Multiplicador Combinacional con Signo

Taller de Sistemas Embebidos

Ishai Gun Roffe

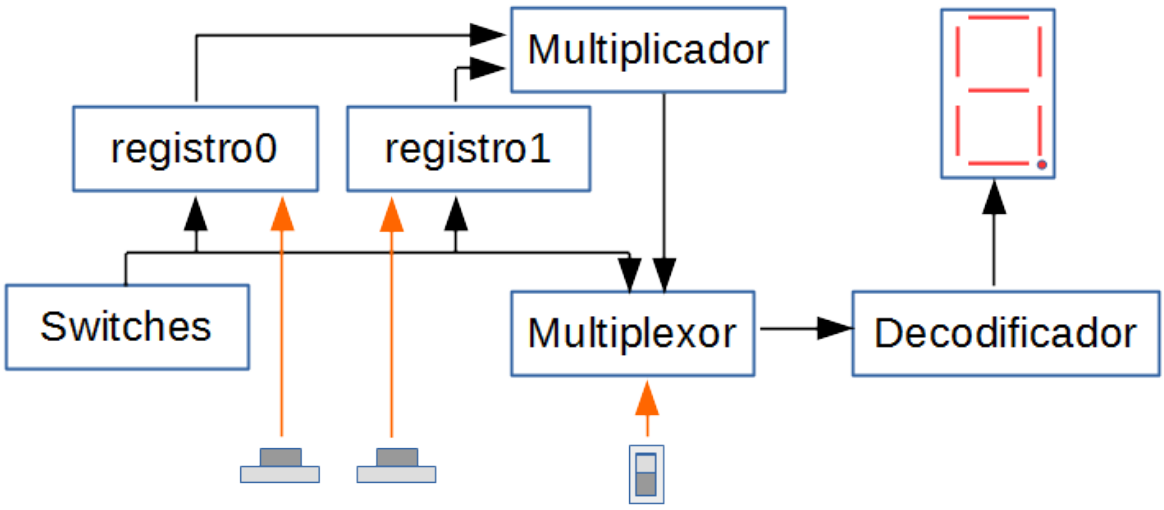
Profesor: Oscar Alvarado Nava

Índice

Descripción del Circuito

El circuito descrito en este reporte es capaz de almacenar dos números de siete bits en formato signo y magnitud (bit más significativo indica el signo y los demás la magnitud) y de mostrar el resultado de su multiplicación.

Los valores son ingresados mediante siete switches, el valor de estos es desplegado constantemente en el display, después es guardado en uno de los registros al presionar el push button correspondiente, al activar el primer switch se despliega el resultado de la multiplicación, cuando se despliega en el display un valor negativo los puntos decimales de este se encienden.



Registro

Estos son los encargados de guardar los valores de entrada a ser multiplicados, cada uno se compone de siete flip flops para guardar cada uno de los bits de cada dato.

d	*entrada*	Valor a guardar en el registro.	7 bits, bit Signo + 6 bits Magnitud
set	*entrada*	Señal para guardar el valor de entrada.	1 bit
reset	*entrada*	Señal de Reset, asigna *0000000* a q.	1 bit
q	*salida*	Regresa el valor guardado en el registro.	7 bits, bit Signo + 6 bits Magnitud

Código

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

entity reg is
    Port ( d   : in  STD_LOGIC_VECTOR
                  (6 downto 0);
           set : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           q   : out  STD_LOGIC_VECTOR
                  (6 downto 0));
end reg;

architecture Behavioral of reg is

component ff
      port ( d : in STD_LOGIC;
             s : in STD_LOGIC;
             r : in STD_LOGIC;
             q : out STD_LOGIC);
end component;

begin
ff0:ff
port map(
      d => d(0),
      s => set,
      r => rst,
      q => q(0)
);

ff1:ff
port map(
      d => d(1),
      s => set,
      r => rst,
      q => q(1)
);

ff2:ff
port map(
      d => d(2),
      s => set,
      r => rst,
      q => q(2)
);

ff3:ff
port map(
      d => d(3),
      s => set,
      r => rst,
      q => q(3)
);

ff4:ff
port map(
      d => d(4),
      s => set,
      r => rst,
      q => q(4)
);

ff5:ff
port map(
      d => d(5),
      s => set,
      r => rst,
      q => q(5)
);

