UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENERIA Y TECNOLOGIAS AVANZADAS

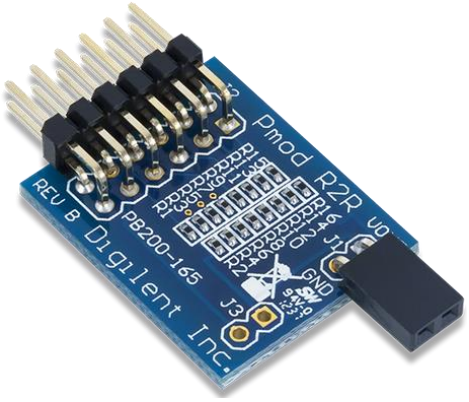
“Generador de frecuencias de 0000 a 9999Hz de 4 tipos de forma de onda”

ALUMNO: ZARAZUA AGUILAR LUIS FERNANDO

GRUPO: 2MM9

PROFESOR: RODRÍGUEZ FUENTES MIGUEL ÁNGEL

MATERIA: DISPOSITIVOS LÓGICOS PROGRAMABLES



Planteamiento del Problema

Esta práctica tiene como objetivo mostrar 4 señales distintas en el osciloscopio con frecuencias desde 0 Hz hasta 9999 Hz, para esto se usó un arreglo R2R de 16 bits que divide ponderadamente el voltaje en potencias de 2. En la resolución del problema se usaron dos memorias una que contenía las señales y otra que tenía los valores del contador para generar correctamente la frecuencia esto para reducir la labor matemática de calcular los valores. Para seleccionar la frecuencia adecuada se cuenta con un selector de 2 bits, un botón de incremento y otro de decremento, con el selector se puede seleccionar si se incrementa/decrementa de 1,10,100 o 1000.

Código Principal

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Principal is
    Port ( clk_nexys : in  STD_LOGIC;
           Sel_Escala : in  STD_LOGIC_VECTOR (1 downto 0);
           Boton_bajar : in  STD_LOGIC;
           Boton_subir : in  STD_LOGIC;
           reset: in STD_LOGIC;
           Sel Senal : in  STD_LOGIC_VECTOR (1 downto 0);
           Vsal : out  STD_LOGIC_VECTOR (15 downto 0);
           Salidas 7segc : out  STD_LOGIC_VECTOR (7 downto 0);
           Control_Disp_7segc: out  STD_LOGIC_VECTOR (3 downto 0));
end Principal;
-- Modulos usados.
architecture Behavioral of Principal is

component debounce4 is
    Port ( clr : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           inp : in  STD_LOGIC_VECTOR(3 downto 0);
           outp : out  STD_LOGIC_VECTOR(3 downto 0));
end component;

component Selector_de_Frecuencia is
    Port ( clk_in : in  STD_LOGIC;
              Escala : in  STD_LOGIC_VECTOR (1 downto 0);
           Boton_Subida : in  STD_LOGIC;
           Boton_Bajada : in  STD_LOGIC;
           Frecuencia : out  STD_LOGIC_VECTOR (13 downto 0));
end component;

component Generador_Func is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           cuentamax : in  STD_LOGIC_VECTOR (17 downto 0);
                                    sel_senal : in  STD_LOGIC_VECTOR (1 downto 0);
           Direccion_Memoria_Senal : out  STD_LOGIC_VECTOR (9 downto 0));
end component;

component Mem_Senales IS
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
end component;

component Mem_Frecuencias IS
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(17 DOWNTO 0));
end component;

component Mem_Digitos IS
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
end component;

component MUX_Freq_0 is
    Port ( Dato : in  STD_LOGIC_VECTOR (15 downto 0);--Dato de la señal generada.
           Freq : in  STD_LOGIC_VECTOR (13 downto 0);--Frecuencia deseada.
           Vout : out  STD_LOGIC_VECTOR (15 downto 0));--Dato de salida.
end component;

component Decodififcador_Digitos is
    Port ( clk : in  STD_LOGIC;
                  Dato : in  STD_LOGIC_VECTOR (13 downto 0);
           Digito_4 : out  STD_LOGIC_VECTOR (7 downto 0);
           Digito_3 : out  STD_LOGIC_VECTOR (7 downto 0);
```

