Práctica Nº 4. Decodificador Hexadecimal a 7 segmentos

1. Datos de la práctica

Carrera	INGENIERÍA ELECTRÓNICA			
Semestre			Grupo	
Tipo de Práctica	☐ Laboratorio ☐ Simulación		Fecha	
Asignatura	Electrónica Digital I			
Unidad Temática				
Nº Alumnos por práctica	2	Nº Alumnos por reporte	2	
Nombre del Profesor				
Nombre(s) de Alumno(s)	1. 2.			
Tiempo estimado		Vo. Bo. Profesor		
Comentarios				

2. Objetivo

Comprender el funcionamiento de un decodificador a 7 segmentos para que el estudiante utilice este módulo en prácticas posteriores.

3. Medios a utilizar

Por cada práctica y por cada puesto de laboratorio, los materiales a utilizar es:

Cantidad	Descripción
1	Computadora
1	Tarjeta de desarrollo Basys2 Digilent
1	Software Xilinx ISE® Webpackv14.7

4. Introducción

El display de 7 segmentos es un componente que se utiliza para la representación de caracteres en muchos dispositivos electrónicos (**Figura 1**). El principal componente de este diseño será un decodificador de Hexadecimal a 7 segmentos que es un circuito combinacional utilizado para tomar una entrada Hexadecimal de cuatro bits y proporcionar las salidas que pasaran corriente a través de los segmentos apropiados para desplegar visualmente el dígito decimal.

Código hex	display	abcdefg
0000	0	
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	A	
1011	B	
1100	C	
1101	D	
1110	E	
1111	F	

Tabla 1

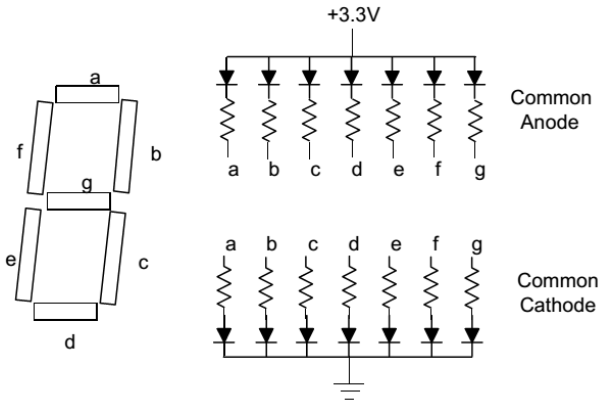


Figura 1

5. Actividades previas

Diseñar un decodificador de hexadecimal a 7 segmentos. Incluya la tabla de verdad con las ecuaciones lógicas y el diagrama con compuertas. Completar en la **Tabla 1** la señal de los segmentos ***abcdefg*** para activar el código correspondiente a cada número hexadecimal.

6. Desarrollo de la práctica

Decodificador de 7 Segmentos

1. Crear un nuevo proyecto en ISE®.
2. Crear el módulo **Hex7Segb** descrito en el siguiente programa.

```
entity Hex7Segb is
     port(
          x     :     in STD_LOGIC_VECTOR(3 downto 0);
          a_to_g     :     out  STD_LOGIC_VECTOR(6 downto 0)
     );
end Hex7Segb;

architecture Behavioral of Hex7Segb is
begin
     P1:  process(x)
```

```
    begin
        case x is
            when x"0" => a_to_g <= "0000001"; --0
            when x"1" => a_to_g <= "1001111"; --1
            when x"2" => a_to_g <= "0010010"; --2
            when x"3" => a_to_g <= "0000110"; --3
            when x"4" => a_to_g <= "1001100"; --4
            when x"5" => a_to_g <= "0100100"; --5
            when x"6" => a_to_g <= "0100000"; --6
            when x"7" => a_to_g <= "0001101"; --7
            when x"8" => a_to_g <= "0000000"; --8
            when x"9" => a_to_g <= "0000100"; --9
            when x"A" => a_to_g <= "0001000"; --A
            when x"B" => a_to_g <= "1100000"; --B
            when x"C" => a_to_g <= "0110001"; --C
            when x"D" => a_to_g <= "1000010"; --D
            when x"E" => a_to_g <= "0110000"; --E
            when others => a_to_g <= "0111000"; --F
        end case;
    end process;
end Behavioral;
```

3. Simule e Implemente el circuito en la tarjeta, verificando los números que se muestran en el display de 7 segmentos.

```
net "x[3]"  loc = "B4" ;
net "x[2]"  loc = "K3" ;
net "x[1]"  loc = "L3" ;
net "x[0]"  loc = "P11";
net "a_to_g[6]" loc = "M12"  ;
net "a_to_g[5]" loc = "L13"  ;
net "a_to_g[4]" loc= "P12"  ;
net "a_to_g[3]" loc = "N11"  ;
net "a_to_g[2]" loc = "N14"  ;
net "a_to_g[1]" loc = "H12"  ;
net "a_to_g[0]" loc = "L14"  ;
```

Nota: Este ejercicio una vez implementado en el kit se visualizará en los 4 displays, esto debido a que ellos están conectados en paralelo y mostrarán la misma información. Para poder visualizarlos de manera independientes es necesario un selector que encienda y apague cada uno de los displays, logrando una ilusión de encendido total.

