Universidad Politécnica de Madrid
ETSI de Telecomunicación
Departamento de Ingeniería Electrónica

Circuitos Electrónicos (CELT)

Curso 2016-2017

Memoria final

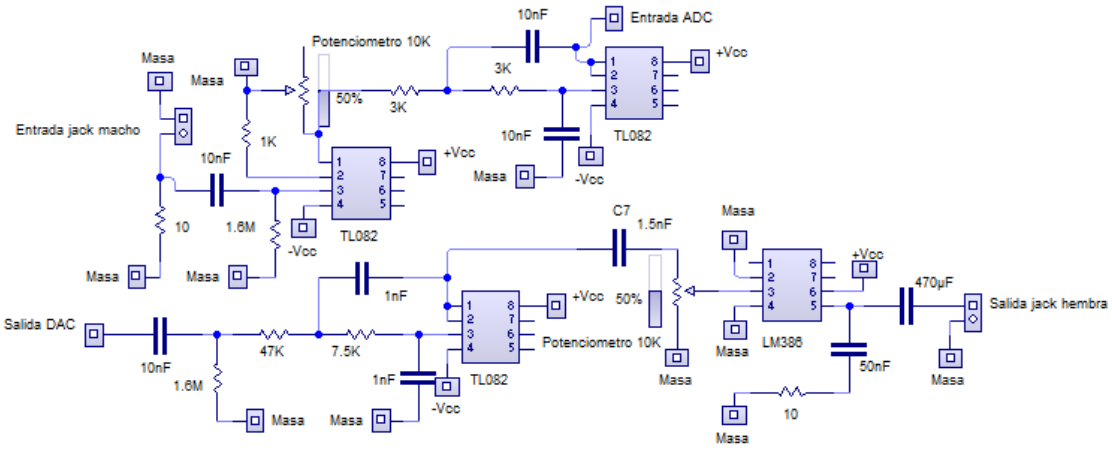
	Apellidos	Nombre
Alumno 1	Fernández Fernández	Pablo
Alumno 2	Romero Pomar	Santiago

Código de pareja	XT-13

(El contenido de esta memoria debe ajustarse exactamente a los bloques analógicos y digitales de la práctica que se presenta)

1. ESQUEMA COMPLETO DEL CIRCUITO ANALÓGICO

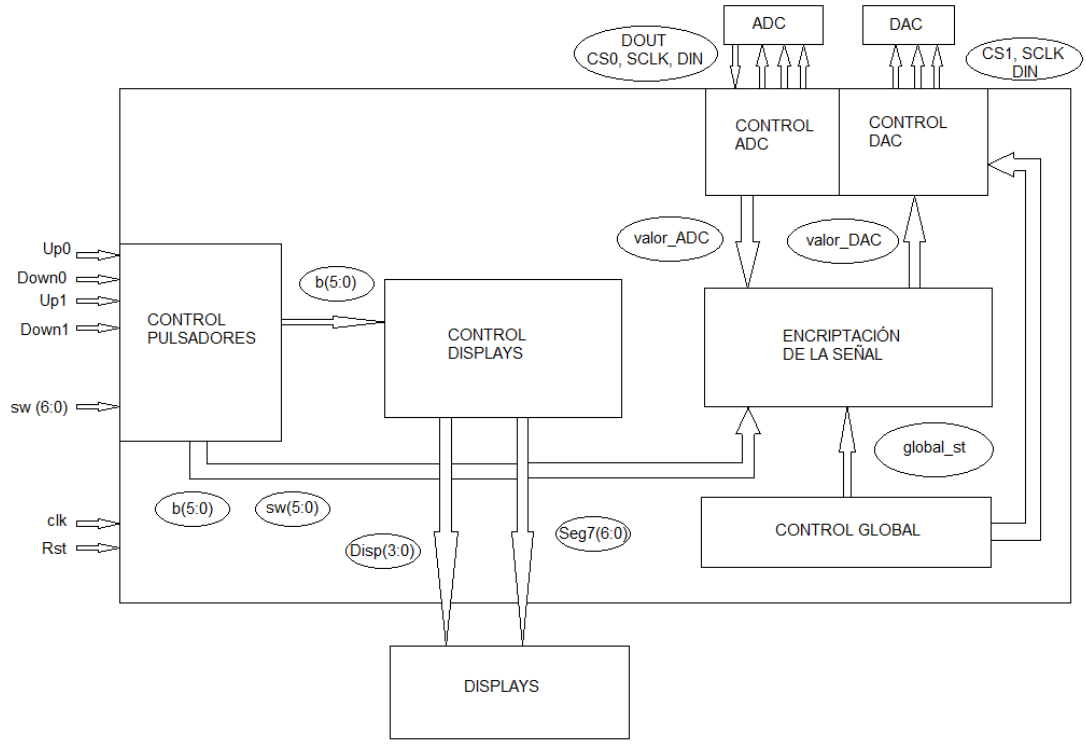
Incluya en esta página un diagrama completo del circuito analógico con los valores de todos los componentes.