```
          Digito_2 : out  STD_LOGIC_VECTOR (7 downto 0);
          Digito_1 : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component Leds_Display_7 is
    Port ( clkin : in  STD_LOGIC;
           Entrada Disp 1 : in  STD LOGIC VECTOR (7 downto 0);
           Entrada Disp 2 : in  STD LOGIC VECTOR (7 downto 0);
           Entrada Disp 3 : in  STD LOGIC VECTOR (7 downto 0);
           Entrada_Disp_4 : in  STD_LOGIC_VECTOR (7 downto 0);
           Salidas_7seg : out  STD_LOGIC_VECTOR (7 downto 0);
           Control_Disp_7seg : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Salida_ROM_Frecuencias: STD_LOGIC_VECTOR(17 DOWNTO 0);
signal Direccion_ROM_Senales: STD_LOGIC_VECTOR(9 DOWNTO 0);
signal Salida ROM Senales,Digitos: STD LOGIC VECTOR(15 DOWNTO 0);
signal Frecuencia: STD LOGIC_VECTOR(13 DOWNTO 0);
signal Btn Subir: STD LOGIC;
signal Btn_Bajar: STD_LOGIC;
signal Btn_Reset: STD_LOGIC;
signal Botones: STD_LOGIC_VECTOR(3 DOWNTO 0);
signal Salida_BTN: STD_LOGIC_VECTOR(3 DOWNTO 0);
signal Display_1c,Display_2c,Display_3c,Display_4c: STD_LOGIC_VECTOR (7 downto 0);
begin
--    Display 1c<=Frecuencia(7 downto 0);
--           Display 2c<="00"&Frecuencia(13 downto 8);
--           Display 3c<=Salida ROM Senales(15 downto 8);
--           Display_4c<=Salida_ROM_Senales(7 downto 0);

Botones<=(Boton_subir&Boton_bajar&reset&reset);
U0: debounce4 port map( clr => reset,
                        clk => clk_nexys,
                        inp => Botones,
                        outp => Salida_BTN);
     Btn_Subir<=Salida_BTN(3);
           Btn Bajar<=Salida BTN(2);
           Btn_Reset<=Salida_BTN(1) and Salida_BTN(0);
U1: Selector_de_Frecuencia port map( clk_in => clk_nexys,
                                     Escala => Sel_Escala,
                                     Boton_Subida => Btn_Subir,
                                     Boton_Bajada => Btn_Bajar,
                                     Frecuencia => Frecuencia);

ROM1: Mem_Frecuencias port map(clka => clk_nexys,
                               addra => Frecuencia,
                               douta => Salida_ROM_Frecuencias);

U2: Generador_Func port map( clk => clk_nexys,
                             rst => Btn_Reset,
                             cuentamax => Salida_ROM_Frecuencias,
                             sel_senal => Sel_Senal,
                             Direccion_Memoria_Senal => Direccion_ROM_Senales);

ROM2: Mem_Senales port map(clka => clk_nexys,
                           addra => Direccion_ROM_Senales,
                           douta => Salida_ROM_Senales);

U3: MUX_Freq_0 port map( Dato => Salida_ROM_Senales,
                         Freq => Frecuencia,
                         Vout => Vsal);

--ROM3: Mem Digitos port map(clka => clk nexys,
--                           addra => Frecuencia,
--                           douta => Digitos);

U4: Decodififcador_Digitos port map(   clk=>clk_nexys,
                                       Dato => Frecuencia,
                                       Digito_4 => Display_4c,
                                       Digito_3 => Display_3c,
                                       Digito_2 => Display_2c,
                                       Digito_1 => Display_1c);

U5: Leds Display 7 port map( clkin => clk nexys,
                             Entrada Disp 1 => Display_1c,
                             Entrada_Disp_2 => Display_2c,
                             Entrada_Disp_3 => Display_3c,
                             Entrada_Disp_4 => Display_4c,
                             Salidas_7seg => Salidas_7segc,
                             Control_Disp_7seg => Control_Disp_7segc);

end Behavioral;
```

Este código se encarga de unir los bloques funcionales del generador de funciones para que se pueda elegir la frecuencia y la señal deseada y posteriormente pueda ser mostrada en el osciloscopio, así como saber en qué frecuencia se está por medio de los displays. Este código contiene 9 componentes de los cuales 2 son memorias.

Código Anti-rebotes

```
library IEEE;
use IEEE.STD LOGIC 1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity debounce4 is
    Port ( clr : in  STD LOGIC;
           clk : in  STD_LOGIC;
```