Multiplexación de Displays de 7 Segmentos

1. Para visualizar distintos datos en los displays se realiza el proceso de multiplexación, es decir se decidirá en qué momento se encenderá uno y se apagaran los demás. El apagado y encendido se logra controlando la alimentación de los displays (transistores), los cuales se muestran en la **Figura 2**.
2. Se crearan varios procesos para lograr la multiplexación tanto de los datos que se quieren visualizar (**mux44**), así mismo de controlar la saturación de los transistores de los displays (**ancode**). Las líneas que decidirán este cambio de un display a otro provienen de un proceso divisor de frecuencia (**clockdiv**) que dará una señal de reloj con una frecuencia más baja para poder visualizar distintos dígitos en los displays según la **Figura 3**.

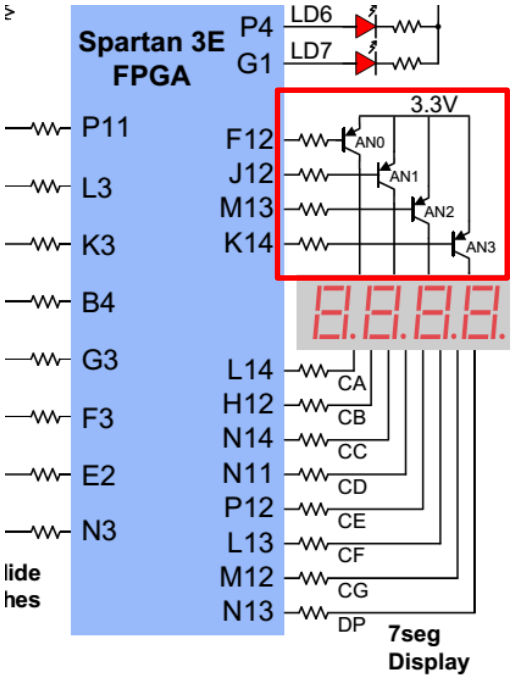


Figura 2

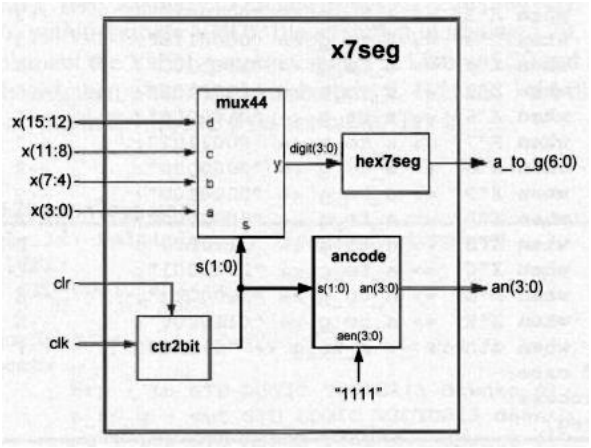


Figura 3

3. Cree un nuevo proyecto llamado **X7seg**.
4. Cree el módulo VHDL llamado **X7Seg** y agregue el siguiente código:

```
entity X7Seg is
     port(
          a_to_g    :    out std_logic_vector(6 downto 0);
          x    :    in std_logic_vector(15 downto 0);
          clr  : in std_logic;
          an   :    out std_logic_vector(3 downto 0);
          clk : in std_logic
     );
end X7Seg;

architecture Behavioral of X7Seg is

signal s: std_logic_vector(1 downto 0);
signal digit: std_logic_vector(3 downto 0);
signal aen: std_logic_vector(3 downto 0);
signal clkdiv: std_logic_vector(19 downto 0);

begin

s <= clkdiv(19 downto 18);
aen <= "1111";

     mux44: process (s,x)
     begin
          case s is
               when "00" => digit <= x(3 downto 0);
               when "01" => digit <= x(7 downto 4);
               when "10" => digit <= x(11 downto 8);
               when others => digit <= x(15 downto 12);
          end case;
     end process;

     hex7seg: process(digit)
     begin
          case digit is
               when X"0" => a_to_g <= "0000001"; --0
               when X"1" => a_to_g <= "1001111"; --1
               when X"2" => a_to_g <= "0010010"; --2
               when X"3" => a_to_g <= "0000110"; --3
               when X"4" => a_to_g <= "1001100"; --4
               when X"5" => a_to_g <= "0100100"; --5
               when X"6" => a_to_g <= "0100000"; --6
               when X"7" => a_to_g <= "0001101"; --7
               when X"8" => a_to_g <= "0000000"; --8
```

