

Circuitos Electrónicos (CELT)

Manual de referencia de la tarjeta BASYS 2



Álvaro de Guzmán Fernández
Abril de 2022

Manual de referencia de la tarjeta BASYS 2

Índice general

Introducción	3
Descripción general de la tarjeta BASYS 2	4
Síntesis de circuitos mediante la tarjeta BASYS 2	5
El equipo de desarrollo del laboratorio	6
La estructura básica de un diseño VHDL	7
Utilización del entorno ISE para realizar la síntesis sobre FPGA.....	8
Diseño de circuitos combinacionales con VHDL	12
Diseño de circuitos secuenciales síncronos con VHDL.....	14
Tutorial de uso de los recursos de la tarjeta BASYS2 a través de ejemplos.....	16
Ejemplo 1: Empleo de los LEDs y los conmutadores.....	16
Ejercicios propuestos sobre el ejemplo1:	17
Ejemplo 2: Utilización del reloj de la FPGA.	17
Ejercicios propuestos sobre el ejemplo 2:	18
Ejemplo 3: Uso de los displays: Visualización de dos cifras hexadecimales diferentes en dos displays. Las cifras hexadecimales se introducen mediante los conmutadores.	19
Ejercicios propuestos sobre el ejemplo 3:	24
Ejemplo 4: Contador rotatorio	25
Ejercicios propuestos sobre el ejemplo 4:	32
Entradas y salidas externas	33
Ejemplo 5: Observación en el osciloscopio de señales rápidas del circuito VHDL.....	35
Ejercicios propuestos sobre el ejemplo 5:	37
Ejemplo 6: Observación de señales internas del circuito.	37
Ejercicios propuestos sobre el ejemplo 6:	38
Diseño de autómatas de estados finitos	39
Ejemplo 7: Diseño de un autómata de Moore para el control del acceso a una puerta	39
Autómatas cuyos estados duran más de un ciclo de reloj.....	44
Ejemplo 8: Autómata para generación de una señal de emergencia SOS en Morse.....	44
Ejercicios propuestos sobre el ejemplo 8:	47
Simulación de circuitos mediante ISIM	48
1. Simulación de circuitos secuenciales (síncronos con una señal de entrada de reloj).....	48
2. Simulación de circuitos combinacionales	52
Utilización de los convertidores A/D y D/A.....	56
Módulo VHDL para la utilización de los convertidores DAC y ADC.....	57
Ejemplo 9: generación de una onda diente de sierra (sawtooth) en la salida analógica	58
Ejercicios propuestos sobre el ejemplo 9:	61
Ejemplo 10: Voltímetro digital	61
Bibliografía	70

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 hecho se proponen ejercicios en cada uno de los ejemplos para profundizar en el funcionamiento de los mismos. 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").

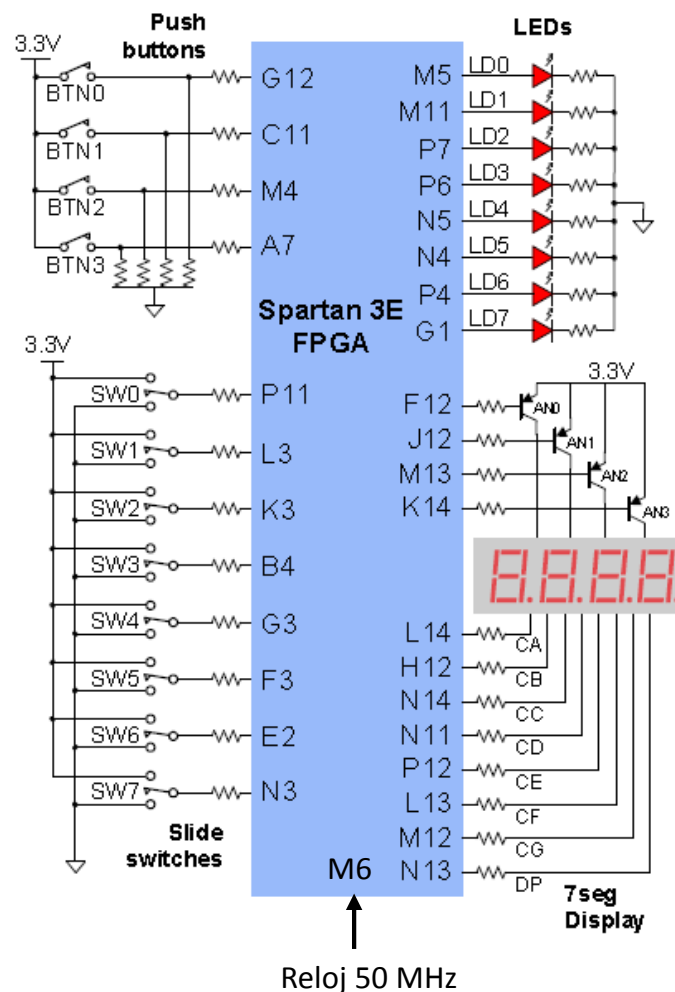
Descripción general de la tarjeta BASYS 2

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:

- Una señal de reloj de 50 MHz.
- 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:



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.

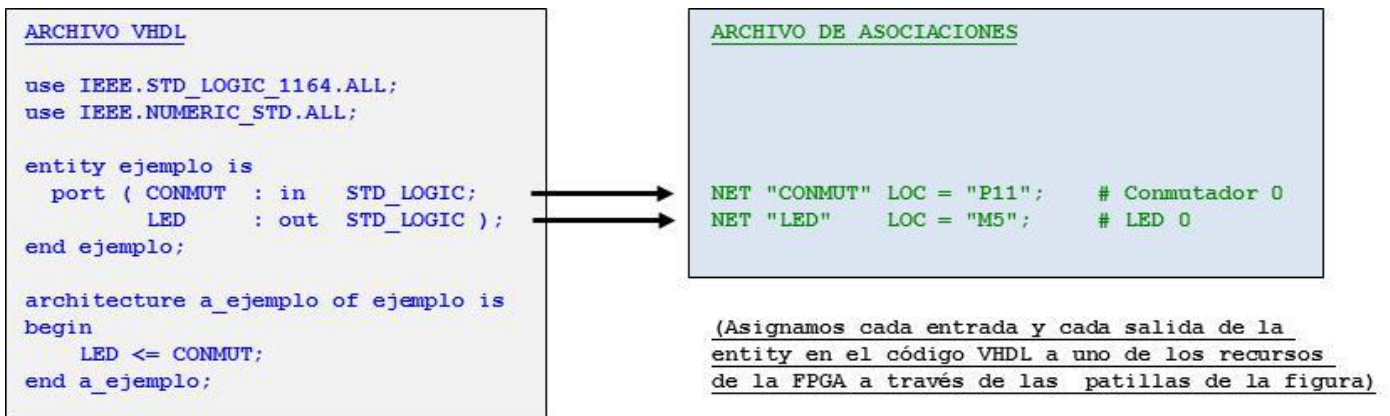


El equipo de desarrollo (armazón azul) posee un interruptor en la parte posterior. **Dicho interruptor deberá estar encendido para que el equipo funcione correctamente.** Asegúrese también que los cables planos se encuentran conectados a los conectores ENTRADAS y SALIDAS del armazón azul.

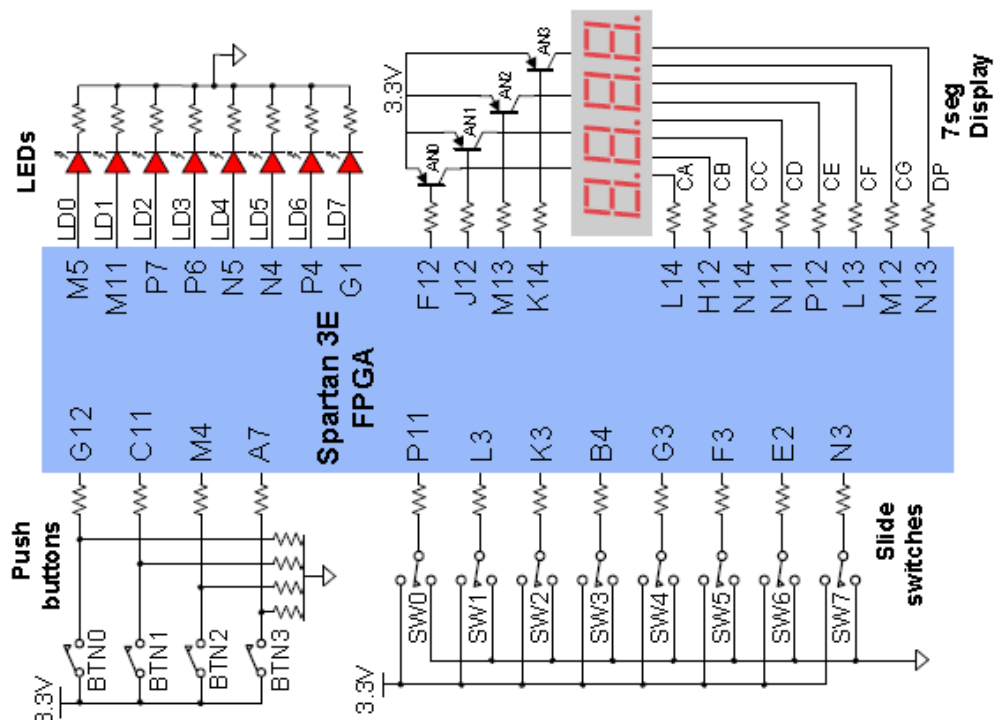
La estructura básica de un diseño VHDL

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 indican las patillas de la FPGA que se corresponden con cada entrada y salida declaradas en las entidades que componen el circuito. Este fichero es IMPRESCINDIBLE para generar el "bit stream". Todas las entradas y salidas del módulo principal VHDL deben tener asociada una patilla de la FPGA.



En este ejemplo tan simple el LED se enciende por medio del conmutador:



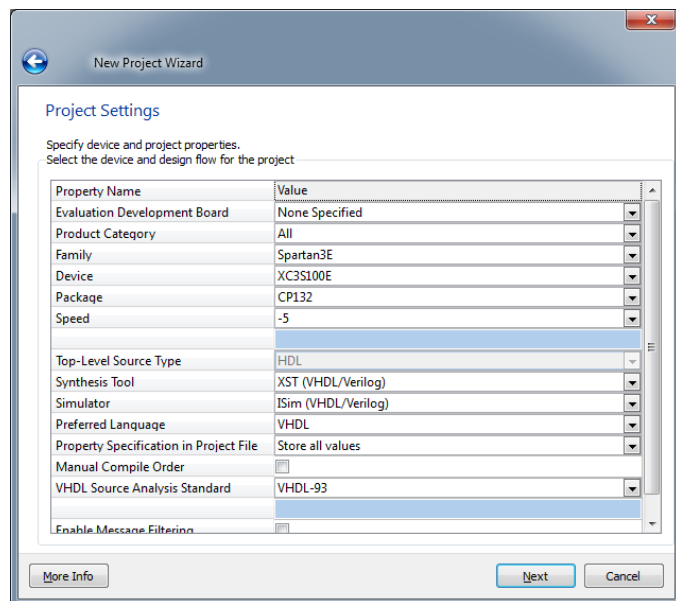
Utilización del entorno ISE para realizar la síntesis sobre FPGA

Para realizar la síntesis de un hardware descrito en VHDL mediante el entorno ISE debe crearse un proyecto. El proyecto incluirá todos los ficheros correspondientes al diseño y además el archivo 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**

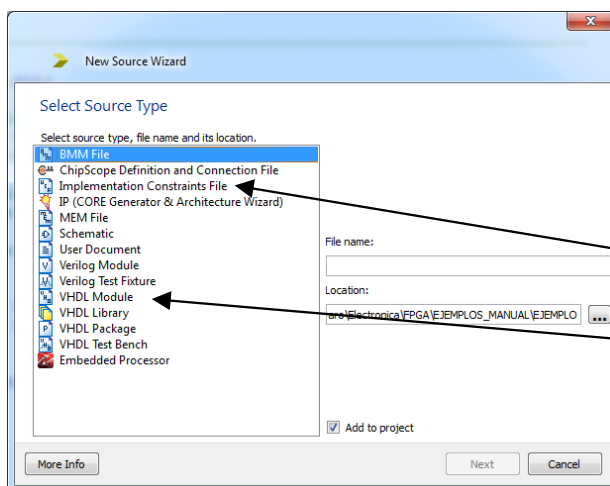
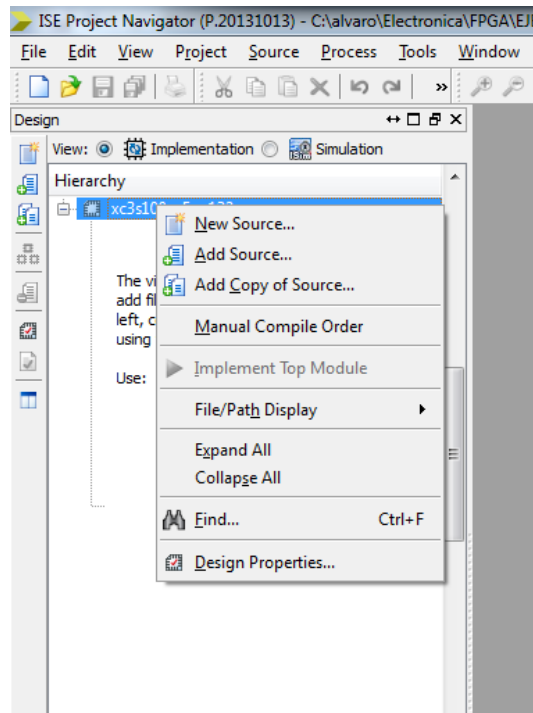


Por último sáltese las siguientes pantallas pulsando en “Next”. En la última aparecerá el botón “Finish”.

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 Copy of 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 para el fichero. Se creará un archivo con extensión UCF. Para acceder a él, selecciónelo en la parte superior izquierda de la pantalla y haga doble click.



Archivo de asociaciones

Archivo de código VHDL

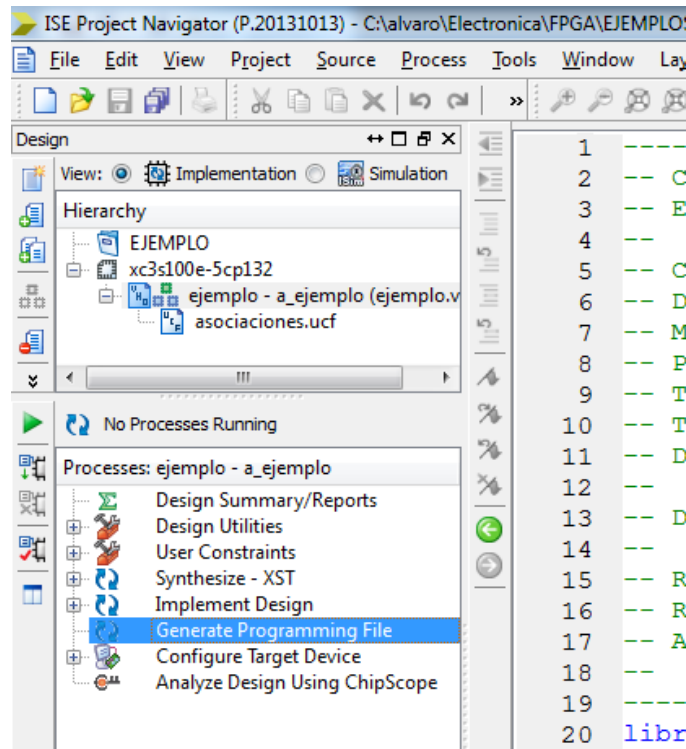
Síntesis del código y generación del archivo de “bitstream”

La síntesis sobre la FPGA consta de dos procesos:

1. Compilar el código VHDL y generar el archivo de configuración de la FPGA (archivo de “bitstream”).
2. Transferir el archivo de “bitstream” hacia la FPGA.

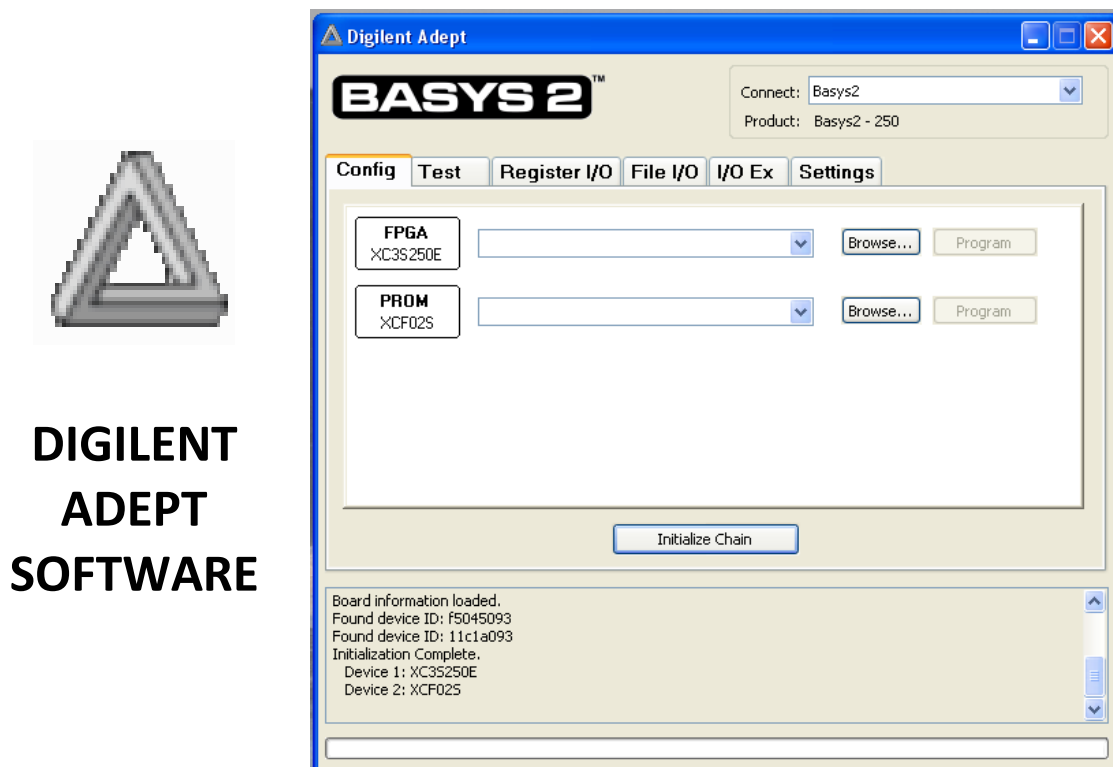
1. Compilación del código

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



2. Transferencia del “bitstream” a la FPGA

Una vez compilado el código VHDL, se genera un archivo con extensión .bit que contiene la secuencia “bitstream”, la cual permite configurar internamente la FPGA. Esta secuencia deberá ser transferida a la tarjeta BASYS2. Para ello necesita el programa ADEPT de DIGILENT disponible en todos los ordenadores del laboratorio. El icono del programa, así como la pantalla de inicio al iniciarlo, aparecen a continuación en la figura siguiente:



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 quedará configurada según el diseño descrito en el código VHDL y podrá interaccionar con los diferentes recursos presentes en la tarjeta BASYS2. Estos cambios serán permanentes mientras no se desconecte la alimentación de la FPGA.

