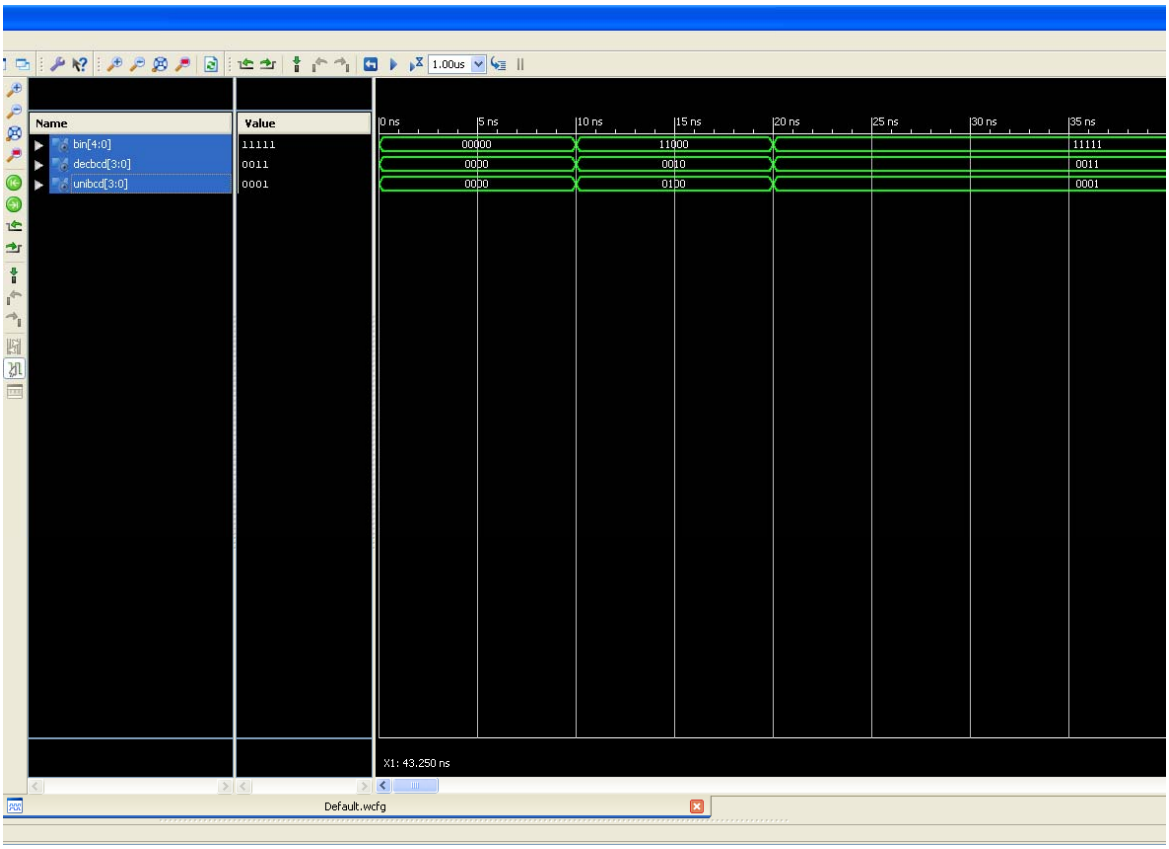
```
"0010" when "11001",
"0010" when "11010",
"0010" when "11011",
"0010" when "11100",
"0010" when "11101",
"0011" when "11110",
"0011" when "11111",
"0000" when others;

end a_BIN2BCD;
```

El test_bench para este circuito se crea de la misma forma que hicimos en el caso anterior. El código que aparecerá hará referencia a una señal <clock> que no existe. Las líneas que contengan esta señal y el proceso de sincronización pueden eliminarse o comentarse (en el siguiente código, correspondiente al citado test_bench, aparecen comentadas y marcadas en rojo).