```
          inp : in  STD_LOGIC_VECTOR(3 downto 0);
          outp : out  STD_LOGIC_VECTOR(3 downto 0));
end debounce4;

architecture Behavioral of debounce4 is
signal delay1,delay2,delay3: std_logic_vector(3 downto 0);
signal clkdiv: std_logic_vector(17 downto 0);
begin
   process(clk)
            begin
                 if rising_edge(clk) then
                                    clkdiv <= clkdiv +1;
                      end if;
            end process;

   process(clr,clkdiv(17))
            begin
               if clr = '1' then
                      delay1 <= "0000";
                               delay2 <= "0000";
                               delay3 <= "0000";
                    elsif rising_edge(clkdiv(17)) then
                      delay1 <= inp;
                               delay2 <= delay1;
                               delay3 <= delay2;
                      end if;
            end process;
   outp <= delay1 and delay2 and delay3;
end Behavioral;
```

Este código se encarga de eliminar los posibles rebotes en la señal de mandar una señal limpia hacia la máquina de estados que ve si un botón ha sido presionado. Elimina los rebotes de 4 botones.

Código Selector de Frecuencia

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Selector_de_Frecuencia is
    Port ( clk_in : in  STD_LOGIC;
           Escala : in  STD_LOGIC_VECTOR (1 downto 0);
           Boton_Subida : in  STD_LOGIC;
           Boton_Bajada : in  STD_LOGIC;
           Frecuencia : inout  STD_LOGIC_VECTOR (13 downto 0):="00000000000000");
end Selector_de_Frecuencia;

architecture Behavioral of Selector_de_Frecuencia is
signal Contador,Contador2: SIGNED(14 downto 0);--14 Signo 13-0 Dato.
signal Cantidad: SIGNED(10 downto 0);--10 Signo 9-0 Dato.
type state_type is (Inicio,Reposo,Verificacion_1,Esperar_1,Verificacion_2,Esperar_2,Cargar_Frecuencia);
signal state: state_type;
signal clkdiv: std_logic_vector(0 downto 0);
begin
--Divisor de Frecuencia
   process(clk_in)
            begin
                      if rising_edge(clk_in) then
                                    clkdiv <= clkdiv +1;
                      end if;
            end process;
            --Selector de Factor
    with Escala select
    Cantidad <=  "00000000001" when "00",--Declaración del factor X1.
                 "00000001010" when "01",--Declaración del factor X10.
                                          "00001100100" when "10",--Declaración del factor X100.
                 "01111101000" when others;--Declaración del factor X1000.
 --Máquina de Estados.
 process(clkdiv(0),Boton_Subida,Boton_Bajada,Cantidad)
            begin
                      if rising_edge(clkdiv(0)) then--Elige como reloj de la máquina de estados a clkdiv.
                                case state is
                                          when  Inicio =>--Estado de Inicio
                                                Frecuencia<="00001111101000";--Frecuencia Inicial de 1000Hz.
                                                           Contador<="0000011111101000";
                                                           state <=Reposo;
                                          when  Reposo =>--Estado de Inicio
                                                if Boton_Subida='1' then
                                                           Contador2<=Contador+Cantidad;
                                                           state <=Verificacion_1;
                                                elsif Boton_Bajada='1' then
                                                           Contador2<=Contador-Cantidad;
                                                           state <=Verificacion_2;
                                                else
                                                           --Contador2<=Contador2;
                                                           state <=Reposo;
                                                end if;
                                          when  Verificacion_1 =>--Estado de Verificación
                                                if Contador2>9999 then
                                                           Contador<=Contador;
                                                else
                                                           Contador<=Contador2;
                                                end if;
                                                           state<=Esperar_1;
                                          when  Esperar_1 =>--Estado de Espera_1
```

```
                                    if Boton_Subida='0' then
                                        state<=Cargar_Frecuencia;
                                    else
                                        state<=Esperar_1;
                                    end if;
                                when  Verificacion_2 =>--Estado de Inicio
                                    if Contador2<=0 then
                                            if Contador2=0 then
                                             Contador<=(others =>'0');
                                            else
                                             Contador<=Contador;
                                            end if;
                                    else
                                            Contador<=Contador2;
                                    end if;
                                    state<=Esperar_2;
                                when  Esperar 2 =>--Estado de Inicio
                                    if Boton Bajada='0' then
                                            state<=Cargar_Frecuencia;
                                    else
                                        state<=Esperar_2;
                                    end if;
                                when  Cargar_Frecuencia =>--Estado de Inicio
                                    Frecuencia<=std_logic_vector(Contador(13 downto 0));
                                            state <=Reposo;
                            end case;
                    end if;
            end process;
end Behavioral;
```

Este código por medio de una máquina de estados recopila que botón fue presionado y en qué posición estaba el selector, luego verifica si el dato no sobrepasa los límites con la operación de decremento o incremento deseado y si no entonces modifica el registro de frecuencia para su posterior lectura en otros bloques. Esta salida de frecuencia es decodificada por una memoria que asigna el valor correcto para el contador en el generador de señal.