Para sintetizar un nuevo código, basta con volver a descargar un nuevo archivo de “bitstream” sobre la FPGA.

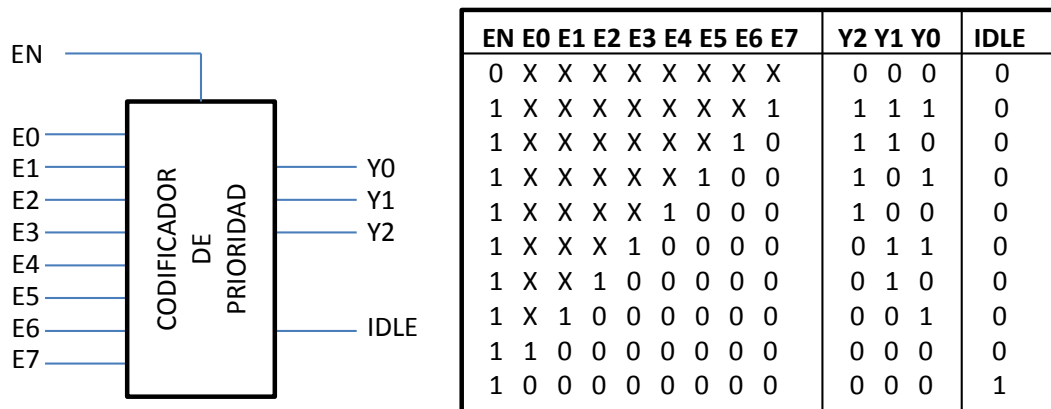
Diseño de circuitos combinacionales con VHDL

En un circuito combinacional, las salidas dependen exclusivamente del valor de las entradas. Este tipo de circuitos puede describirse de dos formas en VHDL:

1. **Sin emplear PROCESS**, cableando la funcionalidad de cada bloque mediante sentencias de asignación o de tipo WHEN o SELECT.
2. **Empleando PROCESS**, describiendo la funcionalidad del bloque mediante sentencias de tipo IF y CASE. En este caso es obligatorio seguir las siguientes reglas:
 - Todas las entradas deben incluirse necesariamente en la lista de sensibilidad del proceso.
 - Al inicio del process debe asignarse necesariamente un valor a todas las salidas.

Ejemplo de síntesis combinacional, codificador de prioridad de 8 bits:

Se muestra a continuación la tabla de verdad y la síntesis VHDL de un codificador de prioridad con ENABLE (EN), 8 entradas (E0..E7) activas a nivel alto, 3 salidas (Y0..Y2) que indican la entrada activa y una salida IDLE que se activa (a nivel alto) cuando ninguna entrada está activa. Cuando EN=0 todas las salidas están a 0.



FORMA 1: Mediante lógica sin emplear PROCESS

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

```
entity CODIF_PRIOR is
  port (
    E    : in  STD_LOGIC_VECTOR (7 downto 0); -- Entradas
    EN   : in  STD_LOGIC;                   -- ENABLE
    Y     : out STD_LOGIC_VECTOR (2 downto 0); -- Salidas
    IDLE : out STD_LOGIC;                   -- Salida IDLE
  );
end CODIF_PRIOR;

architecture a_CODIF_PRIOR of CODIF_PRIOR is
begin

  IDLE<= NOT E(7) AND NOT E(6) AND NOT E(5) AND NOT E(4) AND NOT E(3)
        AND NOT E(2) AND NOT E(1) AND NOT E(0) AND EN;

  Y<= "000" when EN='0' else -- Vamos comprobando las condiciones por orden
    "111" when E(7)='1' else
```

```

        "110" when E(6)='1' else
        "101" when E(5)='1' else
        "100" when E(4)='1' else
        "011" when E(3)='1' else
        "010" when E(2)='1' else
        "001" when E(1)='1' else
        "000" when E(0)='1' else
        "000";

end a_CODIF_PRIOR;

```

FORMA 2: Con un PROCESS

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CODIF_PRIOR is
    port (
        E    : in    STD_LOGIC_VECTOR (7 downto 0); -- Entradas
        EN    : in    STD_LOGIC_VECTOR;             -- ENABLE
        Y     : out   STD_LOGIC_VECTOR (2 downto 0); -- Salidas
        IDLE  : out   STD_LOGIC                     -- Salida IDLE
    );
end CODIF_PRIOR;

architecture a_CODIF_PRIOR of CODIF_PRIOR is

begin

    process (E,EN) -- Todas las entradas en la lista de sensibilidad
    begin

        Y <= "000"; -- Inicialmente damos un valor a todas las salidas
        IDLE <= '0';

        if (EN=0) then -- Primeramente se comprueba el ENABLE
            Y <= "000";
            IDLE <= '0';
        elsif (E="00000000") then -- A partir de esta comparación EN=1
            IDLE <= '1';           -- Comprobamos las condiciones por orden
        elsif (E(7)='1') then
            Y <= "111";
        elsif (E(6)='1') then
            Y <= "110";
        elsif (E(5)='1') then
            Y <= "101";
        elsif (E(4)='1') then
            Y <= "100";
        elsif (E(3)='1') then
            Y <= "011";
        elsif (E(2)='1') then
            Y <= "010";
        elsif (E(1)='1') then
            Y <= "001";
        end if;

    end process;

end a_CODIF_PRIOR;

```

En ambos casos la codificación es muy similar. Recuerde que las sentencias “when” y “select” solamente pueden utilizarse fuera de un process. Igualmente, las sentencias “if” y “case” solamente pueden utilizarse dentro de un process.

Diseño de circuitos secuenciales síncronos con VHDL

En este documento nos centraremos en el diseño de circuitos secuenciales síncronos. Un circuito secuencial síncrono posee una única señal de reloj que comparten todos los flip-flops del sistema. Todos ellos son activos en el mismo flanco del reloj.

La sintaxis básica de un circuito secuencial síncrono emplea un PROCESS donde el reloj es la única señal que está en su lista de sensibilidad y el funcionamiento debe sincronizarse con el flanco de subida del reloj. El siguiente código muestra la plantilla básica de un circuito secuencial síncrono descrito en VHDL:

Plantilla genérica para el diseño de un circuito secuencial síncrono

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

entity sec_sinc is
  port (
    CLK : in  STD_LOGIC    -- Reloj del sistema
  );
end sec_sinc;

architecture a_sec_sinc of sec_sinc is
begin

process (CLK)  -- El reloj es la única señal en la lista de sensibilidad
begin
  if (CLK'event and CLK='1') then  -- Activo en flanco de subida

    AQUÍ DEBE ESCRIBIRSE EL CÓDIGO DEL MÓDULO

  end if;
end process;

end a_sec_sinc;
```

A continuación se muestra el código VHDL para un registro de desplazamiento de 8 bits con entrada serie y salida paralelo, un ENABLE que permite activar el dispositivo y un RESET síncrono que pone a 0 las salidas.

Registro de desplazamiento de 8 bits con entrada serie, salida paralelo y RESET síncrono

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

entity reg_desp is
  port (
    CLK : in  STD_LOGIC;           -- Reloj del sistema
    SIN : in  STD_LOGIC;           -- Entrada serie de datos
    EN  : in  STD_LOGIC;           -- Entrada de ENABLE
    RST : in  STD_LOGIC;           -- Entrada de RESET
    Q   : out STD_LOGIC_VECTOR (7 downto 0)); -- Salida paralelo del registro
end reg_desp;
```

```

architecture a_reg_desp of reg_desp is

    signal S_Q : STD_LOGIC_VECTOR (7 downto 0); -- Contenido del registro (8 bits)

begin

    process (CLK) -- El reloj es la única señal en la lista de sensibilidad
    begin
        if (CLK'event and CLK='1') then -- Activo en flanco de subida
            if (RST='1') then -- Si el RESET está activo
                s_Q="00000000"; -- ponemos la salida a 0
            elsif (EN='1') then -- En otro caso, si el ENABLE está activo
                s_Q<=s_Q(6 downto 0) & SIN; -- Desplazamos el registro metiendo la entrada serie
            end if; -- En otro caso no se hace nada, lo cual deja el
                -- registro inalterado
        end if;
    end process;

    Q <= s_Q; -- La salida del registro es el contenido s_Q

end a_reg_desp;

```

Como puede ver se define una señal `s_Q` que se emplea para contener los 8 bits del registro. Esta señal se asigna al valor 0 en el caso de que la señal `RST` esté activa. En otro caso, si la señal `EN` está activa entonces se desplaza el registro añadiendo la entrada serie en el bit 0. Por último, si ninguna de estas dos señales (`RST` y `EN`) están activas, entonces no se hace nada. Cuando en un process no se asigna ningún valor a una señal, dicha señal permanece inalterada para el próximo ciclo de reloj.

Por último, fuera del process, se cablea la salida `Q` con la señal `s_Q`. Esto hace que los bits de `s_Q` sean accesibles desde la salida.

Tutorial de uso de los recursos de la tarjeta BASYS2 a través de ejemplos

Ejemplo 1: Empleo de los LEDs y los conmutadores.

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

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

FICHERO PUERTAS.VHD (fichero de código VHDL)

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

entity ejemplo1 is
  port (
    CONMUT0 : in  STD_LOGIC;    -- Conmutador conectado a una entrada
    CONMUT1 : in  STD_LOGIC;    -- Conmutador conectado a una entrada
    LED0    : out STD_LOGIC;    -- LED conectado a una salida
    LED1    : out STD_LOGIC;    -- LED conectado a una salida
  );
end ejemplo1;

architecture a_ejemplo1 of ejemplo1 is
begin

  LED0<=CONMUT0 or CONMUT1;    -- Salida LED0 cableada a la salida de la función OR de los
                                dos conmutadores
  LED1<=CONMUT0 and CONMUT1;   -- Salida LED1 cableada a la salida de la función AND de
                                los dos conmutadores

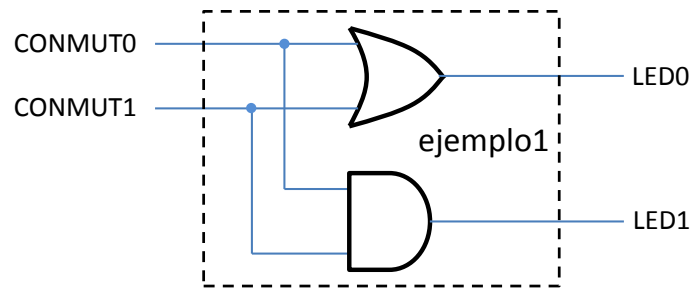
end a_ejemplo1;
```

FICHERO DE ASOCIACIONES: ASOCIACIONES.UCF

```
NET "CONMUT0" LOC = "P11";    # Conmutador 0
NET "CONMUT1" LOC = "L3";    # Conmutador 1
NET "LED0"    LOC = "M5";    # LED 0
NET "LED1"    LOC = "M11";   # LED 1
```

Con estos dos ficheros se puede ya compilar el código y obtener un “bit stream”. En el fichero VHDL declaramos una entidad llamada “ejemplo1” con dos entradas llamadas CONMUT0 y CONMUT1 y dos salidas llamadas LED0 y LED1.

En su descripción funcional, la salida LED0 se cablea como la función lógica OR de las dos entradas, mientras que la salida LED1 se cablea como la función lógica AND de las dos entradas. Es por tanto un circuito combinacional.



En el fichero de asociaciones indicamos cuáles son las patillas de la FPGA que se asocian físicamente con las entradas y las salidas de la entidad. Fíjese que se asocian exactamente con las patillas donde se encuentran conectados los recursos de la tarjeta BASYS 2 (dos conmutadores y dos LEDs).

Ejercicios propuestos sobre el ejemplo1:

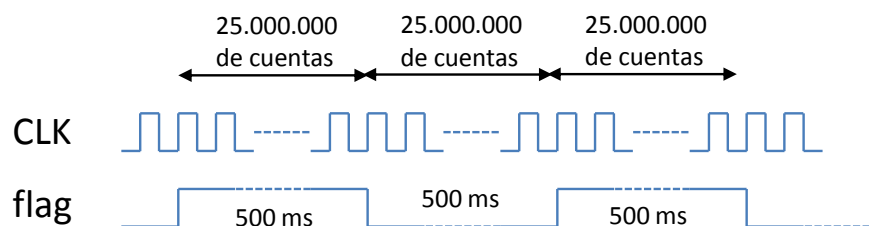
1. Modifique los archivos para que LED2 y LED3 muestren los valores complementarios de los leds LED0 y LED1 respectivamente manteniendo el resto de la funcionalidad del circuito.
2. Realice un nuevo diseño para que el LED0 muestre la paridad de los '1' presentes en los conmutadores SW0 a SW7 (8 entradas). Es decir, LED0 debe ser '1' cuando haya un número par de unos y '0' en caso contrario.
3. Realice un diseño nuevo para que cada pulsador (BTN0, BTN1, BTN2 y BTN3) encienda un LED (LED0, LED1, LED2 y LED3) solamente cuando el conmutador SW0 tiene un '0' lógico.

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 el LED0. Queremos que el LED parpadee con un periodo encendido/apagado de 500 ms cada uno.

Utilizaremos un proceso en cuya lista de sensibilidad se coloca la señal de reloj. De este modo el proceso se ejecutará cada vez que la señal de reloj cambie.

La estrategia consiste en incrementar un contador con cada flanco de subida del reloj. Puesto que el periodo de la señal de reloj es de 20ns (inverso de 50 MHz), cuando el contador llegue al valor 25.000.000 sabremos que habrán transcurrido 500 ms. En ese momento se conmuta el valor de una señal (flag) y se vuelve a poner el contador a 0. La señal flag estará 500 ms a nivel alto y 500 ms a nivel bajo. Tendrá por tanto un periodo de 1 s.



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

FICHERO DIV_RELOJ.VHD (archivo de código VHDL)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ejemplo2 is
    Port ( CLK : in  STD_LOGIC;    -- Reloj de entrada a la FPGA de 50 MHz
          SAL : out STD_LOGIC);    -- salida del circuito para conectar al LED
end ejemplo2;

architecture a_ejemplo2 of ejemplo2 is

    signal contador : unsigned (31 downto 0) := (others=>'0'); -- contador inicializado a 0
    signal flag : STD_LOGIC:='0'; -- señal para generar la salida

begin

    PROC_CONT : process (CLK) -- el proceso se dispara cada vez que el reloj cambia
    begin
        if CLK'event and CLK='1' then -- en cada flanco de subida
            contador<=contador + 1; -- se incrementa el contador
            if contador>=25000000 then -- el reloj tiene un periodo de 20 ns (50 MHz)
                -- tras 25.000.000 de cuentas habrán transcurrido 500 ms
                flag<=not flag; -- entonces conmutamos el valor de flag
                -- (flag tiene por tanto un periodo de 1 s)
                contador<=(others=>'0'); -- y ponemos el contador a 0
            end if;
        end if;
    end process;

    SAL<=flag; -- la señal flag se cablea con la salida

end a_ejemplo2;

```

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 archivos VHDL.

Ejercicios propuestos sobre el ejemplo 2:

1. Realice un diseño donde el LED se mantenga encendido durante 1s y apagado durante 1s.
2. Realice un diseño donde el LED se mantenga encendido durante 500ms y apagado durante 1s.
3. Realice un diseño donde se utilicen dos LEDs. Uno debe encenderse y apagarse con un intervalo de 500 ms y el otro debe encenderse y apagarse con un intervalo de 1s.

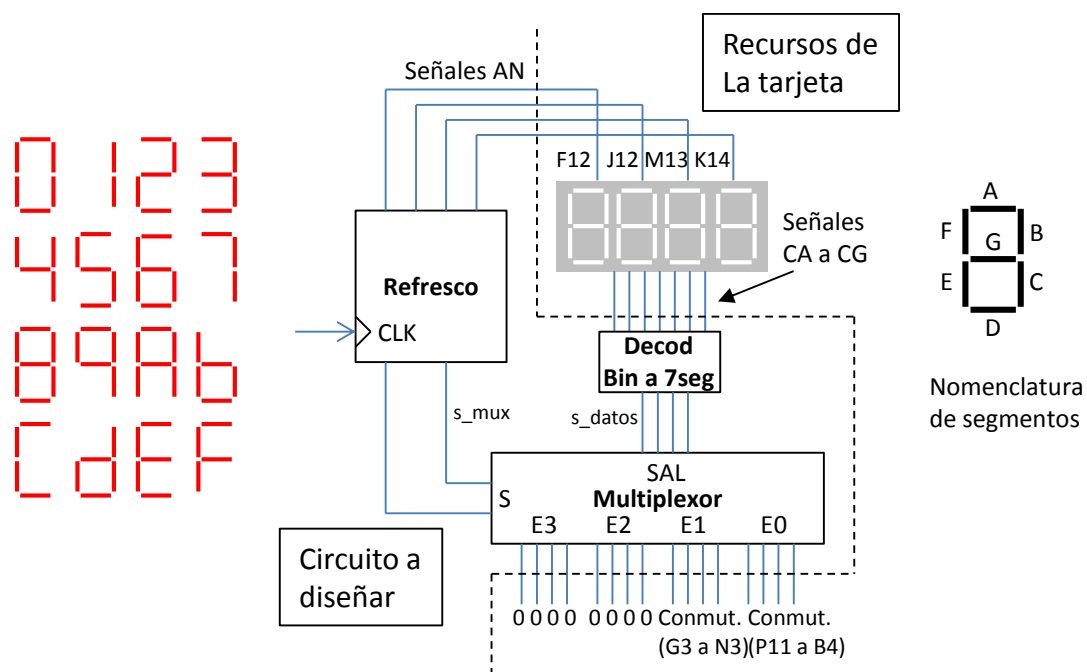
Ejemplo 3: Uso de los displays: Visualización de dos cifras hexadecimales diferentes en dos displays. Las cifras hexadecimales se introducen mediante los conmutadores.

En este caso queremos visualizar dos cifras hexadecimales diferentes en los dos displays de la derecha de la tarjeta BASYS 2. Las cifras (de 4 bits cada una) se introducirán mediante los conmutadores, siendo SW3 a SW0 los bits correspondientes a la cifra de la derecha y SW7 a SW4 los bits correspondientes a la cifra de la izquierda.

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 en cada display.

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 circuito de refresco 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. Los dos displays de la izquierda mostrarán siempre el valor 0.

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.NUMERIC_STD.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

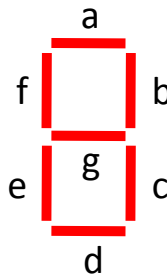
begin

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;

-- S contiene los segmentos (abcdefg)

-- Los segmentos se encienden con un '0'

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.NUMERIC_STD.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
          E2 : in  STD_LOGIC_VECTOR (3 downto 0); -- Entrada de datos 2
          E3 : in  STD_LOGIC_VECTOR (3 downto 0); -- Entrada de datos 3
          S : in  STD_LOGIC_VECTOR (1 downto 0); -- 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 "00",
        E1 when "01",
        E2 when "10",
        E3 when "11",
        E0 when others;
end a_MUX;