ff6:ff
port map(
      d => d(6),
      s => set,
      r => rst,
      q => q(6)
);
end Behavioral;
```

Flip Flop

Este se encarga de almacenar cada bit, es asíncrono y cuenta con entradas para guardar el bit de entrada (set) y para borrar el contenido del mismo (reset), ambas señales son comunes en todo el registro.

Entradas y salidas:

s	*entrada*	Señal para guardar el valor de entrada.
r	*entrada*	Señal de reset, asigna el valor cero a la salida q.
d	*entrada*	Dato de entrada, el dato que guardara el flip flop.
q	*salida*	Dato de salida, el dato que devuelve el flip flop.

Comportamiento:

s=0		s=1
reset	d	q
1	X	0
0	d	d

Código

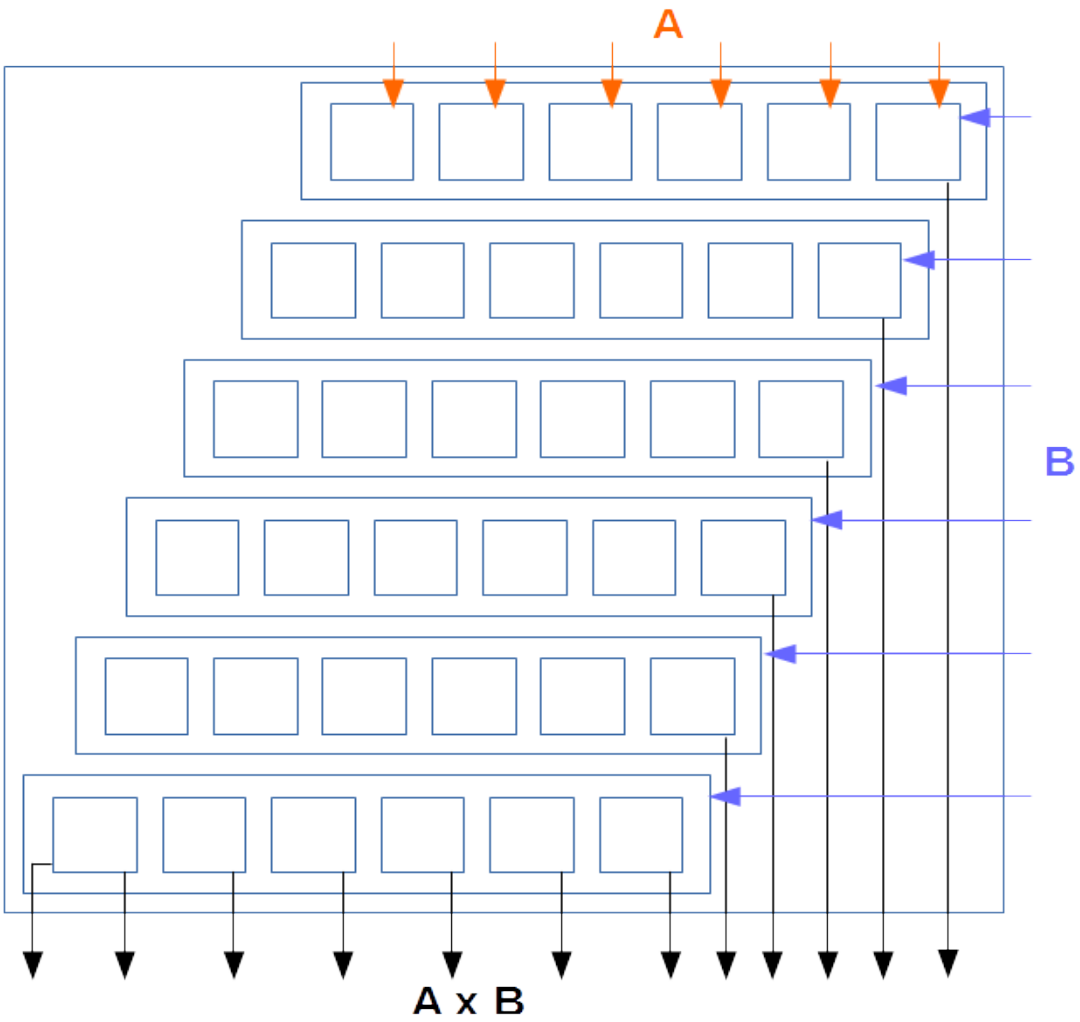
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ff is
    Port ( d : in   STD_LOGIC;
           s : in   STD_LOGIC;
           r : in   STD_LOGIC;
           q : out  STD_LOGIC);
end ff;

architecture Behavioral of ff is

begin

process(s,d)
begin
      if(r='1') then
            q <= '0';
      else
      if (s='1') then
            q <= d;
            end if;
      end if;
end process;

end Behavioral;
```

Multiplicador

El multiplicador es de tipo combinacional, es asíncrono y devuelve un dato de longitud $n*2-1$ siendo n la longitud de los dos operandos, todos los datos son de tipo Signo y Magnitud.