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY test_BIN2BCD IS
END test_BIN2BCD;

ARCHITECTURE behavior OF test_BIN2BCD IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT BIN2BCD
    PORT(
         BIN : IN  std_logic_vector(4 downto 0);
         DECBCD : OUT  std_logic_vector(3 downto 0);
         UNIBCD : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal BIN : std_logic_vector(4 downto 0) := (others => '0');

       --Outputs
   signal DECBCD : std_logic_vector(3 downto 0);
   signal UNIBCD : std_logic_vector(3 downto 0);

   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

-- constant <clock>_period : time := 10 ns; COMENTAMOS ESTA LINEA (NO NECESARIA)

BEGIN

       -- Instantiate the Unit Under Test (UUT)
   uut: BIN2BCD PORT MAP (
          BIN => BIN,
          DECBCD => DECBCD,
          UNIBCD => UNIBCD
        );

   -- Clock process definitions
--   <clock>_process :process
--   begin
--             <clock> <= '0';
--             wait for <clock>_period/2;
--             <clock> <= '1';
--             wait for <clock>_period/2;
--  end process;


   -- Stimulus process
   stim_proc: process
   begin
```

```
      -- hold reset state for 100 ns.
      wait for 10 ns;
               BIN<="11000";
               wait for 10 ns;
               BIN<="11111";

--       wait for <clock>_period*10;

      -- insert stimulus here

      wait;
   end process;

END;
```

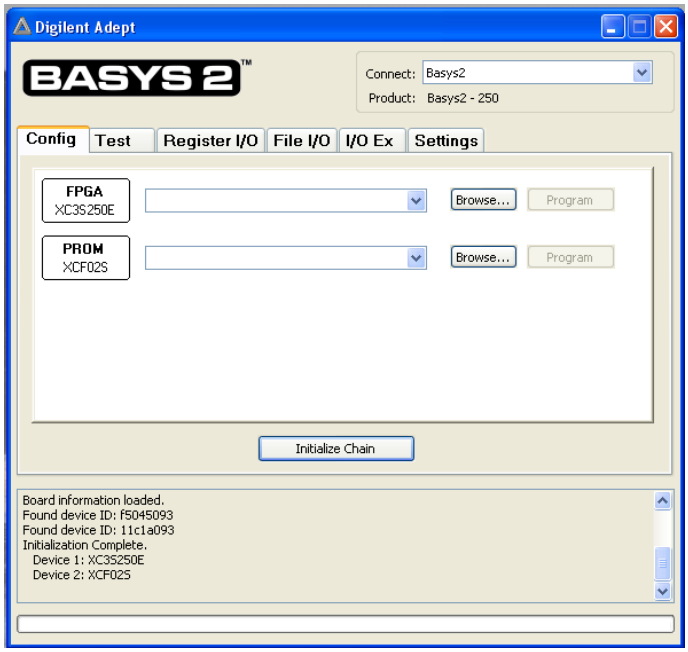
En este código solamente se tiene en cuenta el proceso stim_proc donde asignamos el valor “11000” a la entrada y tras 10 ns asignamos el valor “11111”. Inicialmente se esperan 10 ns en los que la entrada es “00000”. El resultado de la simulación puede verse a continuación, observe que los valores de las salidas son correctos en ambos casos:



n the differences between the Lite and the Full version.

Volcado del “bit stream” sobre la FPGA para sintetizar el circuito

Una vez realizada la síntesis del circuito, se genera un archivo con extensión .bit que contiene la secuencia “bitstream” de configuración interna para la FPGA. Esta secuencia deberá ser cargada a través de la tarjeta BASYS2. Para ello necesita el programa ADEPT de DIGILENT disponible en todos los ordenadores del laboratorio. Arranque dicho programa:



Mediante el botón “Browse…” situado en la línea superior puede seleccionar el archivo que desea cargar. El archivo debe tener extensión .bit

A continuación haga click en el botón “Program”. Acepte la ventana que aparece y el fichero será transferido a la FPGA configurándola de tal manera que sintetice el circuito descrito en su diseño. A partir de este momento, la FPGA comenzará a comportarse según el diseño descrito en el código VHDL.

Bibliografía

BASYS 2 reference manual
(http://www.digilentinc.com/Data/Products/BASYS2/Basys2_rm.pdf)

DIGILENT website:
http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,790&Prod=BASYS2

XILINX ISE Webpack design software:
http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm

Material docente de la asignatura Electrónica Digital (EDIG).

Digital Design, J.F. Wakerly, 4th edition, Prentice Hall, 2005. ISBN: 0-13-186389-4