```

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

FICHERO refresco.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity refresco is
    Port ( CLK      : in  STD_LOGIC;          -- reloj de la FPGA
          AN        : out STD_LOGIC_VECTOR (3 downto 0); -- salidas para activar los
                                                         displays individualmente
          SCONTROL  : out STD_LOGIC_VECTOR (1 downto 0) -- señal para las entradas de
                                                         selección del mux.
    );
end refresco;

architecture a_refresco of refresco is

    signal contador : unsigned (31 downto 0):=(others=>'0'); -- divisor de frecuencia
    signal sel_mux   : unsigned (1 downto 0) :="00";         -- señal para controlar el mux

begin

    process (CLK) -- el proceso se dispara cada vez que cambia el reloj
    begin
        if CLK'event and CLK='1' then -- En cada flanco de subida
            contador<=contador + 1; -- Se incrementa el contador
            if contador>=50000 then -- 50.000 cuentas se corresponden con 1ms.
                sel_mux<=sel_mux+1; -- En ese caso incrementamos sel_mux que toma valores
                                     -- 00 01 10 y 11 cíclicamente
                contador<=(others=>'0'); -- y ponemos el contador a 0
            end if;
        end if;
    end process;

    SCONTROL<=STD_LOGIC_VECTOR(sel_mux); -- Se cablea SCONTROL con la señal sel_mux

    AN<="0111" when sel_mux="00" else -- La señal AN se cablea de modo que se activa
    "1011" when sel_mux="01" else -- cada display
    "1101" when sel_mux="10" else -- en función del valor de sel_mux
    "1110" when sel_mux="11"; -- Los displays se encienden con un '0'

end a_refresco;

```

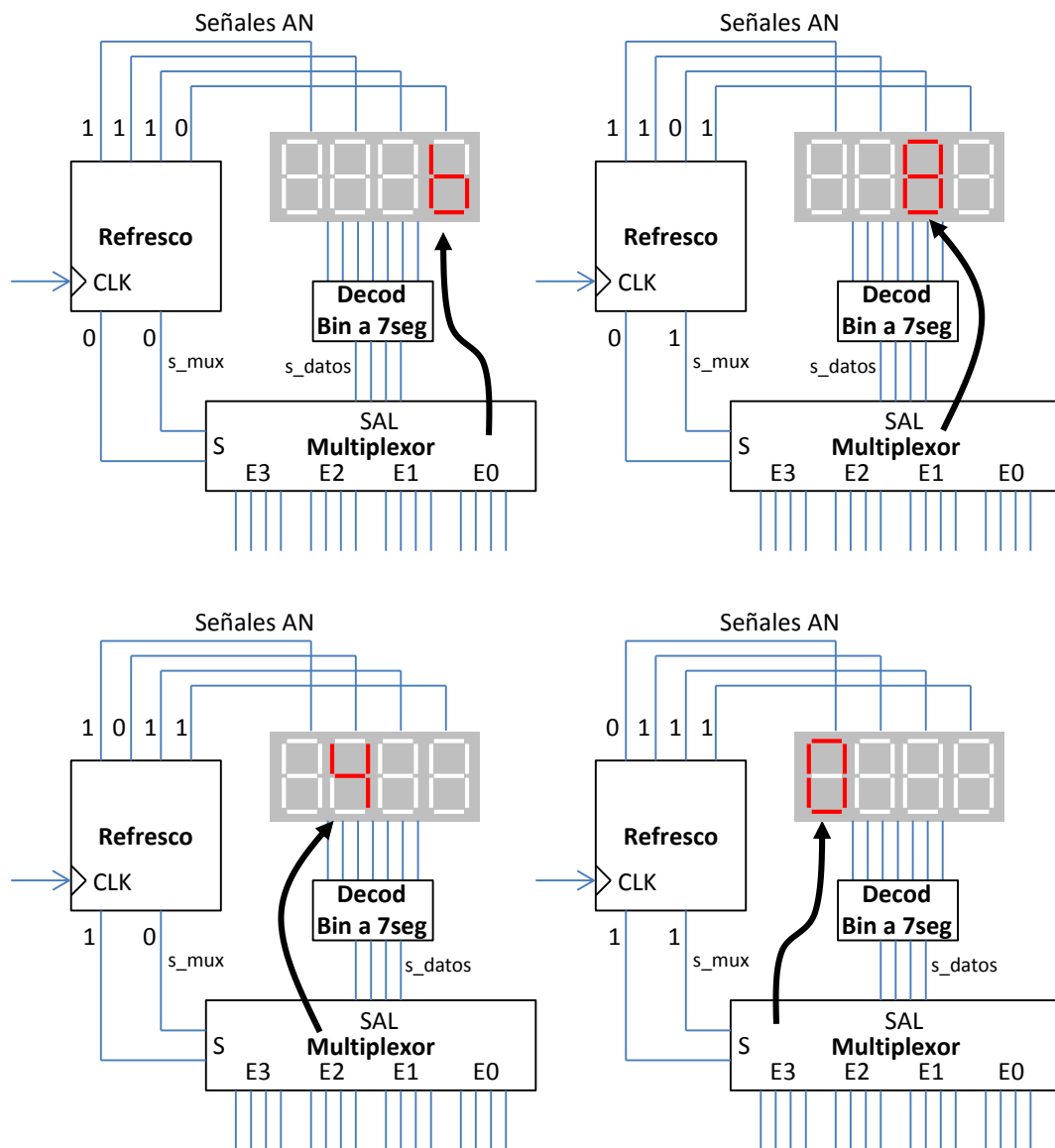
Este fichero describe un circuito secuencial de refresco. De forma similar al ejemplo 2, un proceso divide la frecuencia del reloj utilizando un contador. Cada 1 ms se suma 1 a la señal sel_mux (de 2 bits) de modo que esta señal va tomando cíclicamente los valores: 00, 01, 10 y 11. Dicha señal sel_mux está cableada con la salida SCONTROL.

En paralelo con este proceso la salida AN va tomando los valores 1110, 1101, 1011 y 0111 cíclicamente, en sincronía con los valores de sel_mux.

Con este código se describe por tanto un elemento que selecciona cada una de las entradas del multiplexor (salida SCONTROL) al mismo tiempo que activa el display correspondiente a

esta entrada (salida AN). De esta manera solo uno de los displays está activo cuando se selecciona una entrada del multiplexor. Al producirse la secuencia cada 1 ms, el ojo no es capaz de percibir el parpadeo, pudiendo de esta forma visualizar 4 cifras diferentes en los displays.

La figura siguiente muestra de manera gráfica el proceso de refresco.



El circuito de refresco alterna entre estas cuatro configuraciones cada 1 ms

FICHERO ejemplo3.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ejemplo3 is
    Port ( CLK      : in  STD_LOGIC;                -- reloj de la FPGA
          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 segmentos de
                                                    -- los displays
          AN        : out STD_LOGIC_VECTOR (3 downto 0) -- salidas para activar los
                                                    -- displays individualmente
    );
end ejemplo3;

architecture a_ejemplo3 of ejemplo3 is
    -- en este caso se trata de una
    -- descripción estructural

    component decod7s
        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);
              E2 : in  STD_LOGIC_VECTOR (3 downto 0);
              E3 : in  STD_LOGIC_VECTOR (3 downto 0);
              S : in  STD_LOGIC_VECTOR (1 downto 0);
              SAL : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

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

    -- Señales definidas para interconectar bloques

    signal s_mux      : STD_LOGIC_VECTOR (1 downto 0);
    signal s_datos    : STD_LOGIC_VECTOR (3 downto 0);

begin

    U1 : refresco
        port map (
            CLK => CLK,
            AN  => AN,
            SCONTROL => s_mux
        );

    U2 : MUX
        port map (
            E0 => CIFRA0,
            E1 => CIFRA1,
            E2 => "0000",
            E3 => "0000",
            S  => s_mux,
            SAL => s_datos
        );

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

end a_ejemplo3;

```

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 "AN<0>" LOC = "K14"; # SEÑAL DE ACTIVACIÓN DEL DISPLAY 0
NET "AN<1>" LOC = "M13"; # SEÑAL DE ACTIVACIÓN DEL DISPLAY 1
NET "AN<2>" LOC = "J12"; # SEÑAL DE ACTIVACIÓN DEL DISPLAY 2
NET "AN<3>" LOC = "F12"; # SEÑAL DE ACTIVACIÓN DEL DISPLAY 3
```

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.

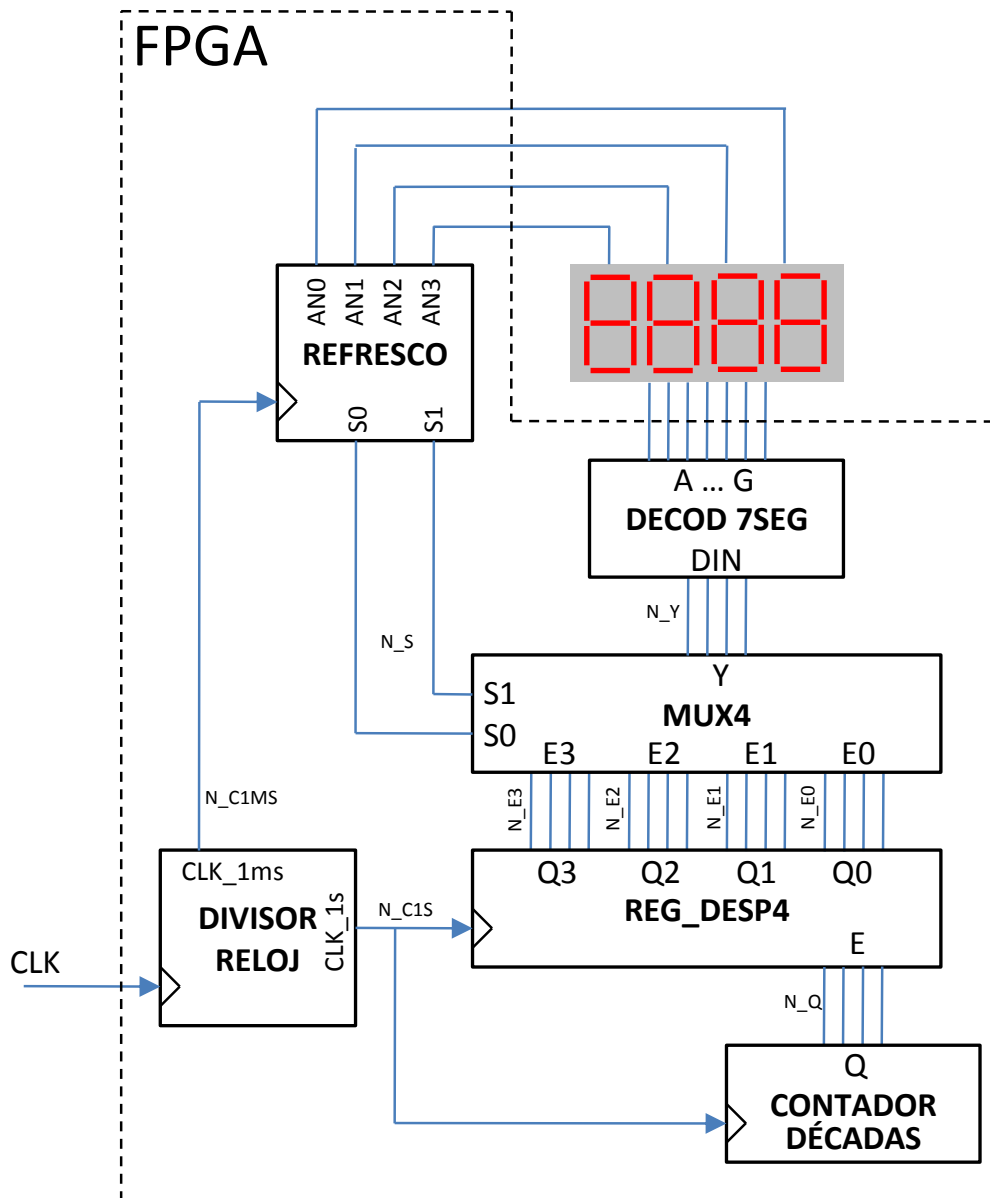
Ejercicios propuestos sobre el ejemplo 3:

1. 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 un valor de 0. Modifique el código para que los dos displays que siempre muestran 0 permanezcan apagados.
2. Modifique el código para que los displays se refresquen con un periodo de 1 s de modo que pueda ver cómo se actualizan secuencialmente.
3. Modifique el código para que las dos cifras hexadecimales aparezcan en los displays de la izquierda en lugar de aparecer en los de la derecha.
4. Modifique el código para que solamente se visualicen cifras decimales. Si se selecciona una cifra binaria superior a 9 en los interruptores, deberá aparecer una línea en el segmento central (segmento g) del display correspondiente.

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.

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.NUMERIC_STD.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 : unsigned (3 downto 0) := "0000"; -- 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<=STD_LOGIC_VECTOR (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;
use IEEE.NUMERIC_STD.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; -- cableado de las señales a 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;
use IEEE.NUMERIC_STD.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;
use IEEE.NUMERIC_STD.ALL;

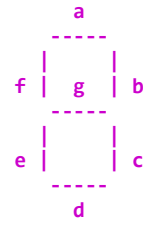
entity decod7s is
    port ( DIN   : in  STD_LOGIC_VECTOR (3 downto 0); -- entrada de datos
          S7SEG : out STD_LOGIC_VECTOR (0 to 6)); -- salidas 7seg (abcdefg)
end decod7s;

```

```

architecture a_decod7s of decod7s is
begin
    with DIN select S7SEG <= -- vector de entrada (abcdefg)
        "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; -- Los segmentos se encienden con un 0
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 refresco.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity refresco 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 refresco;

architecture a_refresco of refresco is

    signal SS : unsigned (1 downto 0); -- Señal que controla el MUX

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<=STD_LOGIC_VECTOR (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"; -- Los displays se activan con '0'

end a_refresco;

```

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.

FICHERO div_reloj.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

```

```

entity div_reloj is
    Port ( CLK : in  STD_LOGIC;      -- Entrada de reloj de la FPGA
          CLK_1S : out STD_LOGIC;    -- Salida reloj con periodo 1s
          CLK_1MS : out STD_LOGIC);  -- Salida reloj con periodo 1 ms
end div_reloj;

architecture a_div_reloj of div_reloj is

    signal div1 : unsigned (31 downto 0); -- contador para dividir el reloj
    signal div2 : unsigned (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 1 ms

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

            if div1>=25000000 then -- Cuando div1=25000000 han transcurrido 500ms
                S_1S<=not S_1S; -- se conmuta el valor de S_1S
                div1<=(others=>'0'); -- y se pone el contador a 0
            end if;

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

        end if;
    end process;

    CLK_1S <= S_1S;      -- Se cablean las señales a las dos salidas
    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;
use IEEE.NUMERIC_STD.ALL;

entity circuito is
    Port ( CLK : in  STD_LOGIC;      -- Reloj FPGA
          SEG7 : out STD_LOGIC_VECTOR (0 to 6); -- Señales para los segmentos
          AN : out STD_LOGIC_VECTOR (3 downto 0)); -- Señales de selección de display
end circuito;

architecture a_circuito of circuito is

    -- Señales para interconectar los módulos

    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;

-- Componentes

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

component reg_desp4
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 component;

component mux4
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 component;

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

component refresco
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 component;

component div_reloj
Port ( CLK : in  STD_LOGIC;           -- Entrada de reloj de la FPGA
      CLK_1S : out STD_LOGIC;         -- Salida reloj con periodo 1s
      CLK_1MS : out STD_LOGIC);       -- Salida reloj con periodo 1 ms
end component;

begin

-- Interconexiones de módulos

U1 : div_reloj
    port map (
        CLK=>CLK,
        CLK_1S=>N_C1S,
        CLK_1MS=>N_C1MS
    );

U2 : decod7s
    port map (
        DIN=>N_Y,
        S7SEG=>SEG7
    );

U3 : mux4
    port map (
        E0=>N_E0,
        E1=>N_E1,
        E2=>N_E2,
        E3=>N_E3,
        Y=>N_Y,
        S=>N_S
    );

```

```

U4 : reg_desp4
    port map (
        CLK=>N_C1S,
        E=>N_Q,
        Q0=>N_E0,
        Q1=>N_E1,
        Q2=>N_E2,
        Q3=>N_E3
    );

U5 : contador
    port map (
        CLK=>N_C1S,
        Q=>N_Q
    );

U6 : refresco
    port map (
        CLK=>N_C1MS,
        AN=>AN,
        S=>N_S
    );

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.

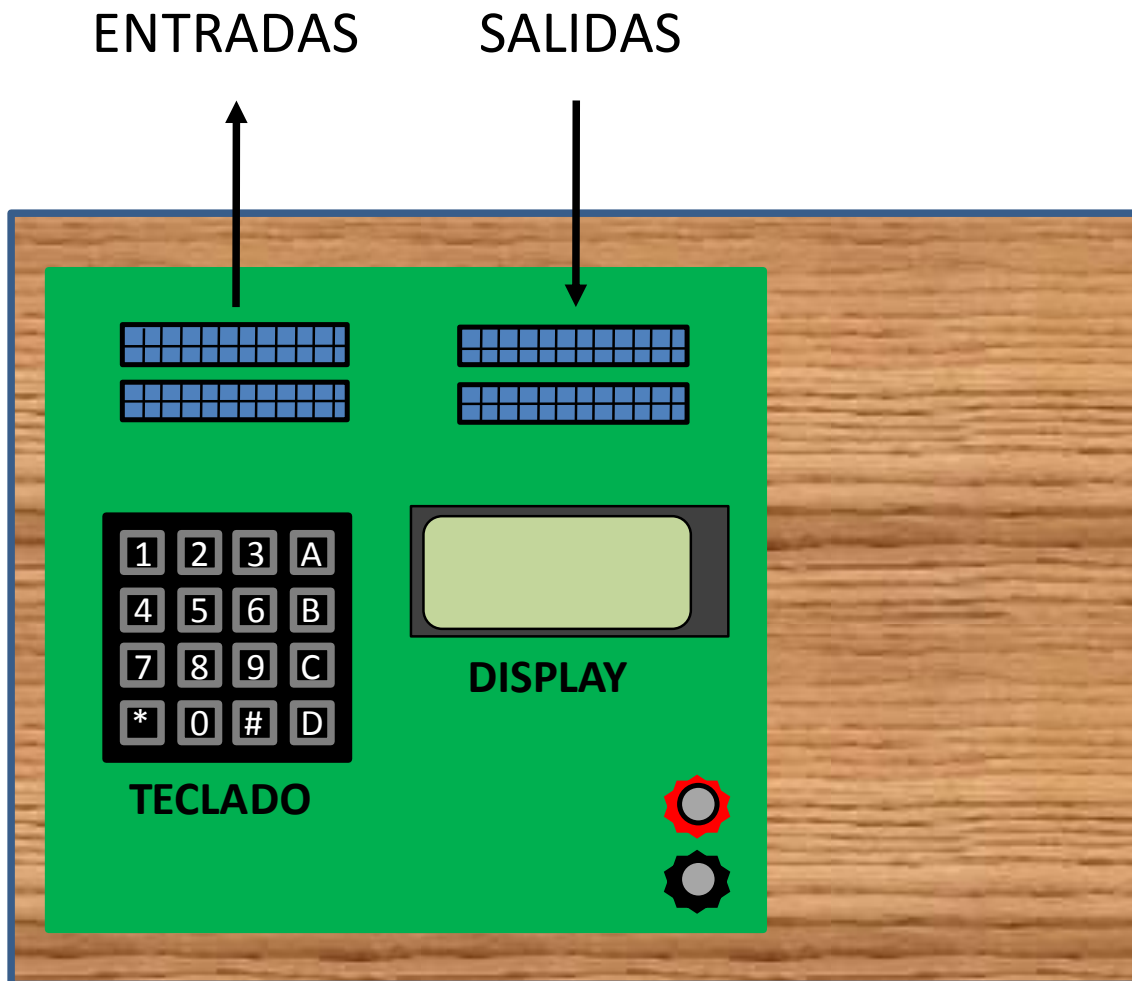
Ejercicios propuestos sobre el ejemplo 4:

1. Modifique el ejemplo para que las cifras aparezcan en la secuencia descendente: 0000, 0009, 0098, 0987, 9876, 8765, 7654, 6543, 5432, 4321, 3210, 2109, 1098, 0987, etc.
2. Modifique el ejemplo para que las cifras aparezcan por la izquierda en lugar de por la derecha según la secuencia: 0000, 1000, 2100, 3210, 4321, 5432, 6543, 7654, 8765, 9876, 0987, 1098, etc.
3. Modifique el ejemplo para que las cifras que aparecen sean hexadecimales y se desplacen hacia la izquierda según la secuencia: 0000, 0001, 0012, 0123, 1234, 2345, 3456, 4567, 5678, 6789, 789A, 89AB, 9ABC, ABCD, BCDE, CDEF, DEF0, EF01, F012, 0123, etc.
4. Modifique el ejercicio anterior para que las cifras se desplacen con un periodo de 500 ms en lugar de 1s
5. Modifique el ejemplo 3 para que las cifras se desplacen solamente cuando se mantenga pulsado el pulsador BTN0. (Para ello tendrá que añadir un ENABLE en el registro de desplazamiento y en el contador).