El multiplicador se compone de 36 Unidades de Multiplicación, las cuales se encuentran agrupados en seis filas.

a	*entrada*	Valor a multiplicar	7 bits, Signo + Dato de 6 bits
b	*entrada*		
o	*salida*	resultado	13 bits, Signo + Dato de 12 bits



El multiplicador solo requiere de la magnitud de los operandos, el signo del resultado se determina a partir de los signos de los operandos mediante la siguiente función:

$$S_R=(S_1)\,XOR\,(S_2)$$

Código

```
entity mult is
    Port ( a : in  STD_LOGIC_VECTOR
              (6 downto 0);
           b : in  STD_LOGIC_VECTOR
              (6 downto 0);
           o : out STD_LOGIC_VECTOR
              (12 downto 0));
end mult;

architecture Behavioral of mult is
component fila
      Port ( si : in  STD_LOGIC_VECTOR
                (5 downto 0);
             mi : in   STD_LOGIC_VECTOR
                (5 downto 0);
             qi : in   STD_LOGIC;
             ci : in   STD_LOGIC;
             so : out  STD_LOGIC_VECTOR
                (5 downto 0);
             mo : out  STD_LOGIC_VECTOR
                (5 downto 0);
             qo : out  STD_LOGIC;
             co : out  STD_LOGIC);
end component;

signal sin,cin,qout,mout,m0,m1,m2,m3,m4
        : STD_LOGIC_VECTOR (5 downto 0);
signal a0,a1,a2,a3,a4
        : STD_LOGIC_VECTOR (6 downto 0);
begin

fil0:fila
port map(
      si => sin,
      mi => a(5 downto 0),
      qi => b(0),
      ci => cin(0),
      so => a0(5 downto 0),
      mo => m0,
      qo => qout(0),
      co => a0(6)
);
o(0) <= a0(0);

fil1:fila
port map(
      si => a0(6 downto 1),
      mi => m0,
      qi => b(1),
      ci => cin(1),
      so => a1(5 downto 0),
      mo => m1,
      qo => qout(1),
      co => a1(6)
);
o(1) <= a1(0);

fil2:fila
port map(
      si => a1(6 downto 1),
      mi => m1,
      qi => b(2),
      ci => cin(2),
      so => a2(5 downto 0),
      mo => m2,
      qo => qout(2),
      co => a2(6)
);
o(2) <= a2(0);

fil3:fila
port map(
      si => a2(6 downto 1),
      mi => m2,
      qi => b(3),
      ci => cin(3),
      so => a3(5 downto 0),
      mo => m3,
      qo => qout(3),
      co => a3(6)
);
o(3) <= a3(0);

fil4:fila
port map(
      si => a3(6 downto 1),
      mi => m3,
      qi => b(3),
      ci => cin(4),
      so => a4(5 downto 0),
      mo => m4,
      qo => qout(4),
      co => a4(6)
);
o(4) <= a4(0);

fil5:fila
port map(
      si => a4(6 downto 1),
      mi => m4,
      qi => b(4),
      ci => cin(5),
      so => o(10 downto 5),
      mo => mout,
      qo => qout(5),
      co => o(11)
);
--signo del resultado
process(a,b)
begin
      o(12) <= a(6) XOR b(6);
end process;
end Behavioral;
```

Fila

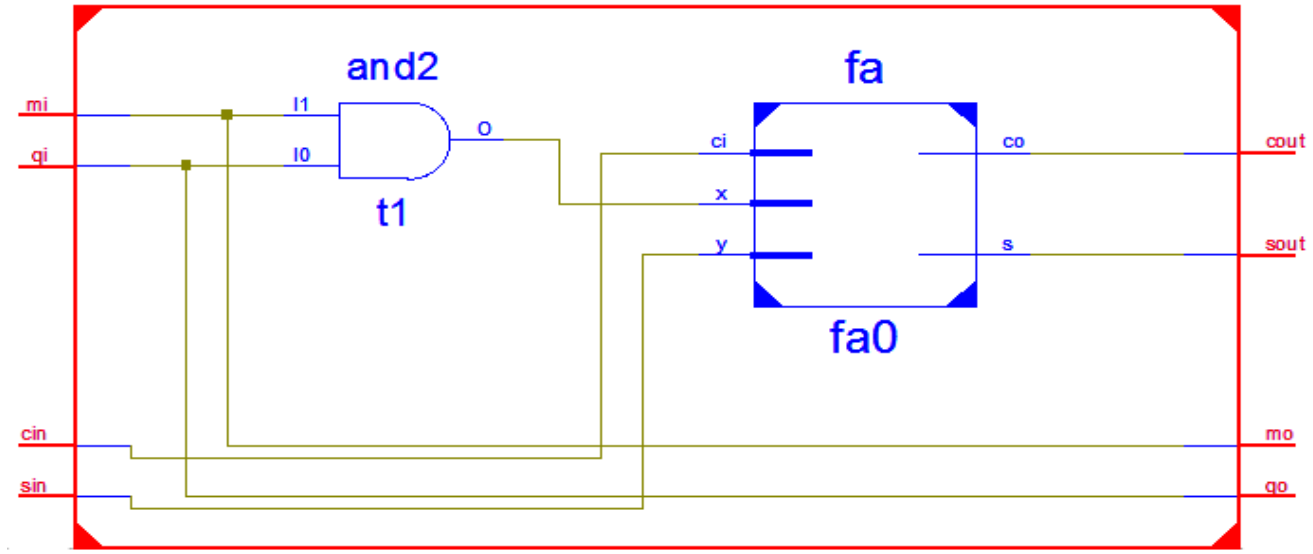
Uno de seis elementos del módulo Multiplicador, cada uno agrupa a seis Unidades de Multiplicación.