2. ESQUEMA COMPLETO DEL CIRCUITO DIGITAL

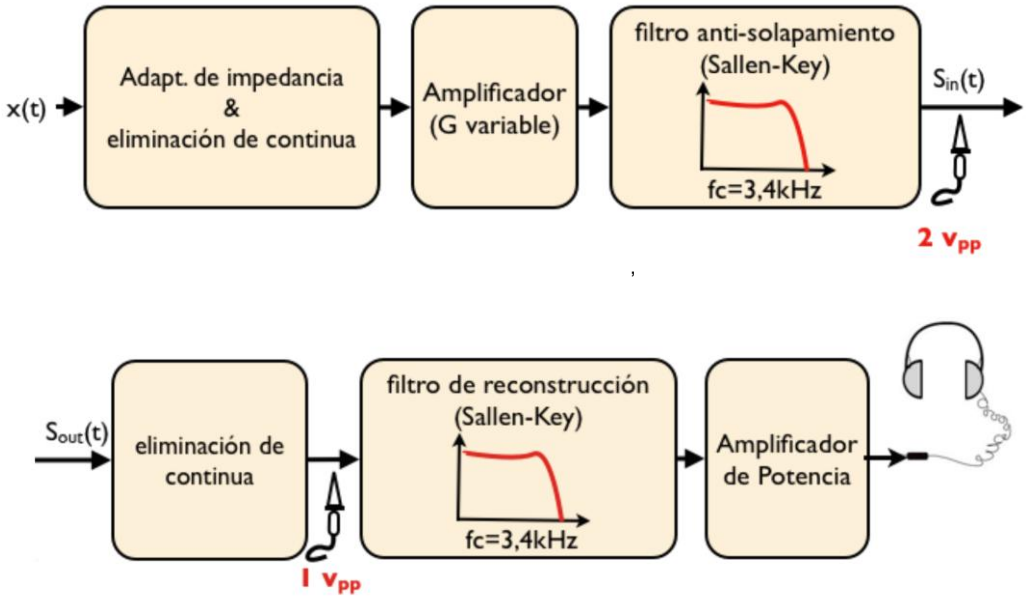
Incluya en esta página un diagrama completo del circuito digital. Represente cada bloque con sus entradas y salidas, y las interconexiones entre ellos.

Lo que se encuentra dentro de los módulos de control de pulsadores y control de displays está detallado en un esquema en la parte digital, dentro de cada uno de sus respectivos apartados.



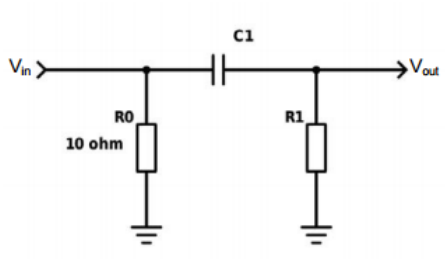
3. DISEÑO DETALLADO DEL CIRCUITO ANALÓGICO

Por cada etapa debe justificar el diseño de los componentes y los criterios de diseño empleados.



