Práctica 4 - CIRCUITOS COMBINACIONALES. DISEÑO CON VHDL

Objetivos

El propósito de esta práctica es:

- Comprender y aplicar las tablas de verdad al diseño de circuitos combinacionales.
- Diseñar, entender e integrar un display de 7 segmentos en un sistema digital.
- Entender el uso de un bus de datos.
- Introducir el diseño con lenguajes de descripción de hardware VHDL.

Material

- Ordenador personal con Quartus II y ModelSim.
- Tarjeta de desarrollo de lógica programable.

Duración

1 sesión.

Trabajo previo e información de consulta

En esta sesión se harán como ejercicios previos:

- Diseño en VHDL de los bloques **BinA7Seg** y **Comparador4Bits**.
- Lectura previa de las notas de clase y tutorial de VHDL.
- Lectura del *Manual de Usuario de la Placa de Lógica Programable DE1*.

Trabajo posterior y entrega de resultados

Después del desarrollo de la práctica se debe entregar un informe de resultados y el trabajo desarrollado en el laboratorio en soporte informático que contenga el proyecto completo. Debe incluir:

1. Descripción del experimento y los pasos seguidos (tablas, expresiones lógicas, K-maps).
2. Esquemas del circuito, código VHDL y su explicación.
3. Simulaciones y su explicación.
4. Resultados y conclusiones.

Introducción

En esta práctica se van a diseñar dos bloques combinacionales. El primero de ellos es el que llamaremos **BinA7Seg**. El bloque es un decodificador que tiene como función la representación en hexadecimal de un número de 4 bits en un display de 7 segmentos, tal como se muestra en la Figura 1. Este componente se usará en prácticas sucesivas. Su tabla de verdad se encuentra en el **anexo A**.

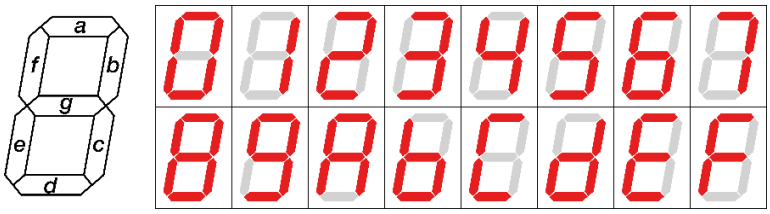

Figura 1. Esquema de un display de 7 segmentos y representación de dígitos hexadecimales.

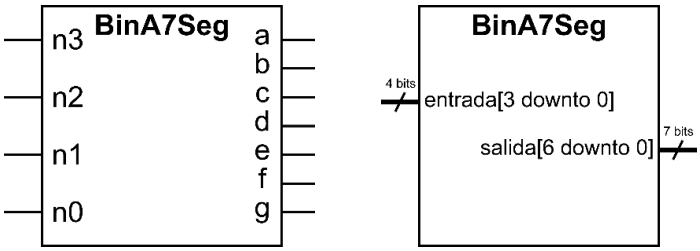

Figura 2. Esquema del componente BinA7Seg (con entradas y salidas en bits y en bus)

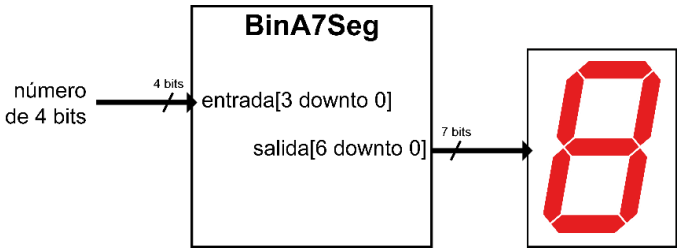

Figura 3. Conexión del componente BinA7Seg.

El segundo bloque es un comparador de 2 números de 4 bits, al que llamaremos Comparador4Bits (en la práctica 2 se ha diseñado un comparador de 2 números de 2 bits).

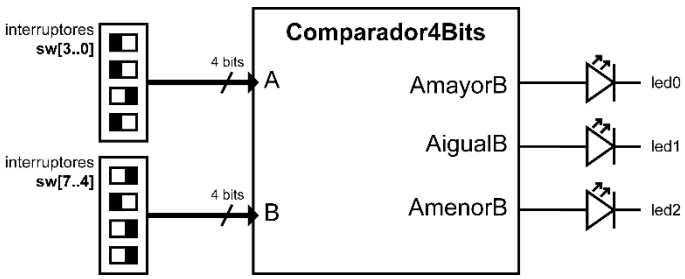

Figura 4. Comparador de 2 números de 4 bits.