Codigo

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

entity fila is
    Port ( si : in   STD_LOGIC_VECTOR
               (5 downto 0);
           mi : in   STD_LOGIC_VECTOR
               (5 downto 0);
           qi : in   STD_LOGIC;
           ci : in   STD_LOGIC;
           so : out  STD_LOGIC_VECTOR
               (5 downto 0);
           mo : out  STD_LOGIC_VECTOR
               (5 downto 0);
           qo : out  STD_LOGIC;
           co : out  STD_LOGIC);
end fila;

architecture Behavioral of fila is

component um
      Port ( sin  : in   STD_LOGIC;
             mi   : in   STD_LOGIC;
             qi   : in   STD_LOGIC;
             cin  : in   STD_LOGIC;
             sout : out  STD_LOGIC;
             mo   : out  STD_LOGIC;
             qo   : out  STD_LOGIC;
             cout : out  STD_LOGIC);
end component;

signal q,c : STD_LOGIC_VECTOR
             (4 downto 0);

begin

u0:um
port map(
      sin  => si(0),
      mi   => mi(0),
      qi   => qi,
      cin  => ci,
      sout => so(0),
      mo   => mo(0),
      qo   => q(0),
      cout => c(0));

u1:um
port map(
      sin  => si(1),
      mi   => mi(1),
      qi   => q(0),
      cin  => c(0),
      sout => so(1),
      mo   => mo(1),
      qo   => q(1),
      cout => c(1));

u2:um
port map(
      sin  => si(2),
      mi   => mi(2),
      qi   => q(1),
      cin  => c(1),
      sout => so(2),
      mo   => mo(2),
      qo   => q(2),
      cout => c(2));

u3:um
port map(
      sin  => si(3),
      mi   => mi(3),
      qi   => q(2),
      cin  => c(2),
      sout => so(3),
      mo   => mo(3),
      qo   => q(3),
      cout => c(3));

u4:um
port map(
      sin  => si(4),
      mi   => mi(4),
      qi   => q(3),
      cin  => c(3),
      sout => so(4),
      mo   => mo(4),
      qo   => q(4),
      cout => c(4));

u5:um
port map(
      sin  => si(5),
      mi   => mi(5),
      qi   => q(4),
      cin  => c(4),
      sout => so(5),
      mo   => mo(5),
      qo   => qo,
      cout => co);

end Behavioral;
```

Unidad de Multiplicación

La Unidad de la matriz de Multiplicación se encarga de multiplicar los bits que recibe de entrada, dar el resultado y sumar y enviar el carry si es necesario. La Unidad se compone de un Full Adder y de una compuerta AND la cual indica si esta celda en particular realizara la operación de multiplicación para los bits de entrada que le corresponden.

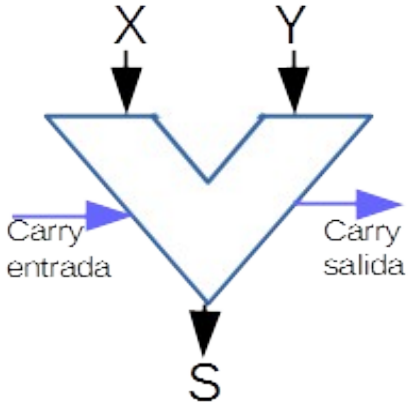


Código:

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

entity um is
    Port ( sin  : in   STD_LOGIC;
           mi   : in   STD_LOGIC;
           qi   : in   STD_LOGIC;
           cin  : in   STD_LOGIC;
           sout : out  STD_LOGIC;
           mo   : out  STD_LOGIC;
           qo   : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end um;

architecture Behavioral of um is

component fa
       Port ( x  : in   STD_LOGIC;
              y  : in   STD_LOGIC;
              ci : in   STD_LOGIC;
              co : out  STD_LOGIC;
              s  : out  STD_LOGIC);
end component;

signal t,u,v,w : STD_LOGIC;

begin

fa0:fa
port map(
      x  => t,
      y  => sin,
      ci => cin,
      co => cout,
      s  => sout
);

qo <= u;
u  <= qi;
mo <= v;
v  <= mi;

process(u,v)
begin
      t <= u AND v;
end process;

end Behavioral;
```

Full Adder

Este es un sumador común de dos datos de longitud de un bit, cuenta con entrada y salida de carry.