Entradas y salidas externas

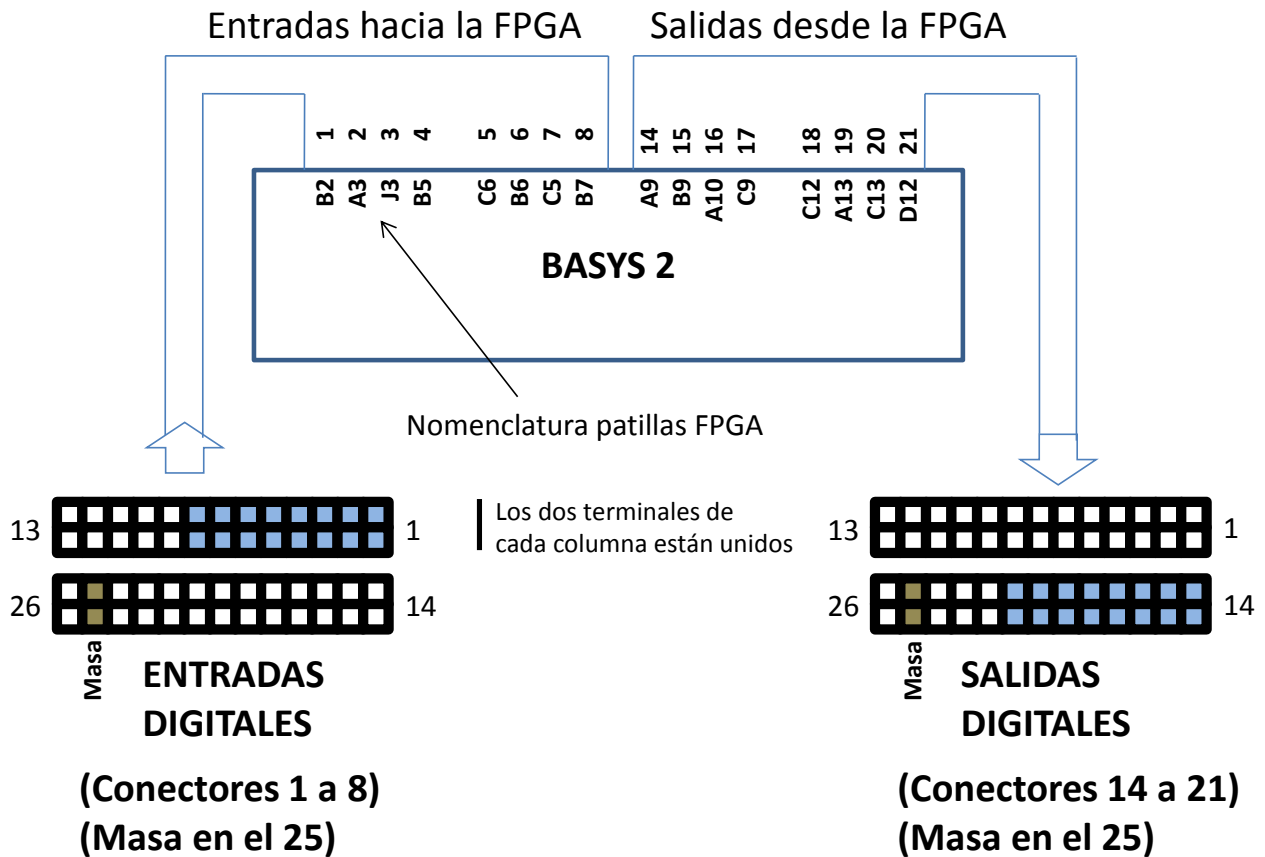
Además de los recursos conectados a algunas de las salidas de la FPGA, en la tarjeta BASYS2 existen también entradas y salidas externas que pueden utilizarse para conectar la tarjeta con circuitos exteriores. Las tensiones de las entradas y salidas son de lógica TTL, es decir: 5 V para el '1' y 0 V para el '0'

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.NUMERIC_STD.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 CLK_10k : STD_LOGIC := '0';           -- Señal de reloj dividida
    signal divisor : unsigned (15 downto 0) := "0000000000000000"; -- divisor de reloj
    signal QS : unsigned (3 downto 0) := "0000"; -- salida del contador

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
                CLK_10k <= not CLK_10k;        -- CLK_10k será de 10 KHz.
            end if;
        end if;
    end process;

    CONT : process (CLK_10k)                   -- proceso que implementa el contador
    begin
        if (CLK_10k'event and CLK_10k='1') then -- activo en flanco de subida
            QS<=QS+1;                          -- se suma 1 en cada flanco activo
        end if;
    end process;

    RCO<= QS(3) and QS(2) and QS(1) and QS(0); -- RCO es 1 cuando Q=15
    Q <= STD_LOGIC_VECTOR(QS);                 -- cableamos 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 (CLK_10k) 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

```

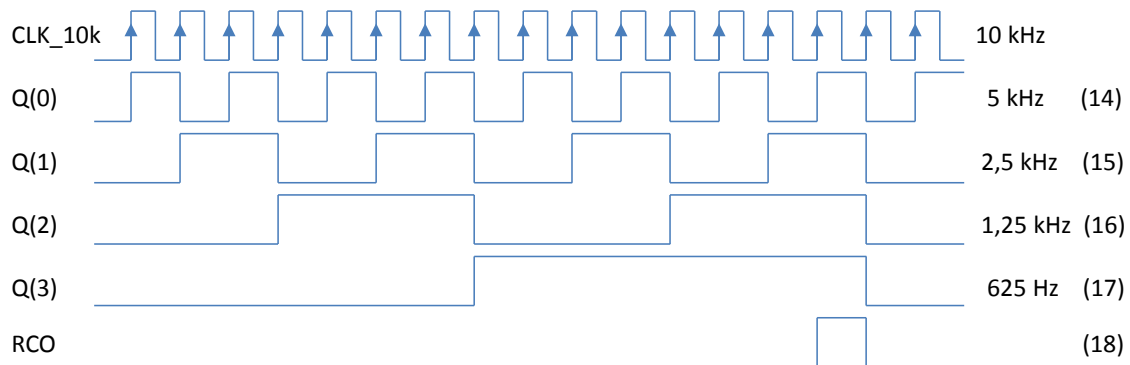
NET "CLK" LOC = "M6"; # RELOJ DE LA FPGA

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 en la salida Q(0). El resto de las señales de salida podrán medirse en los terminales 15, 16, 17 y 18.

Dado que el circuito es un contador binario, en las salidas Q podrán medirse señales cuadradas de las siguiente frecuencias:



Ejercicios propuestos sobre el ejemplo 5:

1. Modifique el ejemplo para que las señales aparezcan en las siguientes salidas:

Q0	Salida terminal 17
Q1	Salida terminal 18
Q2	Salida terminal 19
Q3	Salida terminal 20
RCO	Salida terminal 21

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 "CLK_10k". 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 "CLK_10k". 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.NUMERIC_STD.ALL;

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

architecture a_contador of contador is

    signal CLK_10k : STD_LOGIC := '0'; -- Señal de reloj dividida
    signal divisor : unsigned (15 downto 0) := "0000000000000000"; -- divisor de reloj
    signal QS : unsigned (3 downto 0) := "0000"; -- salida del contador
```

```

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
            CLK_10k<=not CLK_10k;    -- 50 us, luego la frec. de la señal
            divisor<=(others=>'0');   -- CLK_10k será de 10 KHz.
        end if;
    end if;
end process;

CONT : process (CLK_10k)             -- proceso que implementa el contador
begin
    if (CLK_10k'event and CLK_10k='1') then -- activo en flanco de subida
        QS<=QS+1;                    -- se suma 1 en cada flanco activo
    end if;
end process;

RCO <= QS(3) and QS(2) and QS(1) and QS(0); -- RCO es 1 cuando Q=15
Q <= STD_LOGIC_VECTOR(QS);             -- cableamos la señal QS a la salida real
TP1 <= CLK_10k;                        -- cableamos la señal CLK_10k a TP1 para medirla
end a_contador;

```

Modificación del fichero del ejemplo 5 para poder visualizar la señal interna “CLK_10k”. Las líneas resaltadas 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 “CLK_10k”. El resto de las señales podrán medirse en los terminales 14, 15, 16, 17 y 18.

Ejercicios propuestos sobre el ejemplo 6:

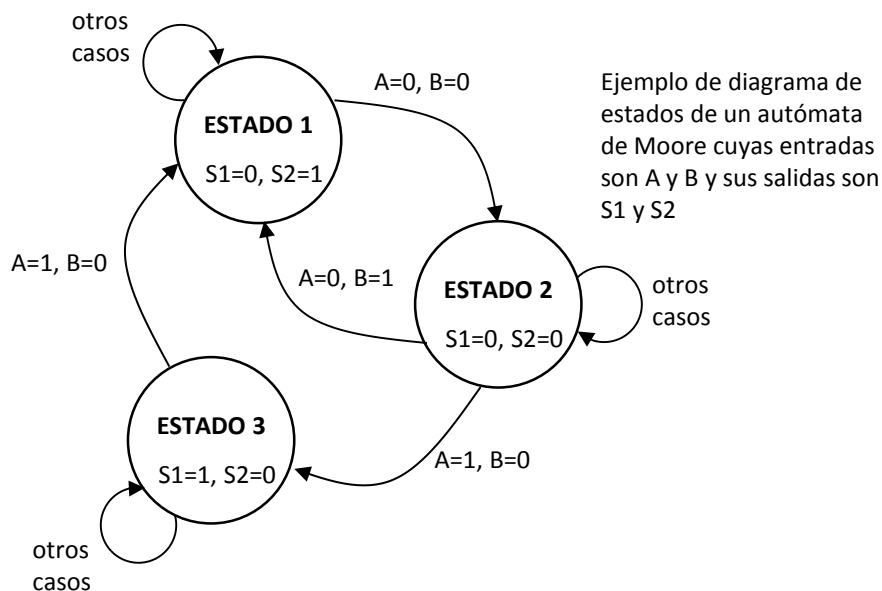
1. Añada dos salidas más donde se pueda medir la señal presente en los bits 7 y 9 de la señal divisor:

Diseño de autómatas de estados finitos

Los autómatas son circuitos secuenciales síncronos cuyo funcionamiento está descrito por una serie de estados. Las transiciones entre estos estados dependen de unas entradas externas. Además, el autómata proporciona unas salidas. Según esta definición, los autómatas pueden clasificarse en dos tipos:

- Autómatas de Moore, donde las salidas solamente dependen del estado en que se encuentra el autómata.
- Autómatas de Mealy, donde las salidas dependen tanto del estado en que se encuentra el autómata como de las entradas.

El funcionamiento de un autómata se esquematiza mediante lo que se conoce como diagrama de estados. Este diagrama muestra los diferentes estados del autómata, las transiciones entre los estados en función de las entradas, y las diferentes salidas que proporciona.



A efectos de simplificar la codificación, nos centraremos exclusivamente en el diseño y síntesis de autómatas de Moore donde emplearemos un único process para describir las transiciones entre los estados y una lógica combinacional para generar las salidas.

Ejemplo 7: Diseño de un autómata de Moore para el control del acceso a una puerta

Se pretende diseñar un autómata de Moore que permita el acceso a una puerta cuando se acciona el código correcto. En este caso el código se introducirá mediante los pulsadores BTN3 a BTN0 presentes en la tarjeta BASYS 2. Cuando se introduce la secuencia correcta, la apertura de puerta se indicará mediante uno de los LED presentes en la tarjeta, el cual deberá permanecer encendido durante 3s. Posteriormente el sistema vuelve al estado inicial.

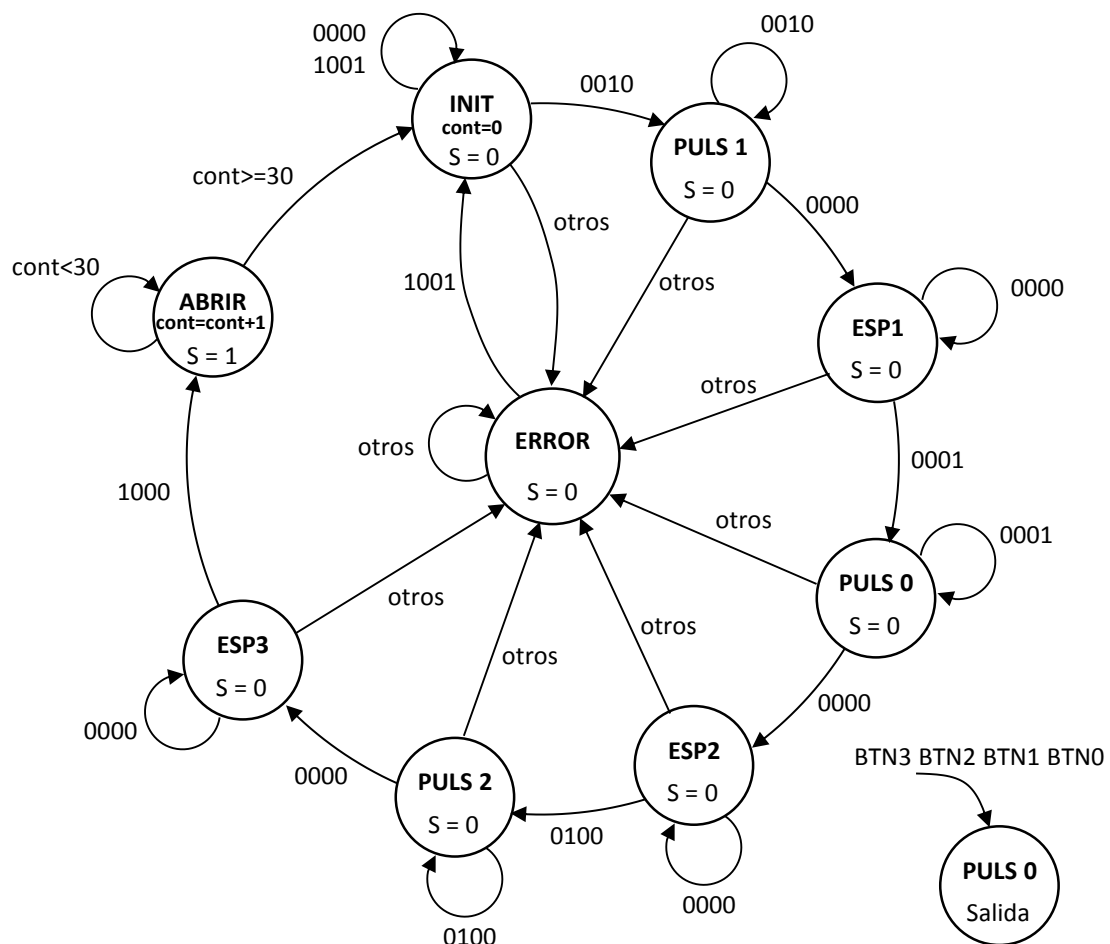
El código de apertura será el **1023**. Para introducirlo se pulsarán secuencialmente los pulsadores correspondientes al código correcto. Para borrar la secuencia introducida y volver al estado inicial se deberán de pulsar simultáneamente los pulsadores 0 y 3.

La secuencia debe ser introducida de forma seguida desde el estado inicial. Es decir, bien desde el inicio, o bien tras la pulsación de las teclas de borrado (0 y 3). Por ejemplo:

1023	La puerta se abre
4501023	La puerta no se abre
450B1023	(B indica borrado) La puerta se abre

Para evitar los posibles rebotes que aparecen en los pulsadores utilizaremos un reloj más lento de frecuencia 100 ms, que se obtendrá como división de frecuencia del reloj principal de 50 MHz mediante un proceso similar al del ejemplo 2 de esta guía. Para establecer la temporización de 3s necesaria para abrir la puerta se utilizará un contador que cuente hasta el valor 30 dentro de un estado. Puesto que cada cuenta se realiza en un ciclo de reloj, al salir del estado habrán transcurrido $30 \times 100 \text{ ms} = 3 \text{ s}$.

El diagrama de estados correspondiente a este autómata se muestra a continuación en la figura siguiente:



La descripción de los estados se indica a continuación:

INIT: Estado inicial. Este es el único estado desde el que puede comenzar la secuencia correcta. Por tanto, se permanece en este estado mientras no se pulsa ninguna tecla, o se pulsan las teclas de borrado. Si se pulsa la tecla 1 se pasa a PULS1, donde comienza la secuencia, y en otro caso se pasa al estado ERROR.

PULS1: La pulsación de la tecla 1 dura varios ciclos de reloj, por lo tanto debemos esperar a que el usuario suelte la tecla. El autómata permanece en este estado mientras la tecla 1 esté pulsada. Si se pulsa cualquier otra tecla se va al estado de ERROR.

ESP1: La tecla ya ha sido soltada, pero el usuario tarda varios ciclos de reloj en volver a pulsar una nueva tecla. El autómata permanece en este estado mientras no se pulsa ninguna tecla. Solamente si la siguiente tecla pulsada es 0 se avanza al PULS0, en otro caso se va al estado de ERROR.