Código para la Generación de la Señal

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Generador_Func is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           cuentamax : in  STD_LOGIC_VECTOR (17 downto 0);
           sel_senal : in  STD_LOGIC_VECTOR (1 downto 0);
           Direccion_Memoria_Senal : out  STD_LOGIC_VECTOR (9 downto 0));
end Generador_Func;

architecture Behavioral of Generador_Func is
signal contador: std_logic_vector(7 downto 0);--Contador de Muestras.
signal masterc: std_logic_vector(17 downto 0);--0-196078 para 1Hz Contador de Intervalo.
begin
      process(clk,rst)
                    begin
                        if rst='1' then
                            contador <="00000000";
                        elsif rising_edge(clk) then
                            masterc<=masterc+1;
                                if masterc>cuentamax then--Indica hasta que número llegar antes de incrementar el contador.
                                    masterc<=(others =>'0');--Resetea el contador que controla la frecuencia.
                                        contador<=contador+1;--Incrementa el contador para acceder a la señal.
                                end if;
                        end if;
                    end process;
                    Direccion_Memoria_Senal<=sel_senal & contador;--Concatenar selector de señal y conteo.
end Behavioral;
```

Este código se encarga se generar un contador ascendente para poder acceder a la memoria correctamente, el tiempo del contador está determinado por la frecuencia ya decodificada en la memoria que nos da hasta que número llegar, este valor de contador es concatenado con el selector de señal para así obtener correctamente a que dirección de la memoria (que contiene los datos de las señales) se va a acceder. Una vez obtenido ese valor de dirección de memoria se accede a la memoria de señales y así se obtiene el dato correcto a mandar.

Código Multiplexor para Frecuencia 0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity MUX_Freq_0 is
    Port ( Dato : in  STD_LOGIC_VECTOR (15 downto 0);--Dato de la señal generada.
           Freq : in  STD_LOGIC_VECTOR (13 downto 0);--Frecuencia deseada.
           Vout : out  STD_LOGIC_VECTOR (15 downto 0));--Dato de salida.
end MUX_Freq_0;

architecture Behavioral of MUX_Freq_0 is

begin
     process(Freq,Dato)
                    begin
        if Freq=0 then
         Vout<=(others =>'0');--Si es 0 la frecuencia conecta a tierra la señal
        else
         Vout<=Dato;--Si la frecuencia es distinta de 0, pasa el dato dado por la memoria de señal.
        end if;
       end process;
end Behavioral;
```

Este código sirve para separar el caso en que la frecuencia es 0 y para evitar que se quede en un voltaje fijo aleatorio, para esto se conecta a tierra por medio de este multiplexor con el valor de 0, en caso de cualquier otra frecuencia deja pasar el dato generado por la memoria.

Código para la decodificación de los dígitos

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
entity Decodififcador_Digitos is
    Port ( clk : in  STD_LOGIC;
                Dato : in  STD_LOGIC_VECTOR (13 downto 0);
           Digito_4 : out  STD_LOGIC_VECTOR (7 downto 0);
           Digito_3 : out  STD_LOGIC_VECTOR (7 downto 0);
           Digito_2 : out  STD_LOGIC_VECTOR (7 downto 0);
           Digito_1 : out  STD_LOGIC_VECTOR (7 downto 0));
end Decodififcador_Digitos;



architecture Behavioral of Decodififcador_Digitos is
type state_type is (Inicio,Recopilar_Datos,Decrementar_Millares,Decrementar_Centenas,Decrementar_Decenas,Cargar_Digitos);
signal state: state_type;
signal Dato_op: STD_LOGIC_VECTOR (13 downto 0);
signal clkdiv: STD_LOGIC_VECTOR (0 downto 0);
signal Dig4,Dig3,Dig2,Dig1: STD_LOGIC_VECTOR (3 downto 0);
signal Dig4b,Dig3b,Dig2b,Dig1b: STD_LOGIC_VECTOR (3 downto 0);
begin
     process(clk)
              begin
                    if rising_edge(clk) then
                                clkdiv <= clkdiv +1;
                    end if;
             end process;


             --Màquina de Estados.
 process(clkdiv(0),Dato)
            begin
                    if rising_edge(clkdiv(0)) then--Elige como reloj de la máquina de estados a clkdiv.
                                case state is
                                        when  Inicio =>--Estado de Inicio
                                                        state <=Recopilar_Datos;
                                        when  Recopilar_Datos =>--Estado de Inicio
                                              Dig1<="0000";
                                                      Dig2<="0000";
                                                      Dig3<="0000";
                                                      Dig4<="0000";
                                                      Dato_op<=Dato;
                                                      state <=Decrementar_Millares;

                                        when  Decrementar_Millares =>--Estado de Decrementar Millares
                                          if Dato_op>=1000 then
                                                        Dato_op<=Dato_op-1000;
                                                        Dig4<=Dig4+1;
                                                        state<=Decrementar_Millares;
                                                else
                                                        Dig4<=Dig4;
                                                        state<=Decrementar_Centenas;
                                                end if;