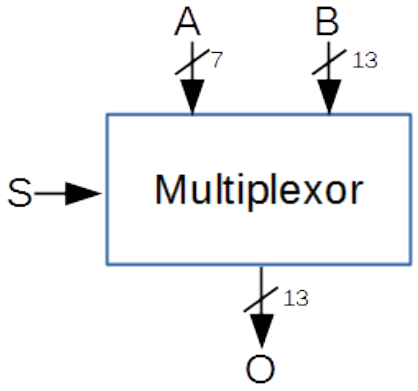
Código

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

entity fa is
    Port ( x  : in   STD_LOGIC;
           y  : in   STD_LOGIC;
           ci : in   STD_LOGIC;
           co : out  STD_LOGIC;
           s  : out  STD_LOGIC);
end fa;

architecture Behavioral of fa is

begin

process(x,y,ci)
begin
      s  <= x xor y xor ci;
      co <= (x and y) or (x and ci) or (y and ci);
end process;

end Behavioral;
```

Multiplexor

Este se encarga de entregar al módulo del decodificador el dato que deseamos desplegar, teniendo como dos opciones el valor que introducimos mediante los Switches y el resultado que nos arroja el multiplicador.

a	*entrada*	Valor de los Switches	7 bits, bit Signo + 6 bits Magnitud
b	*entrada*	Resultado de la multiplicación	13 bits, bit Signo + 12 bits Magnitud
s	*entrada*	Selección del valor a regresar.	1 bit
o	*salida*	Salida al display	13 bits, bit Signo + 12 bits Magnitud



Cuando se selecciona el valor **a** *(s=0)* el contenido del mismo debe ser extendido a 13 bits con el signo en el bit más significativo.

Código

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

entity mux is
    Port ( a : in   STD_LOGIC_VECTOR
              (6 downto 0);
           b : in   STD_LOGIC_VECTOR
              (12 downto 0);
           s : in   STD_LOGIC;
           o : out  STD_LOGIC_VECTOR
              (12 downto 0));
end mux;

architecture Behavioral of mux is
begin

process(a,b,s)
begin
      if s='0' then
            o(12) <= a(6);
            o(11 downto 6) <= "000000";
            o(5 downto 0) <= a(5 downto
0);
      else
            o <= b;
      end if;
end process;

end Behavioral;
```

Decodificador

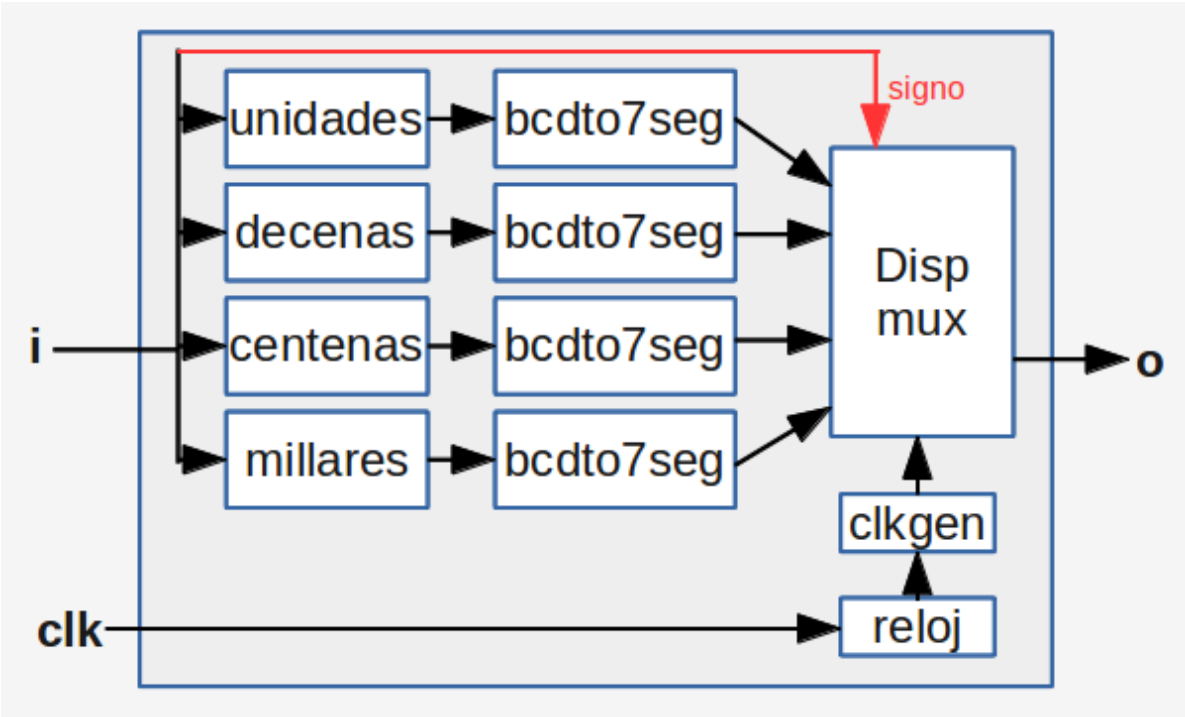
Este módulo se encarga recibir un valor de entrada de 13 bit de formato signo y magnitud y de desplegarlo en cuatro displays de siete segmentos, sin embargo los displays son de ánodo común, por lo que se tienen que multiplexar las salidas lo suficientemente rápido para que de la ilusión de que los cuatro dígitos se encuentran encendidos al mismo tiempo.