Desarrollo Práctico

Para desarrollar esta práctica, hay que crear en **Quartus II** un proyecto con el nombre **Practica4**. Se va a desarrollar todo el proyecto en VHDL. La jerarquía superior de la práctica estaría compuesta por dos decodificadores de 7 segmentos **BinA7Seg** y un bloque **Comparador4Bits**, además de 8 interruptores (SW7 – SW0), 3 LEDs (LED2 – LED0) y 2 displays de 7 segmentos (Display0 y Display1). El diagrama de la práctica quedaría como sigue:

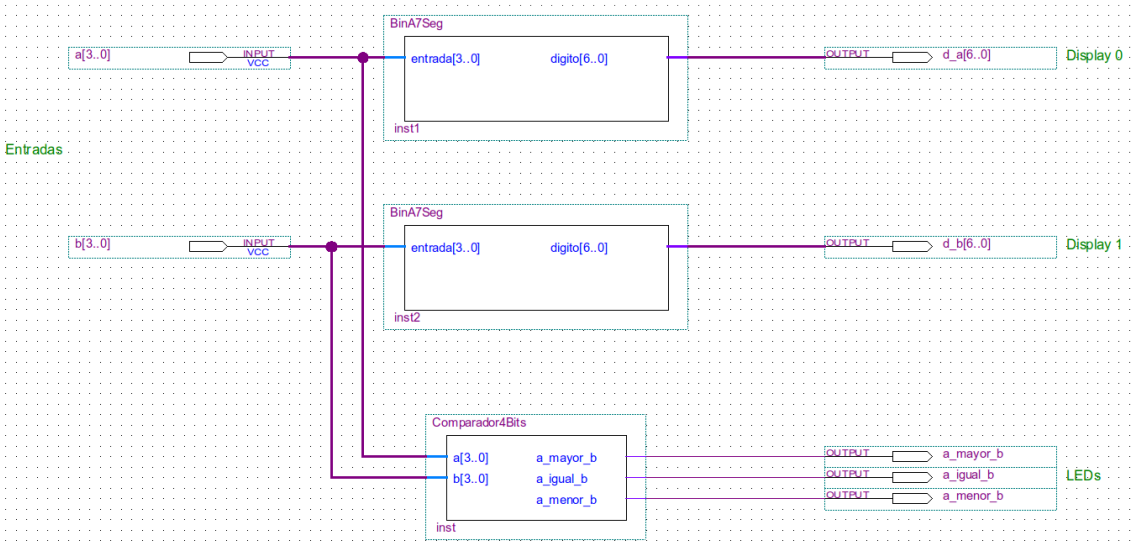

Figura 5. Diagrama general de bloques

El funcionamiento del diagrama anterior es la representación de la salida del comparador en alguno de los tres LEDs si A>B, A=B o A<B. Tanto A como B, son dos números de 4 bits y se introducen a través de los interruptores de la placa de desarrollo y se visualiza su valor en dos displays de 7 segmentos, el Display0 para el número A y el Display1 para el número B.

Una vez creado el proyecto con el nombre **Practica4**, hay que definir en VHDL la descripción de los bloques BinA7Seg.vhd y Comparador4Bits.vhd. Para ello, seleccionamos en **Quartus II** el menú

File ⟶ ***New*** ⟶ elegimos ***VHDL File*** para arrancar el editor de texto.

Hay que codificar en VHDL el componente **BinA7Seg**, siguiendo y completando la tabla de verdad del **anexo A**. El proceso debe repetirse para el bloque **Comparador4Bits**.

Sin embargo, **en lugar de hacer un diagrama de bloques como en la práctica anterior, vamos a diseñar el nivel superior de la jerarquía utilizando también una descripción VHDL** (***File*** ⟶ ***New*** ⟶ ***VHDL File***), que guardaremos con el mismo nombre del proyecto (**Practica4.vhd**). En este archivo instanciaremos los componentes y la lógica correspondiente.

A continuación, consultaremos el *Manual de Usuario de la Placa de Lógica Programable DE1* para identificar el pin de la FPGA con el correspondiente puerto de entrada y de salida según la tabla 1. Dicho manual está disponible en la página de Moodle de la asignatura.