PULS0: La pulsación de la tecla 0 dura varios ciclos de reloj, por lo tanto debemos esperar a que el usuario suelte la tecla. El autómata permanece en este estado mientras la tecla 0 esté pulsada. Si se pulsa cualquier otra tecla se va al estado de ERROR.

ESP2: La tecla ya ha sido soltada, pero el usuario tarda varios ciclos de reloj en volver a pulsar una nueva tecla. El autómata permanece en este estado mientras no se pulsa ninguna tecla. Solamente si la siguiente tecla pulsada es 2 se avanza al PULS2, en otro caso se va al estado de ERROR.

PULS2: La pulsación de la tecla 2 dura varios ciclos de reloj, por lo tanto debemos esperar a que el usuario suelte la tecla. El autómata permanece en este estado mientras la tecla 2 esté pulsada. Si se pulsa cualquier otra tecla se va al estado de ERROR.

ESP3: La tecla ya ha sido soltada, pero el usuario tarda varios ciclos de reloj en volver a pulsar una nueva tecla. El autómata permanece en este estado mientras no se pulsa ninguna tecla. Solamente si la siguiente tecla pulsada es 3 se avanza al estado ABRIR, en otro caso se va al estado de ERROR.

ABRIR: La secuencia ha sido correcta. En este estado se mantiene la salida a '1' y se incrementa el contador. Mientras dicho contador no alcance el valor 30 (3 s), el autómata se queda en el mismo estado. Cuando lo alcanza se vuelve al estado inicial.

ERROR: En algún momento se ha introducido una tecla errónea. Por tanto se permanece en este estado hasta que se pulsan las teclas de borrado para volver al estado INIT.

FICHERO codigo_puerta.vhd

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

entity codigo_puerta is
    Port ( CLK : in  STD_LOGIC;    -- entrada de reloj de la FPGA
          PULS : in  STD_LOGIC_VECTOR (3 downto 0); -- Pulsadores de entrada
          S : out STD_LOGIC);      -- Salida de puerta abierta
end codigo_puerta;
```

```

architecture a_codigo_puerta of codigo_puerta is

type STATE_TYPE is (INIT,PULS1,ESP1,PULS0,ESP2,PULS2,ESP3,ABRIR,ERROR);

signal divisor : unsigned (31 downto 0):=(others=>'0'); -- contador para dividir la
                                                         -- frecuencia del reloj
signal CLK_100ms : STD_LOGIC :='0';                    -- reloj de frecuencia dividida
signal ST : STATE_TYPE := INIT;                        -- estado del autómata
signal cont : unsigned (7 downto 0):=(others=>'0');     -- contador del tiempo de
                                                         -- puerta abierta

begin

-- Proceso para dividir la frecuencia del reloj

DIV : process (CLK) -- circuito síncrono. Solamente CLK en la lista de sensibilidad
begin
    if (CLK'event and CLK='1') then -- activo en flanco de subida
        divisor<=divisor+1;         -- incrementamos el divisor
        if (divisor>=2500000) then -- han transcurrido 50 ms
            divisor<=(others=>'0'); -- ponemos el divisor a 0
            CLK_100ms<=not CLK_100ms; -- e invertimos el valor de CLK_100ms
        end if;
    end if;
end process;

-- Proceso que implementa el autómata

AUT : process (CLK_100ms) -- circuito síncrono. Funciona con el reloj de 100 ms
begin
    if (CLK_100ms'event and CLK_100ms='1') then -- activo en flanco de subida
        case ST is
            when INIT => -- estado INIT
                cont<=(others=>'0'); -- poner a 0 el contador
                if (PULS="0010") then -- Si pulsamos 1
                    ST<=PULS1; -- vamos al estado PULS1
                elsif (PULS="0000" or PULS="1001") then --Si no pulsamos o BORRAR
                    ST<=INIT; -- se queda en el estado INIT
                else -- en otro caso
                    ST<=ERROR; -- Se va a ERROR
                end if;

            when PULS1 => -- codificación de cada estado
                if (PULS="0010") then -- siguiendo el diagrama de estados ...
                    ST<=PULS1;
                elsif (PULS="0000") then
                    ST<=ESP1;
                else
                    ST<=ERROR;
                end if;

            when ESP1 =>
                if (PULS="0000") then
                    ST<=ESP1;
                elsif (PULS="0001") then
                    ST<=PULS0;
                else
                    ST<=ERROR;
                end if;

            when PULS0 =>
                if (PULS="0001") then
                    ST<=PULS0;
                elsif (PULS="0000") then
                    ST<=ESP2;
                else

```

```

        ST<=ERROR;
    end if;
when ESP2 =>
    if (PULS="0000") then
        ST<=ESP2;
    elsif (PULS="0100") then
        ST<=PULS2;
    else
        ST<=ERROR;
    end if;

when PULS2 =>
    if (PULS="0100") then
        ST<=PULS2;
    elsif (PULS="0000") then
        ST<=ESP3;
    else
        ST<=ERROR;
    end if;

when ESP3 =>
    if (PULS="0000") then
        ST<=ESP3;
    elsif (PULS="1000") then
        ST<=ABRIR;
    else
        ST<=ERROR;
    end if;

when ABRIR =>
    cont<=cont+1;
    if (cont>=30) then
        ST<=INIT;
    else
        ST<=ABRIR;
    end if;

when ERROR =>
    if (PULS="1001") then
        ST<=INIT;
    else
        ST<=ERROR;
    end if;
end case;

end if;
end process;

-- Asignación de las salidas en función del estado (Moore)

S <= '1' when ST=ABRIR else '0'; -- La salida S solamente es 1 en el estado ABRIR

end a_codigo_puerta;

```

FICHERO DE ASOCIACIONES: asociaciones.ucf

```

NET "CLK" LOC = "M6";      # Reloj de la FPGA

NET "PULS<0>" LOC = "G12";  # Pulsador 0
NET "PULS<1>" LOC = "C11";  # Pulsador 1
NET "PULS<2>" LOC = "M4";   # Pulsador 2
NET "PULS<3>" LOC = "A7";   # Pulsador 3

NET "S" LOC = "M5";        # LED 0

```

Autómatas cuyos estados duren más de un ciclo de reloj

En ocasiones se necesitan autómatas cuyos estados duren más de un ciclo de reloj. En este caso se debe incrementar un contador en el interior del estado. Cuando el contador llega al final de cuenta se cambia de estado y se pone a 0 el contador de nuevo.

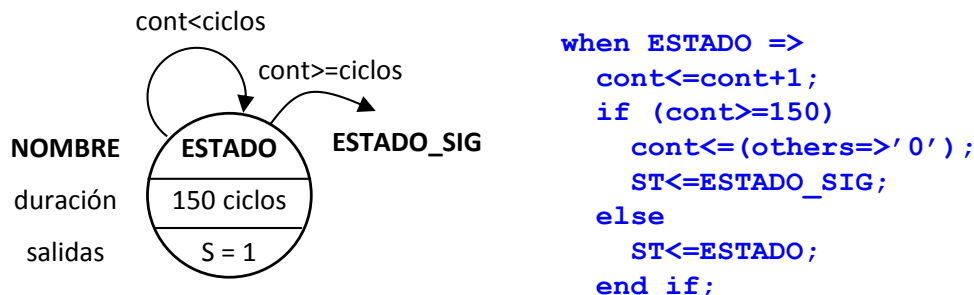
Ejemplo 8: Autómata para generación de una señal de emergencia SOS en Morse

En este ejemplo vamos a describir un autómata que permite generar una secuencia SOS en Morse que aparecerá en uno de los LED de la tarjeta BASYS2. El mensaje SOS se codifica en Morse como:

S O S
 . . . - - - . . .

El mensaje debe estar transmitiéndose constantemente. Cada punto debe durar 100 ms, y cada raya debe durar 300 ms. Las pausas entre cada símbolo (puntos o rayas) debe ser de 150 ms y el espacio entre letras (S – O – S) debe ser de 450 ms.

El autómata estará controlado por la señal de reloj de la FPGA (50 MHz). Por lo tanto necesitaremos codificar estados que duren varios ciclos de reloj. Para ello vamos a utilizar un contador que indique el número de ciclos de reloj transcurridos. La representación gráfica de estos estados, así como su codificación en VHDL se muestran a continuación:



Para establecer el número de ciclos de reloj necesarios en cada estado debemos tener en cuenta que el periodo del reloj de 50 MHz es de 20 ns. Por lo tanto:

EVENTO	DURACIÓN	No. DE CICLOS
PUNTO	100 ms	100 ms / 20 ns = 5.000.000
RAYA	300 ms	300 ms / 20 ns = 15.000.000
PAUSA	150 ms	150 ms / 20 ns = 7.500.000
ESPACIO	450 ms	450 ms / 20 ns = 22.500.000

El diagrama de estados del autómata completo responde al esquema que aparece a continuación en la página siguiente. En cada estado se indica el número de ciclos de reloj que debe durar y la salida que se debe proporcionar al LED (1 indica encendido):



FICHERO morse_SOS.vhd

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

entity morse_SOS is
    Port ( CLK : in  STD_LOGIC;    -- reloj del sistema de 50 MHz
          S   : out STD_LOGIC);    -- Salida para el LED
end morse_SOS;

architecture a_morse_SOS of morse_SOS is

    type STATE_TYPE is
        (PUNTO1, PAUSA1, PUNTO2, PAUSA2, PUNTO3, ESPACIO1, RAYA1, PAUSA4, RAYA2, PAUSA5, RAYA3, ESPACIO2);

    signal ST : STATE_TYPE := PUNTO1;           -- estado del autómata
    signal cont : unsigned (31 downto 0):=(others=>'0'); -- contador de ciclos en estado

begin
    process (CLK)    -- circuito síncrono.
    begin
        if (CLK'event and CLK='1') then -- activo en flanco de subida
            case ST is
                when PUNTO1=>
                    cont<=cont+1;           -- incrementamos el contador
                    if (cont>=5000000) then -- si llega al número de ciclos
                        cont<=(others=>'0'); -- poner a 0 el contador
                        ST<=PAUSA1;         -- y pasar al siguiente estado
                    else
                        ST<=PUNTO1;         -- en otro caso se queda en el mismo estado
                    end if;

                when PAUSA1=>
                    cont<=cont+1;           -- incrementamos el contador
                    if (cont>=7500000) then -- si llega al número de ciclos
                        cont<=(others=>'0'); -- poner a 0 el contador
                        ST<=PUNTO2;         -- y pasar al siguiente estado
                    else
                        ST<=PAUSA1;         -- en otro caso se queda en el mismo estado
                    end if;

                when PUNTO2=>
                    cont<=cont+1;           -- incrementamos el contador
                    if (cont>=5000000) then -- si llega al número de ciclos
                        cont<=(others=>'0'); -- poner a 0 el contador
                        ST<=PAUSA2;         -- y pasar al siguiente estado
                    else
                        ST<=PUNTO2;         -- en otro caso se queda en el mismo estado
                    end if;
            end case;
        end if;
    end process;
end a_morse_SOS;
```

```

end if;
when PAUSA2=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=7500000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=PUNTO3;          -- y pasar al siguiente estado
  else
    ST<=PAUSA2;          -- en otro caso se queda en el mismo estado
  end if;

when PUNTO3=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=5000000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=ESPACIO1;        -- y pasar al siguiente estado
  else
    ST<=PUNTO3;          -- en otro caso se queda en el mismo estado
  end if;

when ESPACIO1=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=22500000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=RAYA1;           -- y pasar al siguiente estado
  else
    ST<=ESPACIO1;        -- en otro caso se queda en el mismo estado
  end if;

when RAYA1=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=15000000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=PAUSA4;          -- y pasar al siguiente estado
  else
    ST<=RAYA1;           -- en otro caso se queda en el mismo estado
  end if;

when PAUSA4=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=7500000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=RAYA2;           -- y pasar al siguiente estado
  else
    ST<=PAUSA4;          -- en otro caso se queda en el mismo estado
  end if;

when RAYA2=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=15000000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=PAUSA5;          -- y pasar al siguiente estado
  else
    ST<=RAYA2;           -- en otro caso se queda en el mismo estado
  end if;

when PAUSA5=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=7500000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=RAYA3;           -- y pasar al siguiente estado
  else
    ST<=PAUSA5;          -- en otro caso se queda en el mismo estado
  end if;

```

```

when RAYA3=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=15000000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=ESPACIO2;        -- y pasar al siguiente estado
  else
    ST<=RAYA3;          -- en otro caso se queda en el mismo estado
  end if;

when ESPACIO2=>
  cont<=cont+1;          -- incrementamos el contador
  if (cont>=22500000) then -- si llega al número de ciclos
    cont<=(others=>'0');  -- poner a 0 el contador
    ST<=PUNTO1;          -- y pasar al siguiente estado
  else
    ST<=ESPACIO2;        -- en otro caso se queda en el mismo estado
  end if;

end case;
end if;
end process;

-- Asignación de salidas en función del estado (Moore)

S <= '1' when (ST=PUNTO1 or ST=PUNTO2 or ST=PUNTO3 or
               ST=RAYA1 or ST=RAYA2 or ST=RAYA3)
        else '0';

end a_morse_SOS;

```

FICHERO DE ASOCIACIONES: asociaciones.ucf

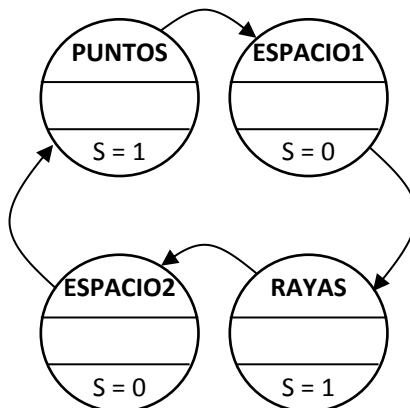
```

NET "CLK" LOC = "M6";      # Reloj de la FPGA
NET "S"   LOC = "M5";      # LED 0

```

Ejercicios propuestos sobre el ejemplo 8:

1. Modifique el ejemplo de modo que se reduzca el número de estados añadiendo otros contadores. Los nuevos contadores deberán tener en cuenta el número de puntos/rayas que se transmiten y también las pausas que deben transmitirse entre ellos. El diagrama diseñado deberá tener la siguiente estructura:



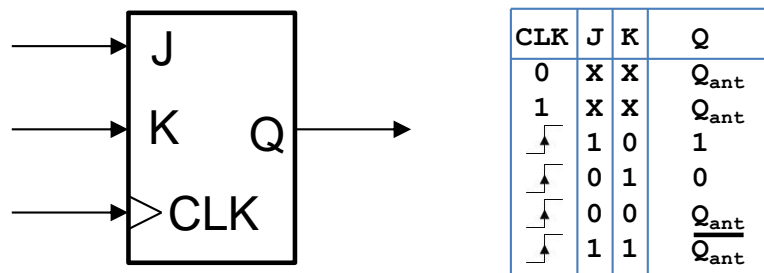
Simulación de circuitos mediante ISIM

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.

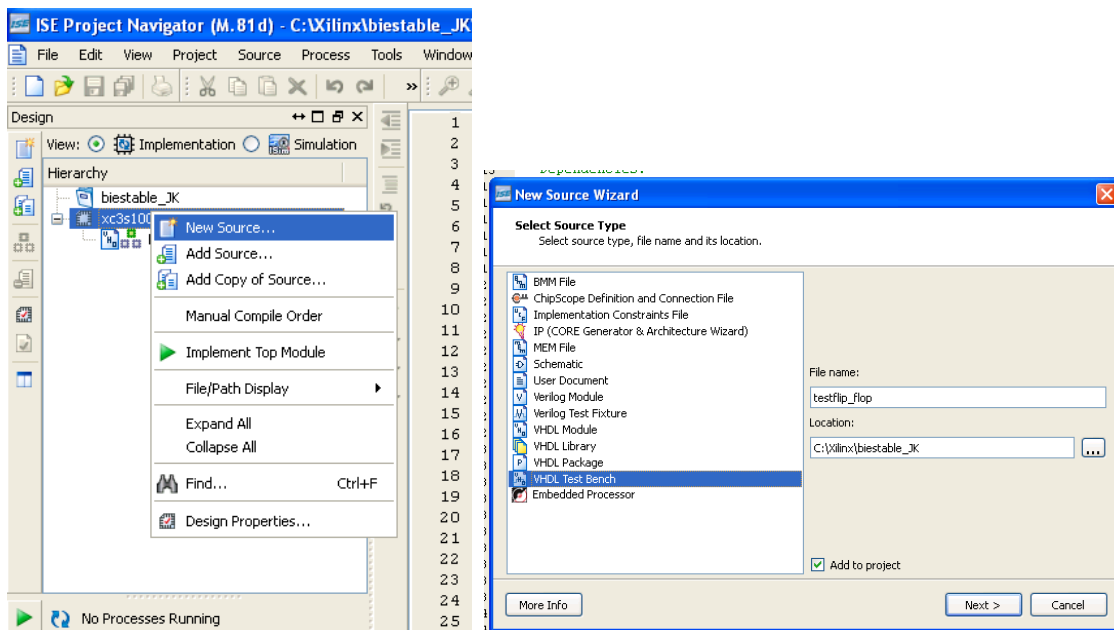
La simulación se lleva a cabo creando un nuevo fichero de test ("*test bench*") donde tendremos que introducir una secuencia lógica dependiente del tiempo en las variables de entrada. Como resultado, el simulador nos representará de forma gráfica los valores de las salidas en función de las citadas entradas.