Los números negativos se expresan encendiendo el punto de los displays.

clk	*entrada*	Entrada de reloj	1 bit, 100 Mhz
i	*entrada*	Entrada del valor	13 bit, bit Signo + 12 bit Magnitud
o	*salida*	Salida al Display	12 bit, siete segmentos multiplexada

	Reloj			
	t	**t+1**	**t+2**	**t+3**
Salida Segmentos	Unidades	Decenas	Centenas	Millares
Salida Ánodos	0111	1011	1101	1110

Ejemplo: “1234” 



Código

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

entity decod is
    Port ( clk : in  STD_LOGIC;
           i   : in  STD_LOGIC_VECTOR
                 (12 downto 0);
           o   : out STD_LOGIC_VECTOR
                 (11 downto 0));
end decod;

architecture Behavioral of decod is

component bcdto7seg is
       Port ( din : in  STD_LOGIC_VECTOR
                    (3 downto 0);
           todisp : out STD_LOGIC_VECTOR
                    (6 downto 0));
end component;

component reloj is
      Port ( cin  : in  STD_LOGIC;
             cout : out STD_LOGIC);
end component;

component clkgen is
      Port ( clk : in  STD_LOGIC;
             c1  : out STD_LOGIC;
             c2  : ou  STD_LOGIC);
end component;

component dispmux is
       Port ( clk : in STD_LOGIC_VECTOR
                    (1 downto 0);
              a   : in STD_LOGIC_VECTOR
                    (6 downto 0);
              b   : in STD_LOGIC_VECTOR
                    (6 downto 0);
              c   : in STD_LOGIC_VECTOR
                    (6 downto 0);
              d   : in STD_LOGIC_VECTOR
                    (6 downto 0);
              neg : in STD_LOGIC;
              disp: out STD_LOGIC_VECTOR
                    (11 downto 0));
end component;

component unidades is
       Port ( din : in  STD_LOGIC_VECTOR
                    (11 downto 0);
             dout : out STD_LOGIC_VECTOR
                    (3 downto 0));
end component;

component decenas is
       Port ( din : in  STD_LOGIC_VECTOR
                    (11 downto 0);
             dout : out  STD_LOGIC_VECTOR
                    (3 downto 0));
end component;

component centenas is
       Port ( din : in  STD_LOGIC_VECTOR
                    (11 downto 0);
              dout : out STD_LOGIC_VECTOR
                    (3 downto 0));
end component;

component millares is
       Port ( din : in  STD_LOGIC_VECTOR
                    (11 downto 0);
              dout : out STD_LOGIC_VECTOR
                    (3 downto 0));
end component;

signal r : STD_LOGIC;
signal r1 : STD_LOGIC_VECTOR
            (1 downto 0);
signal b0,b1,b2,b3 : STD_LOGIC_VECTOR
            (3 downto 0);
signal d0,d1,d2,d3 : STD_LOGIC_VECTOR
            (6 downto 0);

begin

uni:unidades
port map(
      din  => i(11 downto 0),
      dout => b0
);

dec:decenas
port map(
      din  => i(11 downto 0),
      dout => b1
);

cen:centenas
port map(
      din  => i(11 downto 0),
      dout => b2
);

mil:millares
port map(
      din  => i(11 downto 0),
      dout => b3
);

bcd0:bcdto7seg
port map(
      din    => b0,
      todisp => d0
);
```

```
bcd1:bcdto7seg
port map(
      din    => b1,
      todisp => d1
);

bcd2:bcdto7seg
port map(
      din    => b2,
      todisp => d2
);

bcd3:bcdto7seg
port map(
      din    => b3,
      todisp => d3
);