Señal	Tipo	Componente	Pin FPGA
a(0)	Entrada	SW0	
a(1)	Entrada	SW1	
a(2)	Entrada	SW1	
a(3)	Entrada	SW3	
b(0)	Entrada	SW4	
b(1)	Entrada	SW5	
b(2)	Entrada	SW6	
b(3)	Entrada	SW7	
a_mayor_b	Salida	LED Green[0]	
a_igual_b	Salida	LED Green[1]	
a_menor_b	Salida	LED Green[2]	
d_a(6)	Salida	Seven Segment Digit 0[a]	
d_a(5)	Salida	Seven Segment Digit 0[b]	
d_a(4)	Salida	Seven Segment Digit 0[c]	
d_a(3)	Salida	Seven Segment Digit 0[d]	
d_a(2)	Salida	Seven Segment Digit 0[e]	
d_a(1)	Salida	Seven Segment Digit 0[f]	
d_a(0)	Salida	Seven Segment Digit 0[g]	
d_b(6)	Salida	Seven Segment Digit 1[a]	
d_b(5)	Salida	Seven Segment Digit 1[b]	
d_b(4)	Salida	Seven Segment Digit 1[c]	
d_b(3)	Salida	Seven Segment Digit 1[d]	
d_b(2)	Salida	Seven Segment Digit 1[e]	
d_b(1)	Salida	Seven Segment Digit 1[f]	
d_b(0)	Salida	Seven Segment Digit 1[g]	

Tabla 1. Asignación de patillas de la FPGA a las señales del circuito.

Hay que tener en cuenta que **los dispositivos LED de la placa con activos a nivel alto** (para iluminar el LED hay que poner un '1') y **los displays de 7 segmentos son activos a nivel bajo** (para iluminar el segmento correspondiente hay que poner un '0').

Realizado este paso se procede a la compilación del circuito diseñado, corrigiendo los errores, si los hubiera, en el código VHDL.

Procedemos a continuación con la simulación en ModelSim (RTL Simulation). En el anexo B se incluye un banco de prueba para generar los estímulos para la simulación (véase la práctica 3 para recordar cómo añadir el testbench).

Ojo, en el banco de prueba mostrado se ha añadido al generado automáticamente por Quartus el **paquete ieee.numeric_std** (3º línea) y el interior del último PROCESS.

Finalmente realice el volcado a la placa del circuito diseñado para su comprobación por el profesor.

Anexo A

Decodificador de 4 bits a 7 segmentos. Completad la Tabla de verdad para obtener los símbolos de la Figura 1. En la tarjeta de desarrollo, los segmentos de los displays son activos a nivel bajo.

E3	E2	E1	E0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

Anexo B

Archivo de testbench para la Práctica (Practica4.vht).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Practica4_vhd_tst IS
END Practica4_vhd_tst;


ARCHITECTURE Practica4_arch OF Practica4_vhd_tst IS
-- constants
-- signals
SIGNAL a : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL a_igual_b : STD_LOGIC;
SIGNAL a_mayor_b : STD_LOGIC;
SIGNAL a_menor_b : STD_LOGIC;
SIGNAL b : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d_a : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL d_b : STD_LOGIC_VECTOR(6 DOWNTO 0);

COMPONENT Practica4
PORT (
    a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    a_igual_b : OUT STD_LOGIC;
```

```
    a_mayor_b : OUT STD_LOGIC;
    a_menor_b : OUT STD_LOGIC;
    b : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    d_a : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
    d_b : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END COMPONENT;

BEGIN
i1 : Practica4
PORT MAP (
    -- list connections between master ports and signals
    a => a,
    a_igual_b => a_igual_b,
    a_mayor_b => a_mayor_b,
    a_menor_b => a_menor_b,
    b => b,
    d_a => d_a,
    d_b => d_b
    );


init : PROCESS
-- variable declarations
BEGIN
    -- code that executes only once
WAIT;
END PROCESS init;


always : PROCESS
-- optional sensitivity list
-- ( )
-- variable declarations
BEGIN
    -- code executes for every event on sensitivity list
    for i in 0 to 15 loop
        a <= std_logic_vector(to_unsigned(i, 4));
        for j in 0 to 15 loop
            b <= std_logic_vector(to_unsigned(j, 4));
            wait for 100 ns;
            if a = b then
                assert a_igual_b = '1' and a_menor_b = '0' and a_mayor_b = '0'
                    report "fallo en la comparacion a=b"
                    severity failure;
            end if;
            if a < b then
                assert a_igual_b = '0' and a_menor_b = '1' and a_mayor_b = '0'
                    report "fallo en la comparacion a<b"
                    severity failure;
            end if;
            if a > b then
                assert a_igual_b = '0' and a_menor_b = '0' and a_mayor_b = '1'
                    report "fallo en la comparacion a>b"
                    severity failure;
            end if;
        end loop;
    end loop;
    WAIT;
END PROCESS always;

END Practica4_arch;
```