3.1 Adaptación de impedancias y eliminación de la componente continua

(a) Justifique la elección de la resistencia de entrada escogida. (b) Detalle el cálculo de los valores del resto de componentes. (c) Especifique los valores teóricos de ganancia del bloque de eliminación de continua a 5 Hz, 10 Hz y 100 Hz. (d) Compare los valores teóricos de ganancia con los valores medidos



a)b) Conociendo la función de transferencia del filtro y la frecuencia de corte del filtro paso alto:

$$Hhp = \frac{s}{s + wc} = \frac{s}{s + \frac{1}{RC}}$$

$$wc = 2\pi(10)\,{}^{rad}/_{seg}$$

Y dando un valor al condensador (intencionadamente):

Siendo C = 10nF:

$$wc = \frac{1}{RC} ; R = \frac{1}{Cwc} ; \boldsymbol{R \cong 1.6M\Omega}$$

Así, los valores (comerciales) obtenidos son:

R1 = 1.6MΩ

C1 = 10nF

c) Ganancia del filtro a las diferentes frecuencias.

Sabiendo que la ganancia se calcula como:

$$G = 10\log\left(|H(jw)|^2\right)$$

Obtenemos la ganancia en las siguientes frecuencias:

f = 5HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{(2\pi(5))^2}{(2\pi(5))^2 + wc^2}\right) = \boldsymbol{-6.9897dB}$$

f = 10HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{(2\pi(10))^2}{(2\pi(10))^2 + wc^2}\right) = \boldsymbol{-3.0103dB}$$

f = 100HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{(2\pi(100))^2}{(2\pi(100))^2 + wc^2}\right) = \boldsymbol{-0.0432dB}$$

d) Tras la realización de una medida a este módulo en nuestro circuito, y mediante el cálculo de la ganancia como:

$$20log\left[\frac{Ch2(Vpp)}{Ch1(Vpp)}\right]$$

Siendo Ch2 y Ch1 las sondas del osciloscopio. Obtenemos los siguientes resultados:

f = 5HZ: -6.02dB

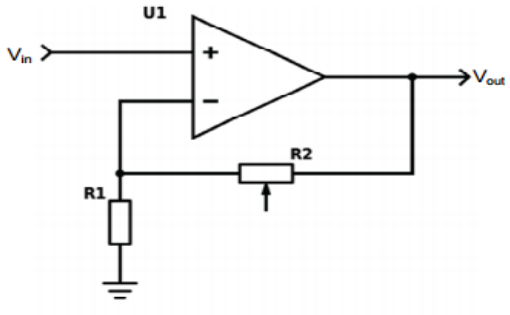
f = 10HZ: -3.928dB

f = 100HZ: -0.26dB

Comprobándose así que esta sección de nuestro circuito tiene un correcto funcionamiento.

3.2 Amplificador de ganancia variable

(a) Detalle el cálculo de los valores de los componentes. (b) Especifique los valores teóricos de ganancia máxima y mínima del amplificador. (c) Compare los valores teóricos de ganancia a la salida con los valores medidos.



a) Conociendo la función de transferencia del circuito:

$$A = \left(1 + \frac{R2}{R1}\right) \frac{1}{1 + \frac{(1 + \frac{R2}{R1})}{a}} \cong 1 + \frac{R2}{R1}$$

Nosotros de primera medida estimamos que para una salida máxima recomendada de 2VPP deberíamos tener una ganancia de 2 (A = 2). Es decir, que para una entrada estima de 1 VPP cubriríamos lo estipulado. Por lo que nuestra primera prueba fue la de poner valores a las resistencias iguales:

R1 = R2 (Potenciómetro) = 10KΩ

Puesto que más adelante tuvimos problemas con la señal de entrada (ya que la amplitud de la señal de entrada era cercana a 100mV), decidimos bajar el valor de la primera resistencia, con el fin de poder darle más potencia (volumen) a la señal. Así, las resistencias son:

R1 = 1KΩ

R2 = 10KΩ

Consiguiendo de esta manera una amplificación diez veces mayor a la original.

b)

La ganancia máxima se conseguiría cuando R2 (potenciómetro) tiene un valor de 10KΩ. Consiguiendo como consecuencia una ganancia de 20 dB.