                                        when  Decrementar_Centenas =>--Estado de Decrementar_Centenas
                                          if Dato_op>=100 then
                                                        Dato_op<=Dato_op-100;
                                                        Dig3<=Dig3+1;
                                                        state<=Decrementar_Centenas;
                                                else
                                                        Dig3<=Dig3;
```

```
                                                    state<=Decrementar_Decenas;
                                                end if;
                                        when  Decrementar_Decenas =>--Estado de Decrementar_Decenas
                                                if Dato_op>=10 then
                                                    Dato_op<=Dato_op-10;
                                                    Dig2<=Dig2+1;
                                                    state<=Decrementar_Centenas;
                                                else
                                                    Dig1<=Dato op(3 downto 0);
                                                    Dig2<=Dig2;
                                                    state<=Cargar_Digitos;
                                                end if;
                                        when  Cargar_Digitos =>--Estado de Cargar_Digitos
                                                Dig1b<=Dig1;
                                                Dig2b<=Dig2;
                                                Dig3b<=Dig3;
                                                Dig4b<=Dig4;
                                                state <=Recopilar_Datos;
                                end case;
                        end if;
                end process;


                with Dig4b select
            Digito_4 <=  not(X"C0") when X"0",--Declaración del dígito "0" para display de 7 segmentos ánodo.
                        not(X"79") when X"1",--Declaración del dígito "1" para display de 7 segmentos ánodo.
                        not(X"A4") when X"2",--Declaración del dígito "2" para display de 7 segmentos ánodo.
                        not(X"30") when X"3",--Declaración del dígito "3" para display de 7 segmentos ánodo.
                        not(X"99") when X"4",--Declaración del dígito "4" para display de 7 segmentos ánodo.
                        not(X"12") when X"5",--Declaración del dígito "5" para display de 7 segmentos ánodo.
                        not(X"83") when X"6",--Declaración del dígito "6" para display de 7 segmentos ánodo.
                        not(X"78") when X"7",--Declaración del dígito "7" para display de 7 segmentos ánodo.
                        not(X"80") when X"8",--Declaración del dígito "8" para display de 7 segmentos ánodo.
                        not(X"18") when X"9",--Declaración del dígito "9" para display de 7 segmentos ánodo.
                        not(X"A7") when X"A",--Declaración del dígito "A" para display de 7 segmentos ánodo.
                        not(X"33") when X"B",--Declaración del dígito "B" para display de 7 segmentos ánodo.
                        not(X"9D") when X"C",--Declaración del dígito "C" para display de 7 segmentos ánodo.
                        not(X"16") when X"D",--Declaración del dígito "D" para display de 7 segmentos ánodo.
                        not(X"87") when X"E",--Declaración del dígito "E" para display de 7 segmentos ánodo.
                        not(X"7F") when others;--Declaración del dígito "F" para display de 7 segmentos ánodo.

                with Dig3b select
            Digito_3 <=  not(X"C0") when X"0",--Declaración del dígito "0" para display de 7 segmentos ánodo.
                        not(X"79") when X"1",--Declaración del dígito "1" para display de 7 segmentos ánodo.
                        not(X"A4") when X"2",--Declaración del dígito "2" para display de 7 segmentos ánodo.
                        not(X"30") when X"3",--Declaración del dígito "3" para display de 7 segmentos ánodo.
                        not(X"99") when X"4",--Declaración del dígito "4" para display de 7 segmentos ánodo.
                        not(X"12") when X"5",--Declaración del dígito "5" para display de 7 segmentos ánodo.
                        not(X"83") when X"6",--Declaración del dígito "6" para display de 7 segmentos ánodo.
                        not(X"78") when X"7",--Declaración del dígito "7" para display de 7 segmentos ánodo.
                        not(X"80") when X"8",--Declaración del dígito "8" para display de 7 segmentos ánodo.
                        not(X"18") when X"9",--Declaración del dígito "9" para display de 7 segmentos ánodo.
                        not(X"A7") when X"A",--Declaración del dígito "A" para display de 7 segmentos ánodo.
                        not(X"33") when X"B",--Declaración del dígito "B" para display de 7 segmentos ánodo.
                        not(X"9D") when X"C",--Declaración del dígito "C" para display de 7 segmentos ánodo.
                        not(X"16") when X"D",--Declaración del dígito "D" para display de 7 segmentos ánodo.
                        not(X"87") when X"E",--Declaración del dígito "E" para display de 7 segmentos ánodo.
                        not(X"7F") when others;--Declaración del dígito "F" para display de 7 segmentos ánodo.