mux:dispmux
port map(
      clk   => r1,
      a     => d0,
      b     => d1,
      c     => d2,
      d     => d3,
      neg   => i(12),
      disp  => o
);

rel:reloj
port map(
      cin   => clk,
      cout  => r
);

ckg:clkgen
port map(
      clk => r,
      c1  => r1(0),
      c2  => r1(1)
);

end Behavioral;
```

Unidades

Recibe el dato a desplegar y extrae de este unicamente las unidades.

din	*entrada*	Valor a desplegar	12 bits
dout	*salida*	Unidades	4 bits, BCD

Ejemplo: **din** = *"000111000100"* (452) → **dout** = *"0010"* (2)

Código

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

entity unidades is
    Port ( din  : in  STD_LOGIC_VECTOR (11 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end unidades;

architecture Behavioral of unidades is

signal o : STD_LOGIC_VECTOR(11 downto 0);

begin

process(din)
variable n : unsigned (11 downto 0);
begin
n := unsigned(din);
n := n mod "000000001010";
o <= std_logic_vector(n);
end process;

dout <= o(3 downto 0);

end Behavioral;
```

Decenas

Recibe el dato a desplegar y extrae de este unicamente las decenas.

din	*entrada*	Valor a desplegar	12 bits
dout	*salida*	Decenas	4 bits, BCD

Ejemplo: **din** = *"001111110100"* (1012) → **dout** = *"0001"* (1)

Código

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

entity decenas is
    Port ( din  : in  STD_LOGIC_VECTOR (11 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end decenas;

architecture Behavioral of decenas is

signal o : STD_LOGIC_VECTOR (11 downto 0);

begin

process(din)
variable n : unsigned (11 downto 0);
begin
n := unsigned(din);
n := n/10;
n := n mod "000000001010";
o <= std_logic_vector(n);
end process;

dout <= o(3 downto 0);

end Behavioral;
```

Centenas

Recibe el dato a desplegar y extrae de este unicamente las centenas.

din	*entrada*	Valor a desplegar	12 bits
dout	*salida*	Centenas	4 bits, BCD

Ejemplo: **din** = *"000101011110"* (350) → **dout** = *"0011"* (3)

Código

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

entity centenas is
    Port ( din  : in  STD_LOGIC_VECTOR (11 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end centenas;

architecture Behavioral of centenas is

signal o : STD_LOGIC_VECTOR (11 downto 0);

begin

process(din)
variable n : unsigned (11 downto 0);
begin
n := unsigned(din);
n := n/100;
n := n mod "000000001010";
o <= std_logic_vector(n);
end process;

dout <= o(3 downto 0);

end Behavioral;
```

Millares

Recibe el dato a desplegar y extrae de este unicamente los millares.

din	*entrada*	Valor a desplegar	12 bits
dout	*salida*	Millares	4 bits, BCD

Ejemplo: **din** = *"011111111111"* (2047) → **dout** = *"0010"* (2)

Código

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

entity millares is
    Port ( din  : in  STD_LOGIC_VECTOR (11 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end millares;

architecture Behavioral of millares is

signal o : STD_LOGIC_VECTOR (11 downto 0);

begin

process(din)
variable n : unsigned (11 downto 0);
begin
n := unsigned(din);
n := n/1000;
n := n mod "000000001010";
o <= std_logic_vector(n);
end process;

dout <= o(3 downto 0);

end Behavioral;
```

BCD a Display de 7 Segmentos

Este modulo se encarga de traducir el dato BCD que recibe a un dato que el ser humano pueda leer facilmente en el display de siete segmentos.

din	*entrada*	Digito a desplegar	4 bits, BCD
todisp	*salida*	Salida al display	7 bits

Código

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

entity bcdto7seg is
    Port ( din    : in  STD_LOGIC_VECTOR (3 downto 0);
           todisp : out STD_LOGIC_VECTOR (6 downto 0));
end bcdto7seg;

architecture Behavioral of bcdto7seg is

begin

process(din)
begin
      case din is
            when "0000" => todisp <= "0000001"; --0
            when "0001" => todisp <= "1001111"; --1
            when "0010" => todisp <= "0010010"; --2
            when "0011" => todisp <= "0000110"; --3
            when "0100" => todisp <= "1001100"; --4
            when "0101" => todisp <= "0100100"; --5
            when "0110" => todisp <= "0100000"; --6
            when "0111" => todisp <= "0001111"; --7
            when "1000" => todisp <= "0000000"; --8
            when "1001" => todisp <= "0000100"; --9
            when others => todisp <= "0110000"; --e error
      end case;
end process;

end Behavioral;
```