La ganancia mínima se conseguiría cuando R2 (potenciómetro) tiene un valor lo más cercano a 0Ω. Es decir, conseguiríamos un cortocircuito donde la ganancia del sistema sería prácticamente uno.

c)

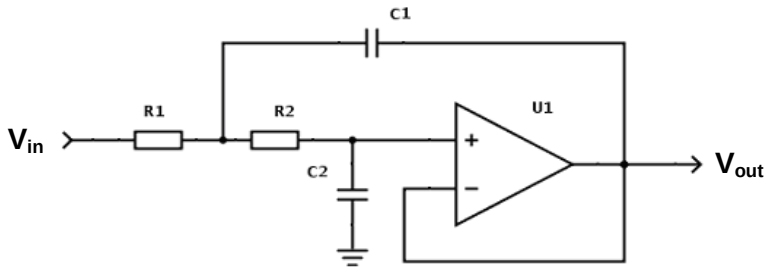
En la medida experimental con el osciloscopio hemos constatado que los cálculos que se describen en el apartado b se cumplen.

La ganancia mínima cuando R2 está en su valor máximo es de 20 dB (es decir, metiendo una señal de entrada de 160 mV conseguimos una salida de 1.68V)

La ganancia máxima cuando R2 está en su valor mínimo es de 0 dB (es decir, metiendo una señal de entrada de 160 mV conseguimos una salida de 164mV).

3.3 Filtro antisolapamiento

(a) Detalle el cálculo de los valores de los componentes. (b) Especifique los valores teóricos de ganancia del filtro a 1000 Hz, 3400 Hz y 10 kHz. (c) Compare el valor teórico de amplitud a la salida con los valores medidos.



a) Conociendo la función de transferencia del filtro:

$$H_{lp}(s) = \frac{w0^2}{s^2 + s\left(\frac{w0}{Q}\right) + w0^2} \cong \frac{\frac{1}{R^2C^2}}{s^2 + \frac{2}{RC}s + \frac{1}{R^2C^2}}$$

Y particularizando para filtros normalizados (K=1) con resistencias y condensadores iguales:

$$|H_{lp}(jwc)| = \left|\frac{w0^2}{-wc^2 + jwc\left(\frac{w0}{Q}\right) + w0^2}\right| = \frac{1}{\sqrt{2}} ; \frac{\sqrt{2}w0^2}{\sqrt{(w0^2 - wc^2)^2 + (2wcw0)^2}} = 1$$

Así, y sabiendo que:

$$Q = \frac{1}{2}$$

$$wc = 2\pi(3.4\times10^3)\,{}^{rad}/_{seg}$$

Entonces:

$$\sqrt{2}\,w0^2 = w0^2 + wc^2 ; w0 = \sqrt{\frac{wc^2}{\sqrt{2} - 1}} ; \quad \boldsymbol{w0 \cong 33193}\,{}^{rad}/_{seg}$$

Aplicando esta solución a la función de transferencia descrita arriba y dando un valor (escogido intencionadamente) a los dos condensadores, obtenemos los valores de las dos resistencias.

Siendo C = 10nF:

$$w0 = \frac{1}{RC} ; R = \frac{1}{Cw0} ; \boldsymbol{R \cong 3K\Omega}$$

Así, los valores (comerciales) obtenidos son:

R1 = R2 = 3KΩ

C1 = C2 = 10nF

b) Ganancia del filtro a las diferentes frecuencias.