                with Dig2b select
            Digito_2 <=  not(X"C0") when X"0",--Declaración del dígito "0" para display de 7 segmentos ánodo.
                        not(X"79") when X"1",--Declaración del dígito "1" para display de 7 segmentos ánodo.
                        not(X"A4") when X"2",--Declaración del dígito "2" para display de 7 segmentos ánodo.
                        not(X"30") when X"3",--Declaración del dígito "3" para display de 7 segmentos ánodo.
                        not(X"99") when X"4",--Declaración del dígito "4" para display de 7 segmentos ánodo.
                        not(X"12") when X"5",--Declaración del dígito "5" para display de 7 segmentos ánodo.
                        not(X"83") when X"6",--Declaración del dígito "6" para display de 7 segmentos ánodo.
                        not(X"78") when X"7",--Declaración del dígito "7" para display de 7 segmentos ánodo.
                        not(X"80") when X"8",--Declaración del dígito "8" para display de 7 segmentos ánodo.
                        not(X"18") when X"9",--Declaración del dígito "9" para display de 7 segmentos ánodo.
                        not(X"A7") when X"A",--Declaración del dígito "A" para display de 7 segmentos ánodo.
                        not(X"33") when X"B",--Declaración del dígito "B" para display de 7 segmentos ánodo.
                        not(X"9D") when X"C",--Declaración del dígito "C" para display de 7 segmentos ánodo.
                        not(X"16") when X"D",--Declaración del dígito "D" para display de 7 segmentos ánodo.
                        not(X"87") when X"E",--Declaración del dígito "E" para display de 7 segmentos ánodo.
                        not(X"7F") when others;--Declaración del dígito "F" para display de 7 segmentos ánodo.

                with Dig1b select
            Digito_1 <=  not(X"C0") when X"0",--Declaración del dígito "0" para display de 7 segmentos ánodo.
                        not(X"79") when X"1",--Declaración del dígito "1" para display de 7 segmentos ánodo.
                        not(X"A4") when X"2",--Declaración del dígito "2" para display de 7 segmentos ánodo.
                        not(X"30") when X"3",--Declaración del dígito "3" para display de 7 segmentos ánodo.
                        not(X"99") when X"4",--Declaración del dígito "4" para display de 7 segmentos ánodo.
                        not(X"12") when X"5",--Declaración del dígito "5" para display de 7 segmentos ánodo.
                        not(X"83") when X"6",--Declaración del dígito "6" para display de 7 segmentos ánodo.
                        not(X"78") when X"7",--Declaración del dígito "7" para display de 7 segmentos ánodo.
                        not(X"80") when X"8",--Declaración del dígito "8" para display de 7 segmentos ánodo.
                        not(X"18") when X"9",--Declaración del dígito "9" para display de 7 segmentos ánodo.
                        not(X"A7") when X"A",--Declaración del dígito "A" para display de 7 segmentos ánodo.
                        not(X"33") when X"B",--Declaración del dígito "B" para display de 7 segmentos ánodo.
                        not(X"9D") when X"C",--Declaración del dígito "C" para display de 7 segmentos ánodo.
                        not(X"16") when X"D",--Declaración del dígito "D" para display de 7 segmentos ánodo.
                        not(X"87") when X"E",--Declaración del dígito "E" para display de 7 segmentos ánodo.
                        not(X"7F") when others;--Declaración del dígito "F" para display de 7 segmentos ánodo.
end Behavioral;
```

Este código sirve para codificar la frecuencia en dígitos que puedan ser mostrados en los displays de 7 segmentos, para esto se realizó máquina de estados que separa en millares, centenas, decenas y unidades, posteriormente cada digito individual es decodificado en su valor para 7 segmentos y enviado al multiplexor que prende los displays.

Código del Display

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Leds_Display_7 is
    Port ( clkin : in  STD_LOGIC;
                  Entrada_Disp_1 : in  STD_LOGIC_VECTOR (7 downto 0);
           Entrada_Disp_2 : in  STD_LOGIC_VECTOR (7 downto 0);
           Entrada_Disp_3 : in  STD_LOGIC_VECTOR (7 downto 0);
           Entrada_Disp_4 : in  STD_LOGIC_VECTOR (7 downto 0);
           Salidas_7seg : out  STD_LOGIC_VECTOR (7 downto 0);
           Control_Disp_7seg : out  STD_LOGIC_VECTOR (3 downto 0));
end Leds_Display_7;

architecture Behavioral of Leds_Display_7 is
signal clkdiv: STD_LOGIC_VECTOR (16 downto 0);
signal contador_disp: STD_LOGIC_VECTOR (1 downto 0):="00";
begin
     process(clkin)
              begin
                        if rising_edge(clkin) then
                                    clkdiv <= clkdiv +1;
                        end if;
              end process;
              process(clkdiv(16),contador_disp)
              begin
      if rising_edge(clkdiv(16)) then
                 contador_disp<=contador_disp+1;
                 if contador_disp=0 then
                             Control_Disp_7seg<="0111";
           Salidas_7seg<=not(Entrada_Disp_1);
                        elsif contador_disp=1 then
                             Control_Disp_7seg<="1011";
                                    Salidas_7seg<=not(Entrada_Disp_2);
                        elsif contador_disp=2 then
                             Control_Disp_7seg<="1101";
                                    Salidas_7seg<=not(Entrada_Disp_3);
                        else
                             Control_Disp_7seg<="1110";
                                    Salidas_7seg<=not(Entrada_Disp_4);
                        end if;
               end if;
              end process;
end Behavioral;
```

Este código se implementa para mostrar los datos decodificados de los dígitos en el display de 7 segmentos.