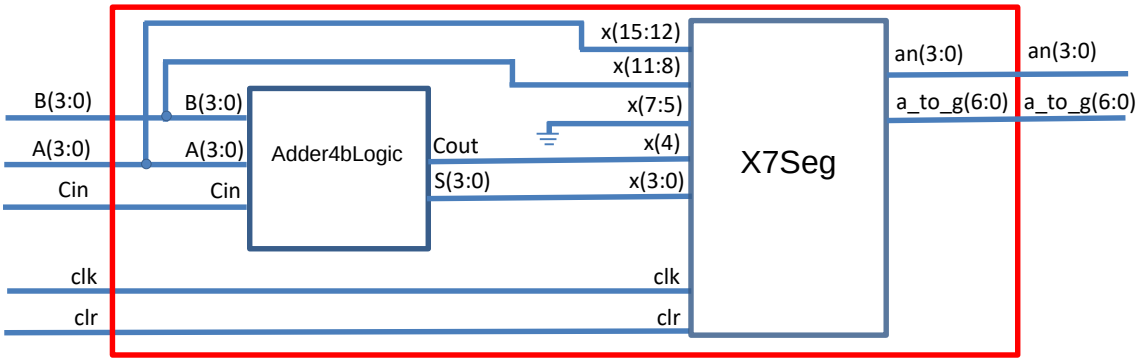
```
                when X"9" => a_to_g <= "0000100"; --9
                when X"A" => a_to_g <= "0001000"; --A
                when X"B" => a_to_g <= "1100000"; --B
                when X"C" => a_to_g <= "0110001"; --C
                when X"D" => a_to_g <= "1000010"; --D
                when X"E" => a_to_g <= "0110000"; --E
                when others => a_to_g <= "0111000"; --F
          end case;
     end process;

-- Digit Select
     ancode: process(s, aen)
     begin
          an <= "1111";
          if aen(conv_integer(s)) = '1' then
               an(conv_integer(s)) <= '0';
          end if;
     end process;

-- Clock Divider
     clkdiv: process(clk,clr)
     begin
          if clr = '1' then
               clkdiv <= (others => '0');
          elsif rising_edge(clk) then
               clkdiv <= clkdiv + 1;
          end if;
     end process;
end Behavioral;
```

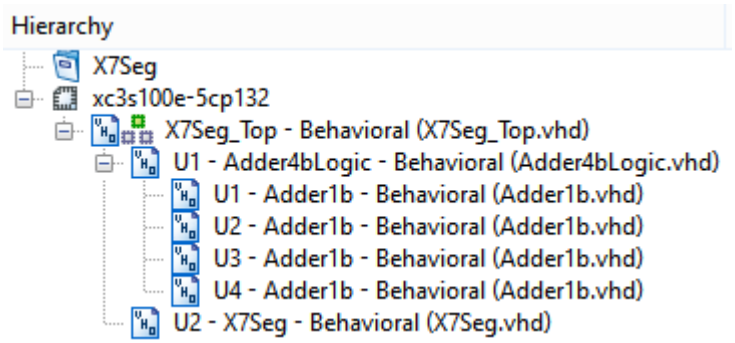
5. Agregue los archivos vhd del sumador de 4 bits (**Adder4bLogic.vhd y Adder1b.vhd**) al proyecto.
6. Cree un Módulo Top llamado **X7Seg_Top**, el cual servirá para visualizar el resultado del sumador completo de 4 bits realizado en la Guía de laboratorio #3. Aquí instanciaremos los tres archivos: **Adder4bLogic.vhd**, **Adder1b.vhd** y el **X7Seg.vhd** según la siguiente tabla de asignación y el siguiente diagrama de bloques.

Display	Resultado a Mostrar	Bits a utilizar de X
Primer Display	Suma (S)	3 al 0
Segundo Display	“000” + Cout	7 al 4
Tercer Display	Numero A	11 al 8
Cuarto Display	Numero B	15 al 12



X7Seg_Top

7. La jerarquía de los archivos del proyecto quedarían de la siguiente manera:



8. Haga el Archivo de simulación correspondiente al Archivo Top para poderlo implementar después en la Basys 2.

7. Actividades propuestas

Práctica

En la práctica de laboratorio se implementó un diseño de un decodificador Hexadecimal a 7 segmentos para visualizar el resultado de la suma de 4 bits. Este diseño mostró el resultado pero en un formato hexadecimal, es decir no lo mostro en el formato decimal que corresponde.

1. Diseñe un módulo que convierta el número binario proporcionado por el sumador completo de 4 bits en un número BCD e impleméntelo en un nuevo proyecto para poder visualizar el resultado en los displays.
2. Haga esto mismo para la resta, multiplicación y división.

Nota: Recuerde que el tamaño del Numero Binario de entrada del Bloque BCD depende del tamaño de los resultados de las operaciones aritméticas realizadas.