Sabiendo que la ganancia se calcula como:

$$G = 10\log\left(|H(jw)|^2\right)$$

Obtenemos la ganancia en las siguientes frecuencias:

f = 1000HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{w0^4}{\left(w0^2-((2\pi(1000))^2\right)^2+(2(2\pi(1000))w0)^2}\right) = \mathbf{-0.3057dB}$$

f = 3400HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{w0^4}{\left(w0^2-((2\pi(3400))^2\right)^2+(2(2\pi(3400))w0)^2}\right) = \mathbf{-3.0103\ dB}$$

f = 10000HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{w0^4}{\left(w0^2-((2\pi(10000))^2\right)^2+(2(2\pi(10000))w0)^2}\right) = \mathbf{-13.2233dB}$$

c) Los valores obtenidos en el laboratorio para las frecuencias pedidas son los siguientes:

f = 1000HZ:

G = -0.2919 dB

f = 3400HZ:

G = -3.2416 dB

f = 10000HZ:

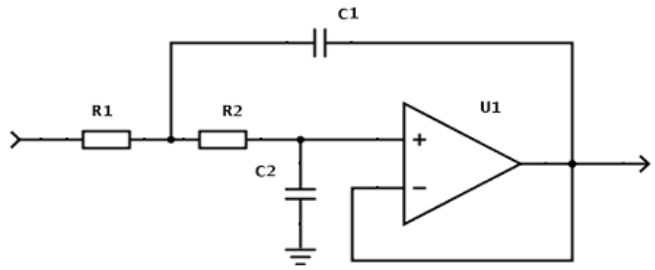
G = -13.0371 dB

Como se puede observar, los valores teóricos son muy próximos a los valores obtenidos en el laboratorio, dando a ver que nuestro circuito tiene un funcionamiento correcto. A su vez, también se puede comprobar que a una frecuencia de 3400Hz la caída de ganancia debería ser de 3dB, prácticamente como se demuestra teórica y experimentalmente.

(Constatado en el Hito Intermedio).

3.4 Filtro de reconstrucción

(a) Detalle el cálculo de los valores de los componentes. (b) Especifique los valores teóricos de ganancia del filtro a 1000 Hz, 3400 Hz y 10 kHz. (c) Compare el valor teórico de amplitud a la salida con los valores medidos.



a) Conociendo la función de transferencia del filtro:

$$H_{lp}(s) = \frac{w0^2}{s^2 + s\left(\frac{w0}{Q}\right) + w0^2}$$

Y particularizando para filtros normalizados (K=1) con condensadores iguales:

$$|H_{lp}(jwc)| = \left|\frac{w0^2}{-wc^2 + jwc\left(\frac{w0}{Q}\right) + w0^2}\right| = \frac{1}{\sqrt{2}}\ ;\ \frac{\sqrt{2}w0^2}{\sqrt{(w0^2 - wc^2)^2 + \left(\frac{wcw0}{Q}\right)^2}} = 1$$

Así, y sabiendo que:

$$Q = \frac{\sqrt{m}}{m+1} = 0.35$$

$$wc = 2\pi(3.4\times10^3)\ ^{rad}/_{seg}$$

$$w0 = \frac{1}{\sqrt{m}RC}$$

Entonces:

$$w0^4 + wc^2w0^2\left(2 - \frac{1}{Q^2}\right) - wc^4 = 0\ ;\ x^2 + wc^2x\left(2 - \frac{1}{Q^2}\right) - wc^4 = 0\ ; x = 2884926399$$

$$\boldsymbol{w0 \cong 53711.51\ ^{rad}/_{seg}}$$

Aplicando esta solución a las expresiones descritas arriba y dando un valor (escogido intencionadamente) a los dos condensadores, obtenemos los valores de las dos resistencias.

Siendo C = 1nF:

$$Q = \frac{\sqrt{m}}{m+1} = 0.35; \boldsymbol{m1 = 5.9965; m2 = 0.1667}$$

$$R = \frac{1}{\sqrt{m}w0C}\ ; \boldsymbol{R1 \cong 45.6K\Omega\ ;\ R2 \cong 7.6K\Omega}$$

Así, los valores (comerciales) obtenidos son:

R1 = 47KΩ

R2 = 7.5KΩ

C1 = C2 = 1nF

b) Ganancia del filtro a las diferentes frecuencias.