Código en Matlab para generar los valores del contador

```
clc, clear, close all
nm=2^8;%%numero de muestras.
nm=nm-1;
freq_ls=1/nm;
valor_cont_ls=50e6*freq_ls;
for i=1:1:9999
     Valores_Freq(i)=round(valor_cont_ls/i);
end
Valores_Freq=[Valores_Freq(1),Valores_Freq];
t=0:1:9999;
stem(t,Valores_Freq);
Palabra=dec2hex(Valores_Freq,4);
outfile='Val_Cont.coe';
s = fopen(outfile,'w+');  %opens the output file
fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n','; .COE file with hex coefficients ');
fprintf(s,'%s\n','memory initialization radix=16;');
fprintf(s,'%s\n','memory initialization_vector=');
for i=1:length(Valores_Freq)-1
    fprintf(s,'%c',Palabra(i,:));
    fprintf(s,'%c',',');
end
fprintf(s,'%c',Palabra(i+1,:));
fprintf(s,'%c',';');
```

Este código se encarga de generar el archivo .coe con los valores del contador para cada frecuencia, para esto se considera de cuantas muestras se quiere la señal y en base a esto y la frecuencia del reloj de la Nexys se obtiene un valor para cada frecuencia, como el contador es entero hay pérdida en los datos generando para algunas frecuencias valores iguales en el contador.

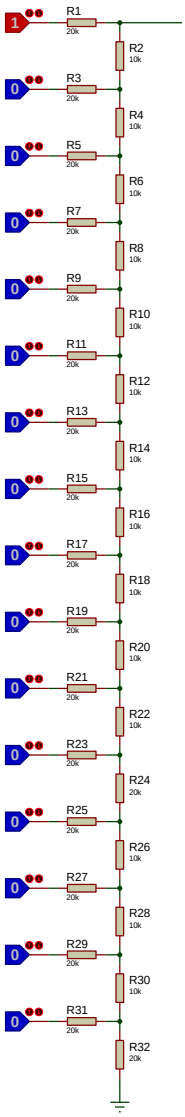
Código en Matlab para generar los valores de la señal