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

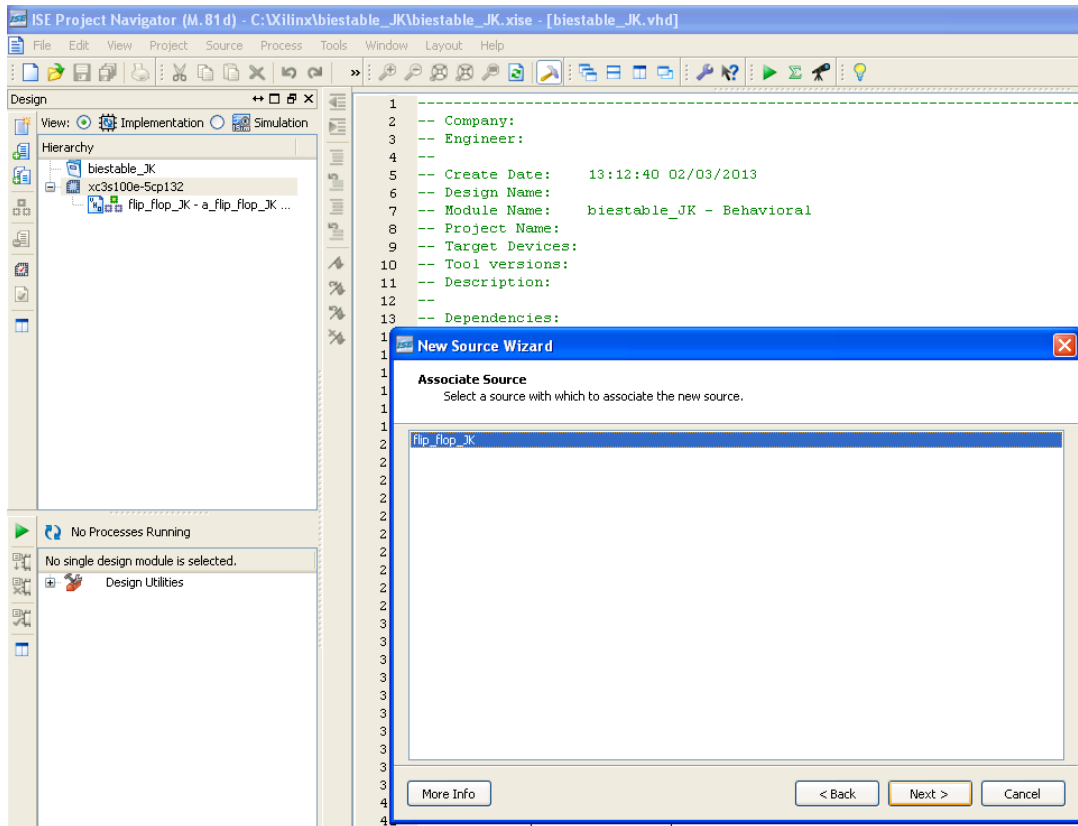
Para ilustrar el funcionamiento del simulador ISIM utilizaremos el código VHDL correspondiente a un flip flop JK. Este circuito biestable responde a la siguiente tabla de verdad:



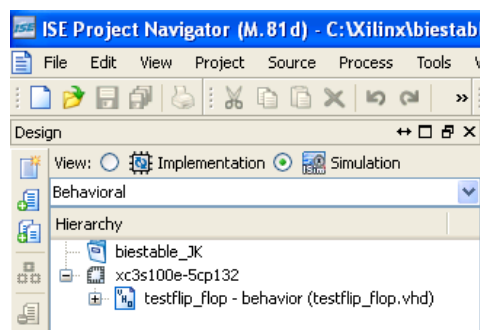
Para añadir al diseño del flip flop el fichero de test haga click con el botón derecho sobre el texto "xc3s100e-5cp132" y elija la opción "New Source". Seleccione en este caso "VHDL test bench" para el tipo e introduzca un nombre para el archivo (por ejemplo "test_flip_flop").



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).

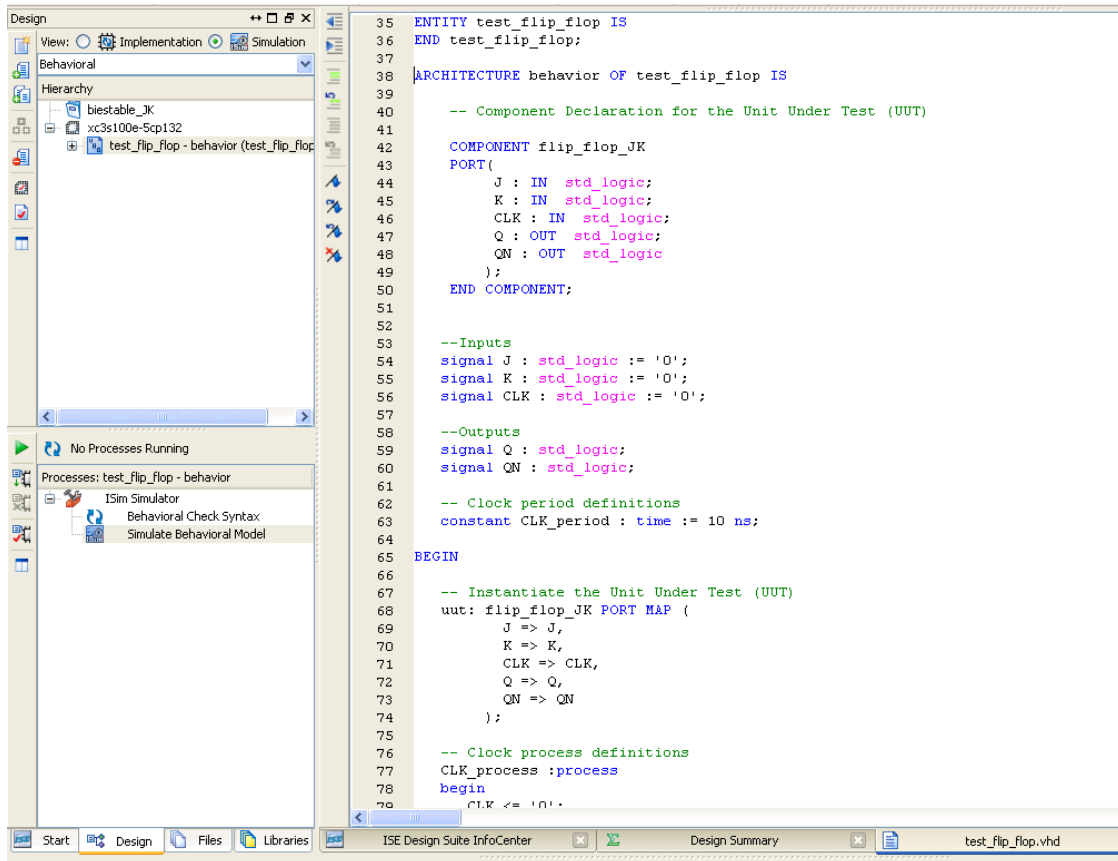


Después de hacer click en "Next" y "Finish" el fichero se añadirá al proyecto, aunque aún no aparece. Para ver su contenido deberá pasar del modo "implementation" al modo "Simulation" seleccionando la opción en la parte superior de la ventana que contiene los nombres de fichero:



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 vacía 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'. Por último, se asignan dichas señales a las entradas y salidas del componente (líneas 68 a 74).



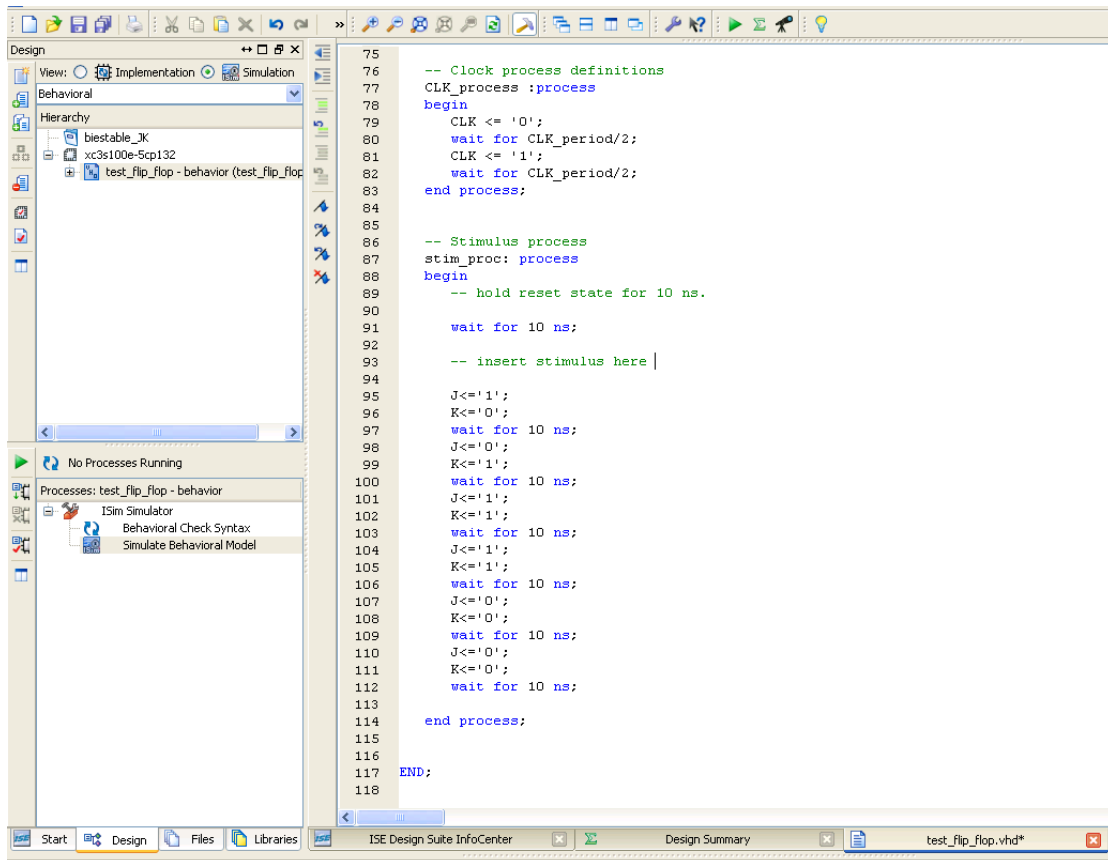
Los dos procesos que siguen a continuación crean la secuencia temporal de las señales de entrada que serán empleadas por el simulador para generar las salidas. Se trata en este caso de dos procesos:

- **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).

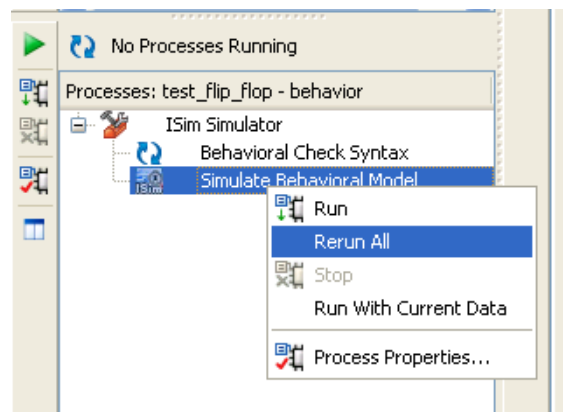
En nuestro caso (en la figura siguiente), hemos escrito algunas líneas con diferentes valores para las entradas (J y K). Tenga en cuenta que se trata solamente de un ejemplo reducido, no un banco de prueba completo para comprobar el funcionamiento exhaustivo del flip flop.

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.

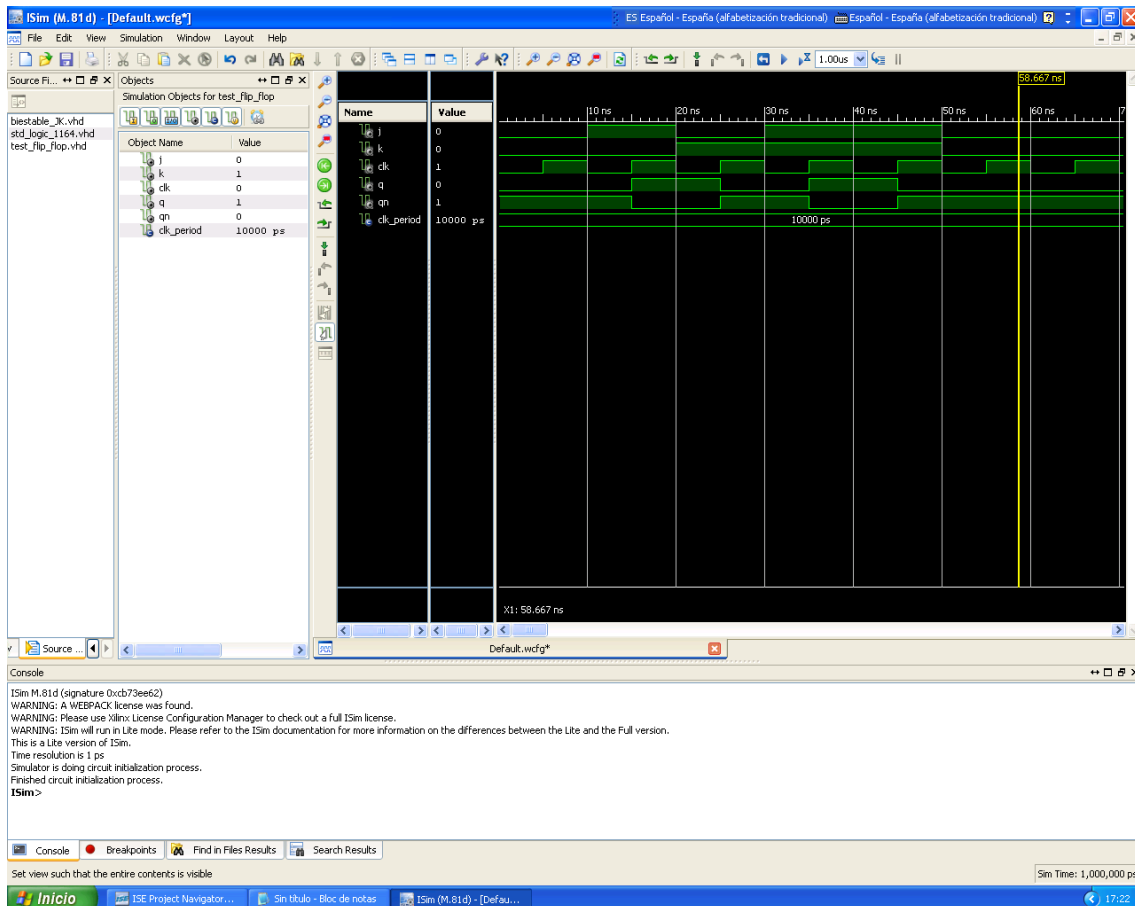
Este proceso se ejecuta en paralelo con el anterior, por lo que es posible sincronizar los valores de las entradas con la señal de reloj. Esta característica es muy importante, por ejemplo, a la hora de simular autómatas.



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
          DECBDCD : 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 DECBDCD<=
"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 "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      (LÍNEAS NO NECESARIAS)
```

```

    -- <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 10 ns.
    wait for 10 ns;
    BIN<="11000";
    wait for 10 ns;
    BIN<="11111";

--      wait for <clock>_period*10;

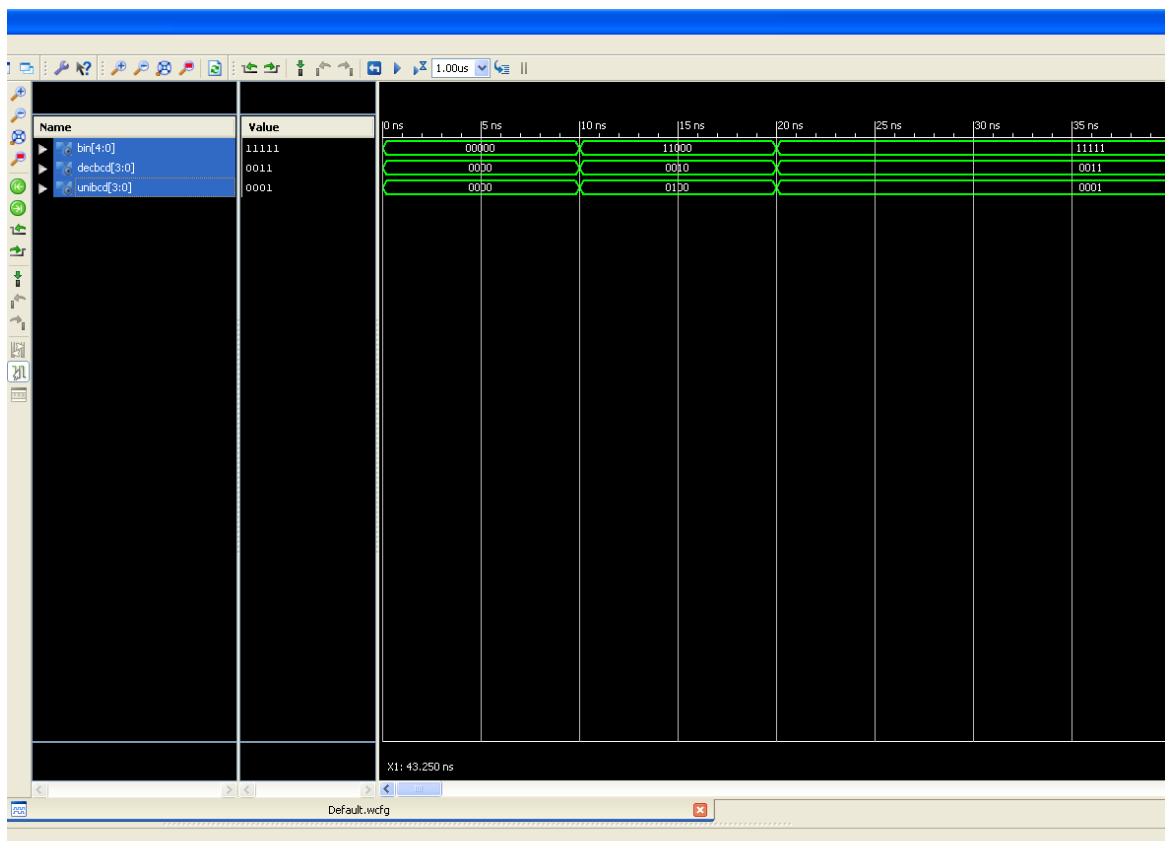
    -- insert stimulus here

    wait;
end process;