Sabiendo que la ganancia se calcula como:

$$G = 10\log\left(|H(jw)|^2\right)$$

Obtenemos la ganancia en las siguientes frecuencias:

f = 1000HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{w0^4}{\left(w0^2-((2\pi(1000))^2\right)^2+(2(2\pi(1000))w0)^2}\right) = \mathbf{-0.3524dB}$$

f = 3400HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{w0^4}{\left(w0^2-((2\pi(3400))^2\right)^2+(2(2\pi(3400))w0)^2}\right) = \mathbf{-3.0103\ dB}$$

f = 10000HZ:

$$G = 10\log\left(|H(jw)|^2\right) = 10\log\left(\frac{w0^4}{\left(w0^2-((2\pi(10000))^2\right)^2+(2(2\pi(10000))w0)^2}\right) = \mathbf{-10.5333dB}$$

c) Los valores obtenidos en el laboratorio para las frecuencias pedidas son los siguientes:

f = 1000HZ:

G = -0.3341 dB

f = 3400HZ:

G = -3.4387 dB

f = 10000HZ:

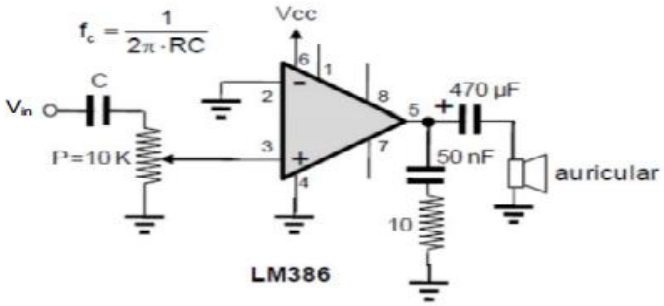
G = -10.7982 dB

Como se puede observar, los valores teóricos son muy próximos a los valores obtenidos en el laboratorio, dando a ver que nuestro circuito tiene un funcionamiento correcto. A su vez, también se puede comprobar que a una frecuencia de 3400Hz la caída de ganancia debería ser de 3dB, prácticamente como se demuestra teórica y experimentalmente.

(Constatado en el Hito Intermedio).

3.5 Etapa de potencia de audio de salida

(a) Detalle el cálculo del valor del condensador y de las resistencias a la entrada del módulo. (b) Calcule el valor teórico máximo de señal a la entrada del LM386 para que no se produzca ningún tipo de distorsión. (c) Si aumentamos la amplitud de la señal por encima de ese valor, ¿debido a qué efecto se produce la distorsión? (d) Compare los valores teóricos máximos de señal a la entrada del amplificador con los valores medidos, y comente los resultados.



a) Cálculo del valor teórico del condensador y de las resistencias de entrada.

Puesto que lo que nos interesa es que haya el menor ruido posible, pero que a su vez no se pierda nada de "información" en este proceso (correspondiente a la señal de audio), lo que nos conviene es que el filtro paso alto de la entrada tenga una frecuencia de corte la más cercana a 10Hz. Así:

Siendo R (Potenciómetro) = 10KΩ

$$fc = \frac{1}{2\pi RC} = 10Hz\,;\, \boldsymbol{C \cong 1.6 \times 10^{-6} F}$$

Así, los valores (comerciales) obtenidos son:

P = 10KΩ

C = 1.6µF

b)

Sabiendo que la alimentación del amplificador (LM386) es de ±5V, se deduce que la señal de salida no podrá soportar un voltaje mayor de la que se alimenta. Por tanto, la amplitud máxima que se podría obtener será de 0.25V (5/20V), siendo 20 la ganancia utilizada por defecto.

c)

Teniendo en cuenta lo dicho en el apartado anterior, es de esperar que para una entrada con una amplitud mayor de 0.25V se producirá una salida recortada y limitada a ±5V, de forma que nos queda una sinusoide recortada en amplitud y, por lo tanto, distorsionada.

d)

En los cálculos experimentales se puede apreciar que variando el potenciómetro (de máximo a mínimo) la señal de salida se ve distorsionada drásticamente.