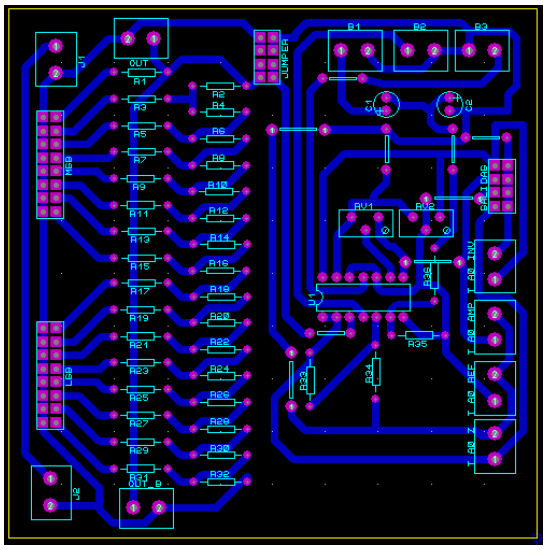
```
clc, clear all, close all
%8 bits
bits=16;
Voltaje=3.3;
n=2^8;%Número de muestras
n=n-1;
t=0:360/n:360;
Vec_1=0:1:n;
%%%Señal Rampa
y1=(t/360*Voltaje);
Valor1(length(t))=0;
for i=1:length(t)
    x=y1(i);
    for j=1:1:bits
     if(x>=Voltaje/(2^j))
       x=x-Voltaje/(2^j);
       Valor1(i)=Valor1(i)+2^(bits-j);
     end
    end
end
%%%Señal Seno
y2=(Voltaje+sind(t)*Voltaje)/2;%Señal Seno
Valor2(length(t))=0;
for i=1:length(t)
    x=y2(i);
    for j=1:1:bits
     if(x>=Voltaje/(2^j))
       x=x-Voltaje/(2^j);
       Valor2(i)=Valor2(i)+2^(bits-j);
     end
    end
end
%%Señal exponencial
 for i=1:length(t) %Exponencial.
     y3(i)=Voltaje*exp((t(i))/36)/22e3;
 end
 Valor3(length(t))=0;
for i=1:length(t)
    x=y3(i);
    for j=1:1:bits
     if(x>=Voltaje/(2^j))
       x=x-Voltaje/(2^j);
       Valor3(i)=Valor3(i)+2^(bits-j);
     end
    end
end
%%Señal libre
 for i=1:length(t) %Circulo
   if i<=length(t)/2;
     y4(i)=Voltaje*0.1+Voltaje*0.9*(1+sqrt(90^2-(t(i)-90)^2)/90)/2-(Voltaje+sind(15*t(i))*Voltaje)/20;
   else
     y4(i)=Voltaje*0.1+Voltaje*0.9*(1-sqrt(90^2-(t(i)-270)^2)/90)/2-(Voltaje+sind(30*t(i))*Voltaje)/20;
   end
 end
% for i=1:length(t) %Triangular-Seno
%    if i<length(t)/2;
%      y4(i)=Voltaje*(t(i)/180);
%    else
%      y4(i)=Voltaje*(1+sind(t(i)-90))/2;
%    end
% end
Valor4(length(t))=0;
for i=1:length(t)
    x=y4(i);
    for j=1:1:bits
     if(x>=Voltaje/(2^j))
       x=x-Voltaje/(2^j);
       Valor4(i)=Valor4(i)+2^(bits-j);
     end
    end
end
figure(1)
axis equal, subplot(2,2,1),plot(t,y1)
subplot(2,2,2),stem(t,Valor1)
subplot(2,2,3),plot(Vec_1,y1)
subplot(2,2,4),stem(Vec_1,Valor1)
figure(2)
axis equal, subplot(2,2,1),plot(t,y2)
subplot(2,2,2),stem(t,Valor2)
subplot(2,2,3),plot(Vec_1,y2)
subplot(2,2,4),stem(Vec_1,Valor2)
figure(3)
axis equal, subplot(2,2,1),plot(t,y3)
subplot(2,2,2),stem(t,Valor3)
subplot(2,2,3),plot(Vec_1,y3)
subplot(2,2,4),stem(Vec_1,Valor3)
figure(4)
axis equal, subplot(2,2,1),plot(t,y4)
subplot(2,2,2),stem(t,Valor4)
subplot(2,2,3),plot(Vec_1,y4)
for i=1:1:length(t)
   Valor4b(2*i-1)=Valor4(i);
   Angulo(2*i-1)=t(i);
   if i<length(t)
   Angulo(2*i)=t(i+1);
   Valor4b(2*i)=Valor4(i);
   end
end
subplot(2,2,4),plot(Angulo,Valor4b)
%%Generacion del punto coe.
Senales=[Valor1,Valor2,Valor3,Valor4];
Palabra=dec2hex(Senales);
outfile='Senales.coe';
s = fopen(outfile,'w+');  %opens the output file
fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n',' .COE file with hex coefficients ');
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');
for i=1:length(Senales)-1
    fprintf(s,'%c',Palabra(i,:));
```

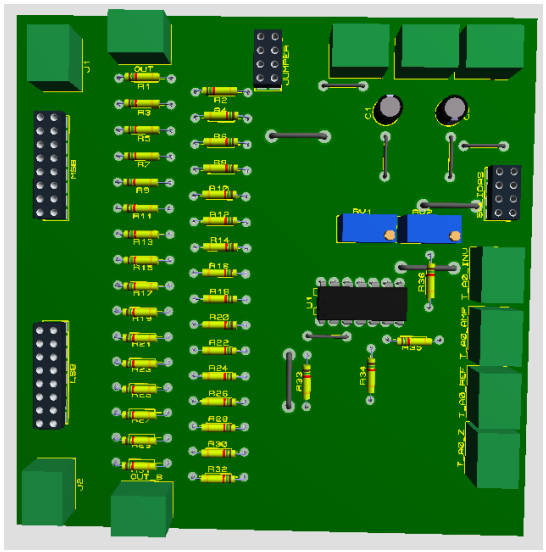
```
    fprintf(s,'%c',',');
end
fprintf(s,'%c',Palabra(i+1,:));
fprintf(s,'%c',';');
```

En este código se genera el archivo para las 4 señales distintas a las cuales se les asigna una función en el tiempo para que se puedan representar más fácilmente, posteriormente se encuentra el valor correcto por medio de un comparador y un ciclo while que decide qué valor digital tiene la señal y este se convierte a hexadecimal, posteriormente se concatenan las 4 señales en una arreglo y se escribe el archivo .coe de la memoria de señal.

Arreglo R2R







Señales Mostradas