END;

```

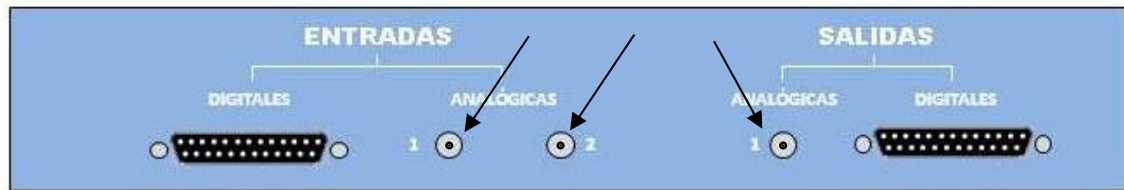
En este código solamente se tiene en cuenta el proceso stim_proc donde asignamos el valor “11000” a la entrada y tras 10 ns asignamos el valor “11111”. Inicialmente se esperan 10 ns en los que la entrada es “00000”. El resultado de la simulación puede verse a continuación, observe que los valores de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



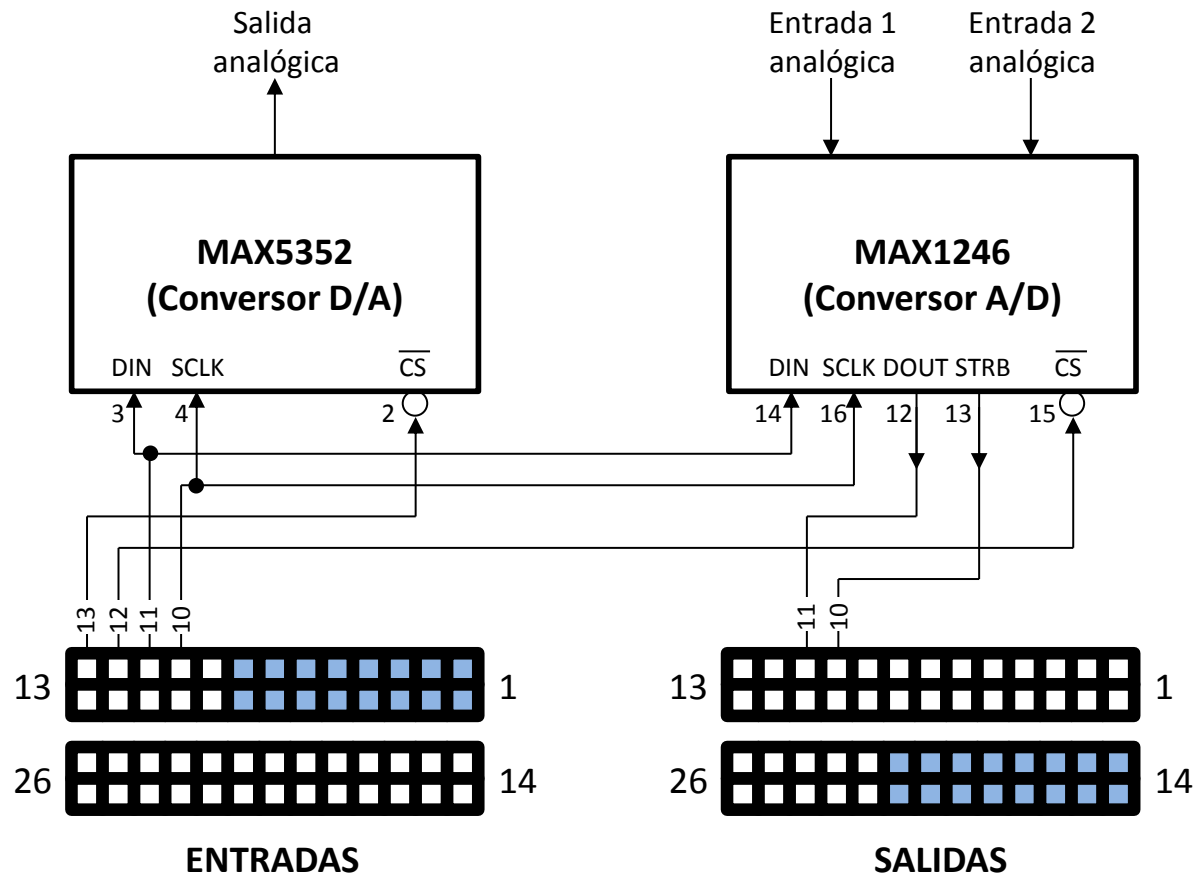
n the differences between the Lite and the Full version.

Utilización de los convertidores A/D y D/A

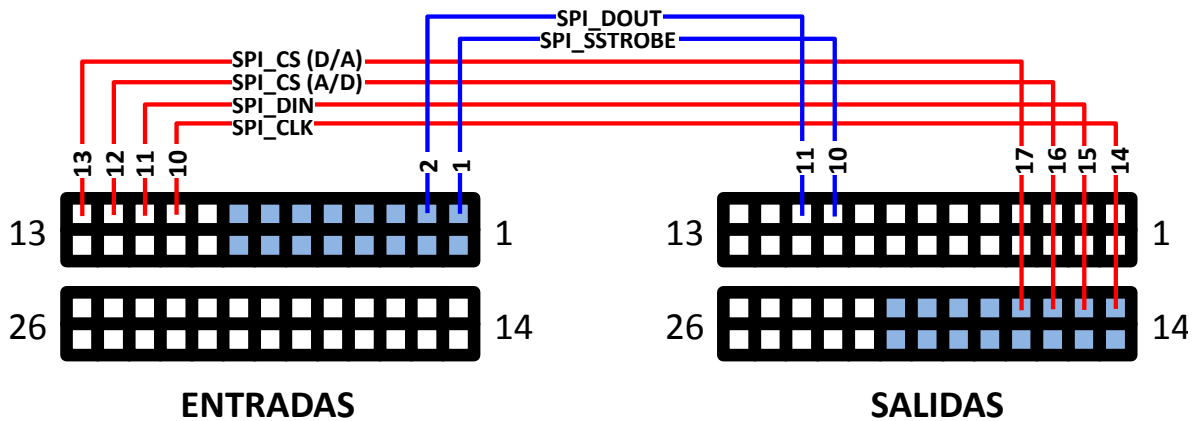
El entorno de desarrollo de la FPGA contiene dos convertidores de 12 bits: un MAX5352 (convertidor D/A) y un MAX1246 (convertidor A/D) cuyos terminales analógicos están disponibles en la parte frontal de la caja azul. Se puede acceder a dos canales analógicos de entrada y un canal analógico de salida (ver figura siguiente):



Los convertidores utilizan el protocolo SPI para su comunicación con el elemento de control (en este caso la BASYS2). Los terminales digitales que permiten el control y la transferencia de datos hacia y desde estos convertidores están conectados a los conectores de entrada y salida de la tabla de madera según las siguientes conexiones:



Por lo tanto, para utilizar los convertidores es necesario conectar físicamente con unos cables, las entradas y salidas de la FPGA (sombreadas en la figura y descritas en la pg. 33) con los terminales correspondientes al control de estos dispositivos. A continuación se muestra un ejemplo de conexión para poder utilizar los dos convertidores empleando la tarjeta BASYS2:



Ejemplo de conexión entre las entradas y salidas de la FPGA
y los terminales de los convertidores para su utilización

Se han escrito dos módulos VHDL para la utilización de estos componentes. Las interfaces de entrada/salida, así como su utilización se muestran a continuación. No obstante, el funcionamiento interno de estos módulos se sale de los objetivos de este manual. Ambos módulos están disponibles en los ejemplos 9 y 10.

Módulo VHDL para la utilización de los convertidores DAC y ADC

El convertidor DAC MAX5352 genera una señal analógica en respuesta al valor digital binario de 12 bits presente en su entrada. Según las configuraciones internas del citado circuito integrado y de las protecciones presentes en el interior del entrenador (caja azul), los valores analógicos de tensión correspondientes a cada valor binario proporcionados en el conector exterior son los siguientes:

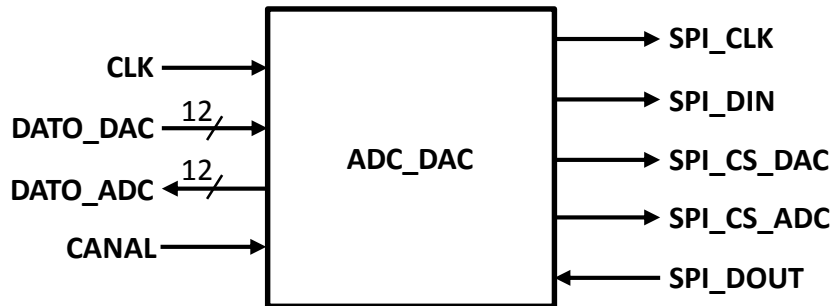
Código binario (12 bits)	Tensión analógica de salida
0000 0000 0000	0,00 V
0000 0000 0001	0,61 mV
.....
1111 1111 1111	2,50 V

El convertidor ADC MAX1246 lee la tensión analógica presente en una de sus dos entradas y genera el valor digital de 12 bits correspondiente representado en complemento a 2, según la siguiente tabla.

Tensión analógica de entrada	Código binario (12 bits)	Valor decimal
+2,50 V	0111 1111 1111	+2047
.....
+1,00 V	0011 0011 0100	+820
.....
0,00 V	0000 0000 0000	0
.....
-1,00 V	1100 1100 1100	-820
.....
-2,50 V	1000 0000 0000	-2048

Debido a la configuración interna y las protecciones del entrenador (caja azul), el intervalo práctico aprovechable del ADC es de -1 V a $+1\text{ V}$ donde los valores digitales correspondientes a cada tensión varían ligeramente en cada puesto.

El módulo diseñado para utilizar los convertidores (ADC_DAC.VHD) tiene las siguientes entradas y salidas (observe que la señal STROBE no se utiliza):



```
entity ADC_DAC is
  Port ( CLK      : in   STD_LOGIC;           -- Reloj de la FPGA
        DATO_DAC  : in   STD_LOGIC_VECTOR (11 downto 0); -- Dato para enviar al DAC
        DATO_ADC  : out  STD_LOGIC_VECTOR (11 downto 0); -- Dato leído desde el ADC
        CANAL     : in   STD_LOGIC;           -- Canal de entrada del ADC
        SPI_CLK   : out  STD_LOGIC;           -- Salida de reloj hacia el puerto SPI
        SPI_DIN   : out  STD_LOGIC;           -- Salida de datos hacia el puerto SPI
        SPI_CS_DAC : out  STD_LOGIC;           -- Salida de CS del DAC hacia el puerto SPI
        SPI_CS_ADC : out  STD_LOGIC;           -- Salida de CS del ADC hacia el puerto SPI
        SPI_DOUT  : in   STD_LOGIC;           -- Entrada de datos desde el puerto SPI
end ADC_DAC;
```

- **CLK:** Debe conectarse al reloj de la FPGA de 50 MHz (patilla M6).
- **DATO_DAC:** Es el dato de 12 bits de entrada para el convertidor D/A.
- **DATO_ADC:** Es el dato de 12 bits leído desde el convertidor A/D.
- **CANAL:** Indica el canal de entrada al ADC: 0 para el canal 1 y 1 para el canal 2.
- **SPI_CLK:** Salida de la FPGA que debe conectarse a la entrada SCLK (ENTRADA 10).
- **SPI_DIN:** Salida de la FPGA que debe conectarse a la entrada DIN (ENTRADA 11).
- **SPI_CS_DAC:** Salida de la FPGA que debe conectarse a la entrada CS del convertidor DAC (ENTRADA 13).
- **SPI_CS_ADC:** Salida de la FPGA que debe conectarse a la entrada CS del convertidor ADC (ENTRADA 12).
- **SPI_DOUT:** Entrada de la FPGA que debe conectarse a la salida DOUT del convertidor DAC (SALIDA 11).

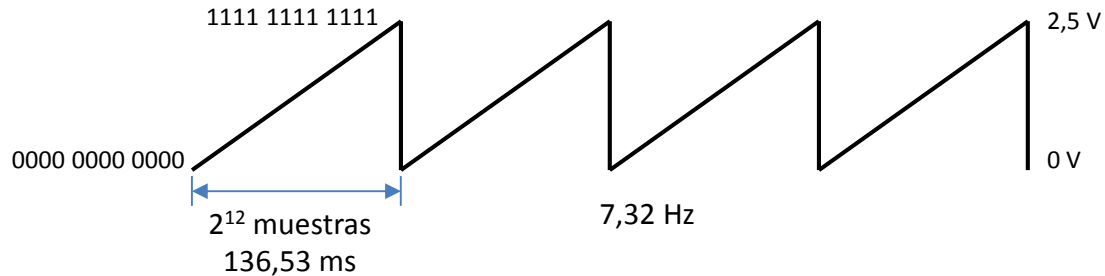
El módulo realiza conversiones AD y DA constantemente en bucle a la máxima frecuencia soportada por los dispositivos. Según las especificaciones del MAX5232 y del MAX1246 y en función de las configuraciones internas, **la máxima frecuencia de muestreo que se permite con este módulo es de 30 kHz (30.000 muestras por segundo).**

Ejemplo 9: generación de una onda diente de sierra (sawtooth) en la salida analógica

El ejemplo 9 emplea un process que genera una señal diente de sierra en la salida analógica. Para ello se define un contador de 12 bits que se incrementa de uno en uno desde 0000 0000 0000 hasta el valor máximo 1111 1111 1111 donde se desborda y vuelve a 0. Las muestras se

envían al convertidor con una frecuencia de 30 kHz, por lo que el periodo y la frecuencia de la señal diente de sierra generada será de:

$$T = \frac{2^{12} = 4096 \text{ muestras}}{30000 \text{ muestras/s}} = 136,53 \text{ ms}, \text{ por tanto } f = \frac{1}{136,53 \text{ ms}} = 7,32 \text{ Hz}$$



Como consecuencia se obtiene una señal diente de sierra de 2,5 V de amplitud y 7,32 Hz de frecuencia.

A continuación se describe el fichero que sintetiza este ejemplo donde se incluye como componente el módulo anterior.

FICHERO: sawtooth.vhd

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

entity sawtooth is
    port ( CLK          : in  STD_LOGIC;          -- reloj FPGA
          SPI_CLK       : out STD_LOGIC;          -- Salida de reloj hacia el puerto SPI
          SPI_DIN        : out STD_LOGIC;          -- Salida de datos hacia el puerto SPI
          SPI_CS_DAC     : out STD_LOGIC;          -- Salida de CS del DAC hacia el puerto SPI
          SPI_CS_ADC     : out STD_LOGIC;          -- Salida de CS del ADC hacia el puerto SPI
          SPI_DOUT       : in  STD_LOGIC);         -- Entrada de datos desde el puerto SPI
end sawtooth;

architecture a_sawtooth of sawtooth is

    component ADC_DAC is
        Port ( CLK          : in  STD_LOGIC;          -- Reloj de la FPGA
              DATO_DAC      : in  STD_LOGIC_VECTOR (11 downto 0); -- Dato para enviar al DAC
              DATO_ADC      : out STD_LOGIC_VECTOR (11 downto 0); -- Dato leído desde el ADC
              CANAL         : in  STD_LOGIC;          -- Canal de entrada del ADC
              SPI_CLK       : out STD_LOGIC;          -- Salida de reloj hacia el puerto SPI
              SPI_DIN        : out STD_LOGIC;          -- Salida de datos hacia el puerto SPI
              SPI_CS_DAC     : out STD_LOGIC;          -- Salida de CS del DAC hacia el puerto SPI
              SPI_CS_ADC     : out STD_LOGIC;          -- Salida de CS del ADC hacia el puerto SPI
              SPI_DOUT       : in  STD_LOGIC);         -- Entrada de datos desde el puerto SPI
    end component;

    constant TPO_MUEST : integer := 1667;          -- 33 us correspondiente a 30 kHz

    signal cont : unsigned (31 downto 0):=(others=>'0'); -- contador para determinar la
                                                         -- frec. de muestreo
    signal s_dat : unsigned (11 downto 0):=(others=>'0'); -- contador que realiza la
                                                         -- escalera de tensiones
    signal s_null : STD_LOGIC_VECTOR (11 downto 0):=(others=>'0'); -- señal para volcar la
                                                         -- salida del ADC no utilizada
```

```

begin

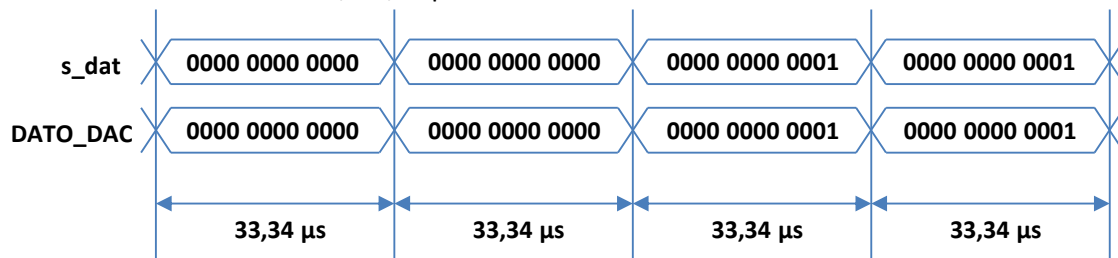
U1 : process (CLK)
begin
    if (CLK'event and CLK='1') then      -- En cada flanco de subida
        cont<=cont+1;                    -- incrementar contador
        if (cont>=TPO_MUEST) then        -- muestreo 30000 Hz
            cont<=(others=>'0');          -- poner el contador a 0
            s_dat<=s_dat+1;               -- incrementar s_dat (escalera de tensiones)
        end if;
    end if;
end process;

U2 : ADC_DAC port map
(
    CLK      => CLK,
    DATO_DAC => STD_LOGIC_VECTOR(s_dat),
    DATO_ADC => s_null,
    CANAL    => '0',
    SPI_CLK  => SPI_CLK,
    SPI_DIN  => SPI_DIN,
    SPI_CS_DAC => SPI_CS_DAC,
    SPI_CS_ADC => SPI_CS_ADC,
    SPI_DOUT => SPI_DOUT
);

end a_sawtooth;

```

En este código se declara una señal `s_dat` que tiene 12 bits. El process espera a que hayan transcurrido 1667 ciclos de reloj ($1667 \times 20\text{ns} = 33,34 \mu\text{s}$) y entonces incrementa la señal `s_dat`. La entrada `DATO_DAC` está conectada a la señal `s_dat`. De este modo se consigue una frecuencia de muestreo de $1 / 33,34 \mu\text{s} = 30.000 \text{ Hz}$:



Observe que se declara una señal llamada `s_null` donde se vuelca la lectura del convertidor ADC. Recuerde que el módulo realiza la conversión AD y DA constantemente en bucle. Esta señal no se utiliza para nada, por lo tanto al compilar este ejemplo aparecerán una serie de WARNINGS que deberán ser ignorados.

Conecte los cables como se indica en la figura de la página 57 y utilice el siguiente archivo de asociaciones. Conectando el osciloscopio a la SALIDA ANALOGICA debería observar una señal diente de sierra con una amplitud de 2,5 V y una frecuencia de 7,32 Hz.

FICHERO: asociaciones.ucf

```

NET "CLK" LOC = "M6";           # Reloj de la FPGA

# SEÑALES DEL PUERTO SPI

NET "SPI_CLK" LOC = "A9";       # Salida 14
NET "SPI_DIN" LOC = "B9";       # Salida 15
NET "SPI_CS_ADC" LOC = "A10";   # Salida 16
NET "SPI_CS_DAC" LOC = "C9";    # Salida 17
NET "SPI_DOUT" LOC = "A3";      # Entrada 2

```

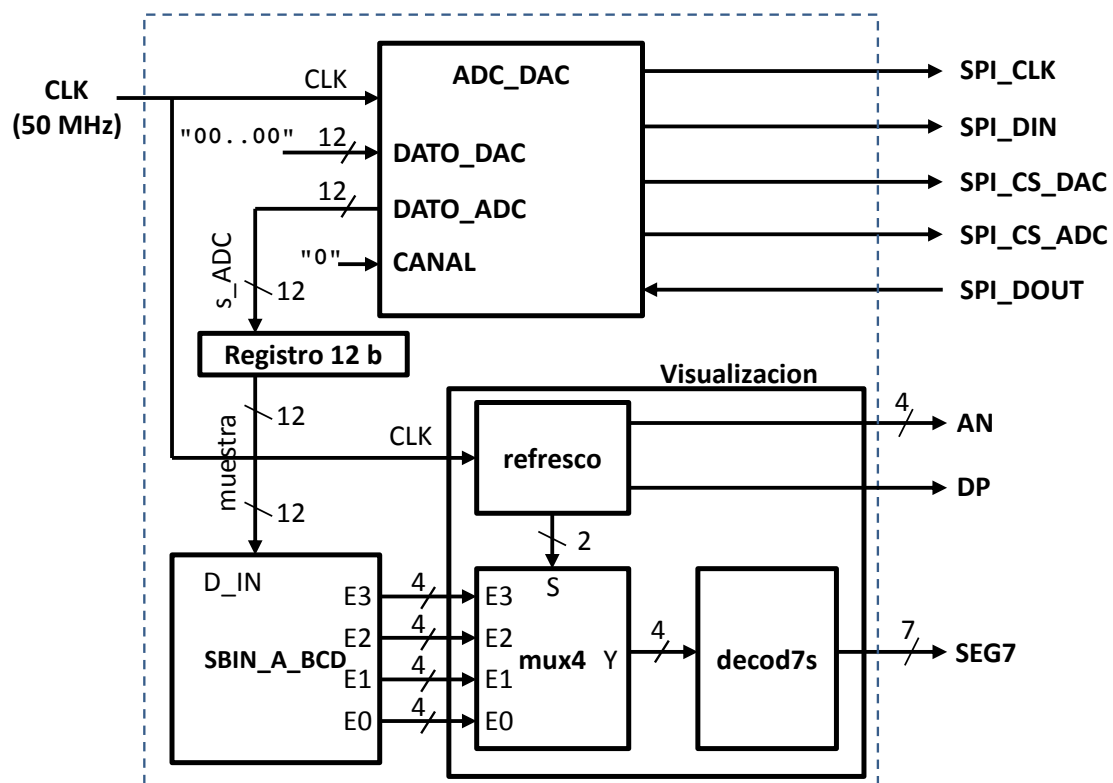

Ejercicios propuestos sobre el ejemplo 9:

1. Modifique el código del ejemplo para generar una señal diente de sierra de 1 kHz de frecuencia y 2,5 V de amplitud manteniendo la frecuencia de muestreo de 30 kHz.
2. Modifique el código del ejemplo para generar una señal diente de sierra de 1 kHz de frecuencia y 1 V de amplitud manteniendo la frecuencia de muestreo de 30 kHz.
3. Modifique el código del ejemplo para obtener una señal diente de sierra de 100 Hz de frecuencia y 1 V de amplitud bajando la frecuencia de muestreo a 8 kHz.
4. Modifique el código del ejemplo para obtener una señal senoidal de 200 Hz de frecuencia y 1 V de amplitud con una frecuencia de muestreo de 8 kHz. (NOTA: Para realizar este ejercicio deberá almacenar un determinado número de muestras de la función seno en un módulo combinacional como puede ser una ROM).