Esto es, para una señal de entrada (Vin) de 0.06vp se produce una señal máxima (sin distorsión) de salida (Vout) 1.64 Vp.

4. DISEÑO DETALLADO DEL CIRCUITO DIGITAL

Para todos los módulos incluya el código VHDL DEBIDAMENTE COMENTADO.

NO SE OLVIDE DE INCLUIR EL DIAGRAMA DE BLOQUES COMPLETO EN EL APARTADO 2.

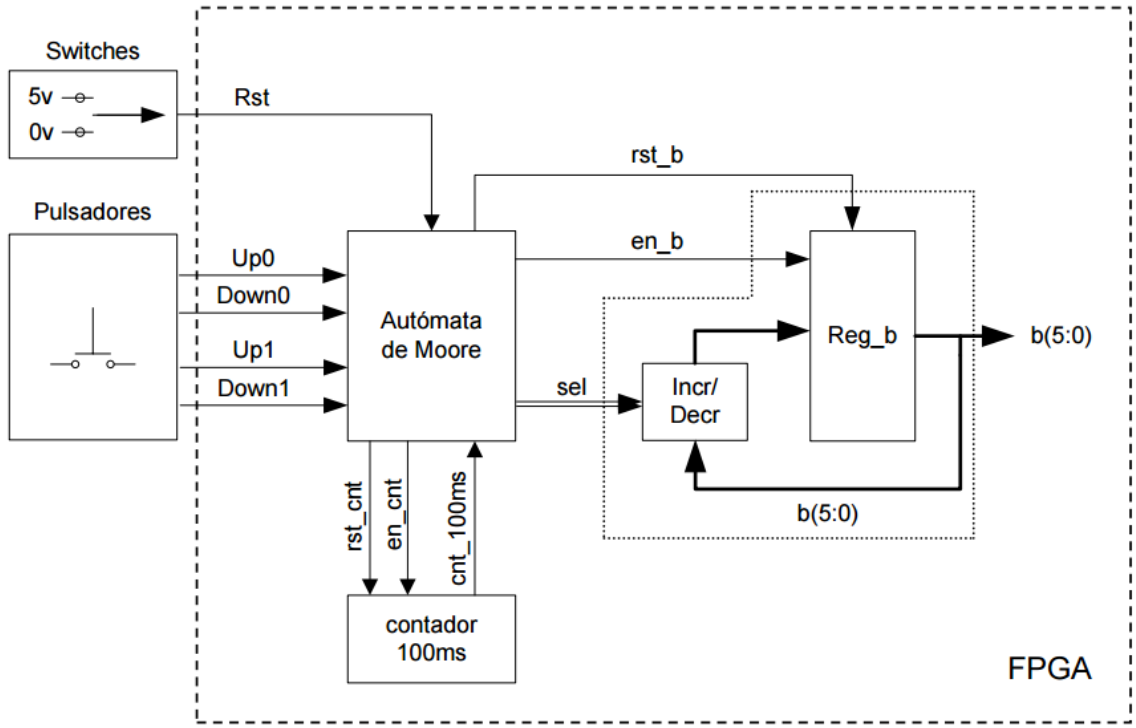
Ejemplo de código debidamente comentado:

```
----------------------------------------------------
-- AUTÓMATA DE CONTROL DE REFRESCO DE DISPLAY
--
-- Este autómata se encarga de activar cada uno de los
-- 4 displays secuencialmente, al mismo tiempo que
-- presenta un valor diferente en cada uno de ellos
-- proporcionado por un MUX. La frecuencia de refresco
-- viene dada por la señal de reloj CLK.
----------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control 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 control;

architecture a_control of control is
  signal SS  : STD_LOGIC_VECTOR (1 downto 0):=”00”; ; -- señal para control del MUX
begin                                                 -- dentro del proceso (valor inicial=00)

process (CLK)                        -- proceso que refresca los displays
begin
  if (CLK'event and CLK='1') then
    SS <= SS + 1;                    -- genera la secuencia 00,01,10 y 11
  end if;                            -- con cada flanco positivo del reloj CLK
end process;

S<=SS;                               -- la salida S se corresponde con la señal SS
AN<="0111" when SS="00" else         -- activa cada display en función del valor de SS
    "1011" when SS="01" else         -- (displays activos a nivel bajo)
    "1101" when SS="10" else
    "1110" when SS="11";

end a_control;
```

4.1 Control de los pulsadores

Inserte aquí el código VHDL del módulo de control de pulsadores (descripción estructural) y sus correspondientes submódulos (contador 100 ms, registro de generación del código de usuario, y autómata de Moore) debidamente comentados. Utilice un sub-apartado para cada componente.