Reloj

El módulo de Decodificación requiere de una señal de reloj para poder multiplexar los displays, sin embargo la frecuencia de reloj que nos proporciona la tarjeta Nexys es demasiado elevada para poder realizar esta tarea, por lo que utilizamos este módulo para poder reducir esa frecuencia de 100 Mhz a cerca de 100 Hz.

cin	*entrada*	Entrada de reloj.	1 bit, 100 Mhz
cout	*salida*	Salida de reloj.	1 bit, 100 Hz

Código

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

entity reloj is
    Port ( cin  : in  STD_LOGIC;
           cout : out STD_LOGIC);
end reloj;

architecture Behavioral of reloj is

signal x: STD_LOGIC := '0';
signal y: integer range 0 to 9999 := 0;
begin

      rel: process(cin)
      begin
            if rising_edge(cin) then
                  if (y=9999) then
                        x <= NOT(x);
                        y <= 0;
                  else
                        y <= y + 1;
                  end if;
            end if;
      end process;

cout <= x;

end Behavioral;
```

Generador de Ciclos para el Multiplexador del Display

Como se vio en la sección del módulo del Decodificador, el Multiplexador del Display requiere de dos entradas del reloj, esto con el fin de multiplexar $n^2=4$ digitos, Este modulo se encarga de generar una señal de reloj con la mitad de la frecuencia de la suministrada, y juntas se envían al Multiplexador.

clk	entrada	Entrada de reloj.	1 bit, 100Hz
c1	salida	Salida de reloj.	1 bit, 100Hz
c2	salida	Salida de reloj.	1 bit, 50Hz

Código

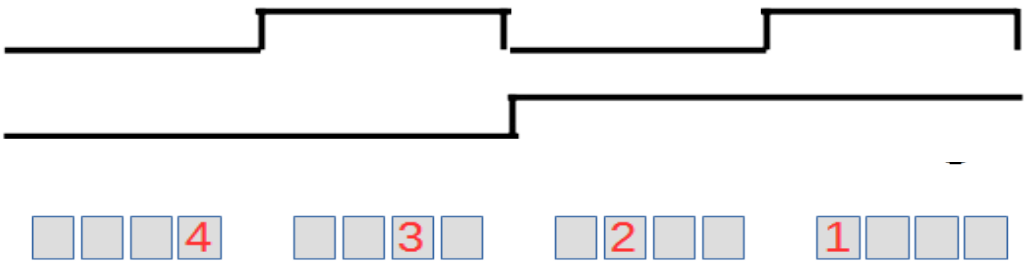
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clkgen is
    Port ( clk : in  STD_LOGIC;
           c1  : out STD_LOGIC;
           c2  : out STD_LOGIC);
end clkgen;

architecture Behavioral of clkgen is

signal c : STD_LOGIC:= '0';

begin

process(clk)
begin
      if (clk='1' and clk'event) then
            c <= not c;
      end if;
end process;

c1 <= clk;
c2 <= c;

end Behavioral;
```

Multiplexador del Display

Este recibe los cuatro dígitos del número a desplegar, la señal del reloj y el bit más significativo del valor inicial a imprimir (su signo). Los bits de salida del multiplexador son los siguientes:

1. La salida al display para el dígito a imprimir. *(7 bits)*
2. El signo del número a imprimir. *(1 bit)*
3. El ánodo del display a encender. *(4 bits)*

clk	*entrada*	Señales de reloj para el multiplexado.	2 bits, 100 Hz y 50 Hz
a	*entrada*	Dígito de unidades.	4 bits, BCD
b	*entrada*	Dígito de decenas.	4 bits, BCD
c	*entrada*	Dígito de centenas.	4 bits, BCD
d	*entrada*	Dígito de millares.	4 bits, BCD
neg	*entrada*	Signo del número.	1 bit
disp	*salida*	Salida a los Displays.	12 bits.



Código

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

entity dispmux is
    Port ( clk : in  STD_LOGIC_VECTOR (1 downto 0);
           a   : in  STD_LOGIC_VECTOR (6 downto 0);
           b   : in  STD_LOGIC_VECTOR (6 downto 0);
           c   : in  STD_LOGIC_VECTOR (6 downto 0);
           d   : in  STD_LOGIC_VECTOR (6 downto 0);
           neg : in  STD_LOGIC;
           disp: out STD_LOGIC_VECTOR (11 downto 0));
end dispmux;

architecture Behavioral of dispmux is
```

```
begin

process(clk,a,b,c,d)
begin

case clk is
      when "00" => disp <= a & NOT(neg) & "0111";
      when "01" => disp <= b & NOT(neg) & "1011";
      when "10" => disp <= c & NOT(neg) & "1101";
      when "11" => disp <= d & NOT(neg) & "1110";
      when others => disp <="000000001111";
end case;

end process;
end Behavioral;
```