Ejemplo 10: Voltímetro digital

En este ejemplo se utiliza el convertidor AD para leer la tensión presente en el canal analógico 1 y mostrar su valor a través de los displays de 7 segmentos. La tensión de entrada debe encontrarse dentro del intervalo adecuado para el convertidor (entre $-2,5$ y $+2,5$ V). El display de la izquierda muestra el signo (apagado o guión), el siguiente display muestra las unidades de la tensión leída, el tercer display indica las décimas de V y el cuarto indica las centésimas.

El esquema general de bloques de este circuito se muestra a continuación en la figura:



El módulo principal (**voltimetro.vhd**) contiene un proceso que incrementa constantemente un contador. Cuando dicho contador llega al valor equivalente al transcurso de 100 ms (10 Hz) toma una muestra del convertidor A/D cargándola en el registro de 12 bits.

El módulo **SBIN_A_BCD** toma el valor de la muestra como entrada. En primer lugar convierte la escala de ± 2047 a ± 250 (valor máximo de tensión que permite el convertidor). Para ello divide el valor de la muestra por 8 (en realidad produce una conversión a ± 256 , pero el valor de la última centésima no es demasiado significativo). Después obtiene el signo y el valor absoluto de esta medida y extrae las tres cifras BCD que deben representarse en los displays. El signo se presenta en el de la izquierda, bien apagado (para signo positivo), o mostrando un guión (para signo negativo).

El módulo **visualización** realiza la actualización periódica de los displays para representar las cifras BCD. Mantiene encendido el punto decimal entre las unidades y las décimas de V.

FICHERO: voltimetro.vhd

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

entity voltimetro is
    Port ( CLK      : in    STD_LOGIC;           -- Reloj FPGA
          SEG7      : out   STD_LOGIC_VECTOR (0 to 6); -- Activación segmentos
          DP        : out   STD_LOGIC;           -- punto decimal
          AN        : out   STD_LOGIC_VECTOR (3 downto 0); -- Activación displays
          SPI_CLK    : out   STD_LOGIC;           -- Salida de reloj hacia el puerto SPI
          SPI_DIN    : out   STD_LOGIC;           -- Salida de datos hacia el puerto SPI
          SPI_CS_DAC : out   STD_LOGIC;           -- Salida de CS del DAC hacia el puerto SPI
          SPI_CS_ADC : out   STD_LOGIC;           -- Salida de CS del ADC hacia el puerto SPI
          SPI_DOUT   : in    STD_LOGIC;           -- Entrada de datos desde el puerto SPI
    end voltimetro;

    architecture a_voltimetro of voltimetro is

        component ADC_DAC
            Port ( CLK      : in    STD_LOGIC;           -- Reloj de la FPGA
                  DATO_DAC  : in    STD_LOGIC_VECTOR (11 downto 0); -- Dato para enviar al DAC
                  DATO_ADC  : out   STD_LOGIC_VECTOR (11 downto 0); -- Dato leído desde el ADC
                  CANAL     : in    STD_LOGIC;           -- Canal de entrada del ADC
                  SPI_CLK    : out   STD_LOGIC;           -- Salida de reloj hacia el puerto SPI
                  SPI_DIN    : out   STD_LOGIC;           -- Salida de datos hacia el puerto SPI
                  SPI_CS_DAC : out   STD_LOGIC;           -- Salida de CS del DAC hacia el puerto SPI
                  SPI_CS_ADC : out   STD_LOGIC;           -- Salida de CS del ADC hacia el puerto SPI
                  SPI_DOUT   : in    STD_LOGIC;           -- Entrada de datos desde el puerto SPI
            end component;

        component SBIN_A_BCD
            Port ( D_IN : in    STD_LOGIC_VECTOR (11 downto 0); -- Dato de entrada en binario
                  E3   : out   STD_LOGIC_VECTOR (3 downto 0); -- Signo
                  E2   : out   STD_LOGIC_VECTOR (3 downto 0); -- Unidades
                  E1   : out   STD_LOGIC_VECTOR (3 downto 0); -- Décimas
                  E0   : out   STD_LOGIC_VECTOR (3 downto 0); -- Centésimas
            end component;

        component visualizacion
            port (CLK : in    STD_LOGIC;           -- Reloj FPGA
                  E0  : in    STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 0 (uds)
                  E1  : in    STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 1 (dec)
                  E2  : in    STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 2 (cent)
                  E3  : in    STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 3 (uds. millar)
                  SEG7 : out   STD_LOGIC_VECTOR (0 to 6); -- Activación segmentos
                  DP   : out   STD_LOGIC;           -- punto decimal
                  AN   : out   STD_LOGIC_VECTOR (3 downto 0); -- Activación displays
            end component;

        constant TPO_MUEST : integer := 500000; -- 100 ms correspondiente a 10 Hz
```

```

signal cont      : unsigned (31 downto 0):=(others=>'0'); -- contador para determinar la
                                                         -- frec. de muestreo
signal s_ADC     : STD_LOGIC_VECTOR (11 downto 0); -- valor digital leído del ADC
signal muestra    : STD_LOGIC_VECTOR (11 downto 0); -- muestras tomadas
signal s_E0      : STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 0 (uds)
signal s_E1      : STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 1 (dec)
signal s_E2      : STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 2 (cent)
signal s_E3      : STD_LOGIC_VECTOR (3 downto 0); -- Entrada cifra 3 (uds. millar)
begin

  U1 : process (CLK)
  begin
    if (CLK'event and CLK='1') then -- En cada flanco de subida
      cont<=cont+1; -- incrementar contador
      if (cont>=TPO_MUEST) then -- al cumplirse el tiempo de muestreo
        cont<=(others=>'0'); -- poner el contador a 0
        muestra<=s_ADC; -- Capturar la muestra
      end if;
    end if;
  end process;

  U2 : ADC_DAC port map
  (
    CLK      => CLK,
    DATO_DAC => "000000000000", -- El dato en el DAC es 0
    DATO_ADC => s_ADC,
    CANAL    => '0', -- Leer del canal 1
    SPI_CLK  => SPI_CLK,
    SPI_DIN  => SPI_DIN,
    SPI_CS_DAC => SPI_CS_DAC,
    SPI_CS_ADC => SPI_CS_ADC,
    SPI_DOUT => SPI_DOUT
  );

  U3 : SBIN_A_BCD port map
  (
    D_IN => muestra,
    E3   => s_E3,
    E2   => s_E2,
    E1   => s_E1,
    E0   => s_E0
  );

  U4 : visualizacion port map
  (
    CLK  => CLK,
    E0   => s_E0,
    E1   => s_E1,
    E2   => s_E2,
    E3   => s_E3,
    SEG7 => SEG7,
    DP   => DP,
    AN   => AN
  );

```

end a_voltimetro;

El process incrementa un contador. Cuando dicho contador llega al límite (TPO_MUEST) se pone a cero y se captura una muestra en el registro muestra (declarado como signal).

FICHERO: SBIN_A_BCD.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SBIN_A_BCD is
  Port ( D_IN : in  STD_LOGIC_VECTOR (11 downto 0); -- Dato de entrada en binario
        E3   : out STD_LOGIC_VECTOR (3 downto 0); -- Signo
        E2   : out STD_LOGIC_VECTOR (3 downto 0); -- Unidades
        E1   : out STD_LOGIC_VECTOR (3 downto 0); -- Décimas
        E0   : out STD_LOGIC_VECTOR (3 downto 0)); -- Centésimas
end SBIN_A_BCD;

```

```

architecture a_SBIN_A_BCD of SBIN_A_BCD is

    signal VOLT      : unsigned (11 downto 0);      -- Valor en voltios hasta 250 (2,5 V)
    signal D_IN_C2   : STD_LOGIC_VECTOR (11 downto 0); -- Almacena el complemento a 2
    signal V_ABS     : STD_LOGIC_VECTOR (11 downto 0); -- Valor absoluto de D_IN
    signal AUX_E1    : unsigned (11 downto 0);      -- Señal auxiliar para el cálculo de E1
    signal AUX_E0    : unsigned (11 downto 0);      -- Señal auxiliar para el cálculo de E0

begin

    -- Tratamiento del signo

    E3<="1010" when D_IN(11)='0' else "1111";      -- 1010 display apagado, 1111 es el guión

    -- Generación del complemento a 2 de D_IN

    D_IN_C2<=std_logic_vector (("111111111111" xor unsigned(D_IN)) + 1);

    -- Selección del valor absoluto

    with D_IN(11) select V_ABS <=
        D_IN      when '0',      -- Si D_IN es positivo se selecciona D_IN
        D_IN_C2    when '1',      -- Si es negativo se selecciona su complemento a 2
        D_IN      when others;

    -- Cambio de escala: 2048 se corresponden con 2,50 V (250)

    VOLT <= "000" & unsigned(V_ABS(11 downto 3)); -- dividimos por 8: 2048/8 aprox 250

    -- Obtención de las cifras BCD

    E2 <= "0010" when VOLT>199 else
        "0001" when VOLT>99  else
        "0000";

    AUX_E1 <= VOLT-200 when VOLT>199 else
        VOLT-100 when VOLT>99  else
        VOLT;

    E1 <= "1001" when AUX_E1>89 else
        "1000" when AUX_E1>79 else
        "0111" when AUX_E1>69 else
        "0110" when AUX_E1>59 else
        "0101" when AUX_E1>49 else
        "0100" when AUX_E1>39 else
        "0011" when AUX_E1>29 else
        "0010" when AUX_E1>19 else
        "0001" when AUX_E1>9  else
        "0000";

    AUX_E0 <= AUX_E1-90 when AUX_E1>89 else
        AUX_E1-80 when AUX_E1>79 else
        AUX_E1-70 when AUX_E1>69 else
        AUX_E1-60 when AUX_E1>59 else
        AUX_E1-50 when AUX_E1>49 else
        AUX_E1-40 when AUX_E1>39 else
        AUX_E1-30 when AUX_E1>29 else
        AUX_E1-20 when AUX_E1>19 else
        AUX_E1-10 when AUX_E1>9  else
        AUX_E1;

    E0 <= "1001" when AUX_E0=9 else
        "1000" when AUX_E0=8 else
        "0111" when AUX_E0=7 else
        "0110" when AUX_E0=6 else
        "0101" when AUX_E0=5 else
        "0100" when AUX_E0=4 else
        "0011" when AUX_E0=3 else
        "0010" when AUX_E0=2 else
        "0001" when AUX_E0=1 else
        "0000";

end a_SBIN_A_BCD;

```

Este módulo es un bloque combinacional que realiza las siguientes operaciones:

1. En función del signo de D_IN asigna el valor del DISPLAY 3 (más a la izquierda). Si es positivo (0) se asigna un carácter en blanco. Si es negativo (1) se asigna el 1111 que se corresponde con el guión.
2. Genera el complemento a 2 de D_IN en la señal D_IN_C2.
3. Obtiene el valor absoluto de D_IN (V_ABS) mediante un multiplexor cuya entrada de control es el signo de D_IN y sus entradas de datos son D_IN y D_IN_C2.
4. Cambia de escala V_ABS. Esta señal toma valores entre 0 y 2048. Sin embargo el conversor A/D lee tensiones entre 0 y 2,5 V. La señal VOLT toma valores entre 0 y 250 (2,5 V). Para ello se desplaza el valor V_ABS tres bits hacia la derecha, por lo que se divide entre 8. Esto hace que el intervalo de valores de VOLT sea entre 0 y 256. Aproximadamente se consigue la conversión a 250.
5. Por último se obtienen las cifras BCD correspondientes al valor VOLT. Esto se lleva a cabo con varios multiplexores y comparadores que según los intervalos van seleccionando los valores de las cifras.
 - Si $VOLT > 199$ la primera cifra (E2) es un 2. Además $AUX_E1 = VOLT - 200$
 - Si $VOLT > 99$ la primera cifra (E2) es un 1. Además $AUX_E1 = VOLT - 100$
 - En otro caso la primera cifra (E2) es un 0. Además $AUX_E1 = VOLT$.
 - Si $AUX_E1 > 89$ la segunda cifra (E1) es un 9. Además $AUX_E0 = AUX_E1 - 90$
 - Si $AUX_E1 > 79$ la segunda cifra (E1) es un 8. Además $AUX_E0 = AUX_E1 - 80$
 - Si $AUX_E1 > 69$ la segunda cifra (E1) es un 7. Además $AUX_E0 = AUX_E1 - 70$
 - Si $AUX_E1 > 59$ la segunda cifra (E1) es un 6. Además $AUX_E0 = AUX_E1 - 60$
 - Si $AUX_E1 > 49$ la segunda cifra (E1) es un 5. Además $AUX_E0 = AUX_E1 - 50$
 - Si $AUX_E1 > 39$ la segunda cifra (E1) es un 4. Además $AUX_E0 = AUX_E1 - 40$
 - Si $AUX_E1 > 29$ la segunda cifra (E1) es un 3. Además $AUX_E0 = AUX_E1 - 30$
 - Si $AUX_E1 > 19$ la segunda cifra (E1) es un 2. Además $AUX_E0 = AUX_E1 - 20$
 - Si $AUX_E1 > 9$ la segunda cifra (E1) es un 1. Además $AUX_E0 = AUX_E1 - 10$
 - En otro caso, la segunda cifra (E1) es un 0. Además $AUX_E0 = AUX_E1$.
 - Si $AUX_E0 = 9$ la tercera cifra (E0) es un 9.
 - Si $AUX_E0 = 8$ la tercera cifra (E0) es un 8.
 - Si $AUX_E0 = 7$ la tercera cifra (E0) es un 7.
 - Si $AUX_E0 = 6$ la tercera cifra (E0) es un 6.
 - Si $AUX_E0 = 5$ la tercera cifra (E0) es un 5.
 - Si $AUX_E0 = 4$ la tercera cifra (E0) es un 4.
 - Si $AUX_E0 = 3$ la tercera cifra (E0) es un 3.
 - Si $AUX_E0 = 2$ la tercera cifra (E0) es un 2.
 - Si $AUX_E0 = 1$ la tercera cifra (E0) es un 1.
 - En otro caso, la tercera cifra (E0) es un 0.

FICHERO: visualizacion.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity visualizacion is
    port (CLK : in STD_LOGIC;
          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);
          SEG7 : out STD_LOGIC_VECTOR (0 to 6);
          DP : out STD_LOGIC;
          AN : out STD_LOGIC_VECTOR (3 downto 0));
end visualizacion;

architecture a_visualizacion of visualizacion is

    -- Componentes

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

    -- Señales para interconexión de módulos

    signal N_Y : STD_LOGIC_VECTOR (3 downto 0);
    signal N_S : STD_LOGIC_VECTOR (1 downto 0);

    begin

        -- Interconexiones

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

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

        U3 : refresco
            port map (
                CLK=>CLK,
                AN=>AN,
                S=>N_S,
                DP=>DP
            );

    end a_visualizacion;

```

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 "1010",
        "1111111" when "1011",
        "1111111" when "1100",
        "1111111" when "1101",
        "1111111" when "1110",
        "1111100" when "1111",
        "1111111" when others;
end a_decod7s;

```

Se puede observar que este módulo implementa un decodificador que convierte la entrada en BCD de 4 bits a los valores de los segmentos que deben encenderse. En este caso solamente nos interesan cifras decimales de 0 a 9. Hemos asignado las cifras A,B,C,D y E como display apagado (1111111) y la cifra F como un guión (segmento central encendido).

FICHERO: refresco.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity refresco 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
          DP  : out STD_LOGIC);          -- Punto decimal
end refresco;

architecture a_refresco of refresco is

    signal cont : unsigned (31 downto 0);
    signal SS   : unsigned (1 downto 0);

begin

    process (CLK)
    begin
        if (CLK'event and CLK='1') then
            cont<=cont+1;
            if (cont>=50000) then
                cont<=(others=>'0');
                SS<=SS+1;
            end if;
        end if;
    end process;
end a_refresco;

```

```

S<=STD_LOGIC_VECTOR(SS);      -- Señal para el multiplexor

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

DP<='0' when SS="10" else '1'; -- se enciende el punto decimal en el display 2

end a_refresco;

```

Este módulo genera la secuencia de refresco de los displays. La salida DP debe conectarse al punto decimal. Como se puede ver, dicho punto decimal solamente se enciende cuando está activo el display 2 (unidades).

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;

```

Conecte los cables como se indica en la figura de la página 57 y utilice el siguiente archivo de asociaciones. Utilizando un cable BNC terminado en bananas conecte el BNC a la entrada analógica 1 y las bananas a la fuente de alimentación. Al variar la tensión entre 0 y 2,5 V podrá leer su valor en los displays de 7 segmentos con el signo adecuado.

FICHERO: asociaciones.ucf

```

NET "CLK" LOC = "M6"; # Reloj de la FPGA

# SEÑALES DE ACTIVACIÓN DE LOS 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
NET "DP"      LOC = "N13"; # Punto decimal

# 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
```

SEÑALES DEL PUERTO SPI

```
NET "SPI_CLK"    LOC = "A9";    # Salida 14
NET "SPI_DIN"    LOC = "B9";    # Salida 15
NET "SPI_CS_ADC" LOC = "A10";   # Salida 16
NET "SPI_CS_DAC" LOC = "C9";    # Salida 17
NET "SPI_DOUT"   LOC = "A3";    # Entrada 2
```

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

Hoja de características del MAX5352

Hoja de características del MAX1246