En particular, explique los valores asignados a las constantes indicadas en el enunciado (AAAAA, BBBBB, …, GGGGG).




```
--------------------------------------------------------------------------------
-------
-- CONTROL DE PULSADORES
-- Fichero TOP del módulo, que describe la conexión de módulos MooreFSM,
-- Reg_b y contador_100ms; y se encarga del control de los pulsadores.
--------------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_pulsadores is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           Rst : in  STD_LOGIC;     -- entrada del reset
           Up0 : in  STD_LOGIC;     -- pulsador que incrementa 1
           Down0 : in  STD_LOGIC;   -- pulsador que decrementa 1
           Up1 : in  STD_LOGIC;     -- pulsador que incrementa 10
           Down1 : in  STD_LOGIC;   -- pulsador que decrementa 10
           b : out  STD_LOGIC_VECTOR (5 downto 0));  -- salida con el valor
del código introducido
end control_pulsadores;

architecture Behavioral of control_pulsadores is
```

```
component MooreFSM
      Port ( clk : in  STD_LOGIC;   -- entrada del reloj
           Up0 : in  STD_LOGIC;      -- pulsador que incrementa 1
           Down0 : in  STD_LOGIC;    -- pulsador que decrementa 1
           Up1 : in  STD_LOGIC;      -- pulsador que incrementa 10
           Down1 : in  STD_LOGIC;    -- pulsador que decrementa 10
           Rst : in  STD_LOGIC;      -- entrada del reset
           cnt_100ms : in  STD_LOGIC;      -- entrada proveniente del
contador_100ms
           rst_cnt : out  STD_LOGIC; -- salida que resetea el contador
           en_cnt : out  STD_LOGIC;  -- salida que habilita el contador
           rst_b : out  STD_LOGIC;   -- salida que resetea el registro
           en_b : out  STD_LOGIC;    -- salida que habilita el registro
           sel : out  STD_LOGIC_VECTOR (1 downto 0)); -- salida del selector
      end component;

component contador_100ms
      Port ( clk : in  STD_LOGIC;   -- entrada del reloj
           rst_cnt : in  STD_LOGIC;  -- entrada del reset del contador
           en_cnt : in  STD_LOGIC;   -- entrada que habilita el contador
           cnt_100ms : out  STD_LOGIC);    -- salida de contador_100ms
end component;

component Reg_b
      Port ( clk : in  STD_LOGIC;   -- entrada del reloj
           rst_b : in  STD_LOGIC;    -- entrada que resetea el registro
           en_b : in  STD_LOGIC;     -- entrada que habilita el registro
           sel : in  STD_LOGIC_VECTOR (1 downto 0);    -- entrada del
selector
           b : out  STD_LOGIC_VECTOR (5 downto 0));    -- salida con el valor
del código introducido
end component;

      -- señales auxiliares para realizar el port map
      signal  cnt_100ms, rst_cnt, en_cnt, rst_b, en_b: STD_LOGIC;
      signal sel : STD_LOGIC_VECTOR (1 downto 0);

begin

      U1 : MooreFSM port map (clk, Up0, Down0, Up1, Down1, Rst, cnt_100ms,
rst_cnt, en_cnt, rst_b, en_b, sel);
      U2 : contador_100ms port map (clk, rst_cnt, en_cnt, cnt_100ms);
      U3 : Reg_b port map (clk, rst_b, en_b, sel, b);

end Behavioral;
```

4.1.1 Moore_FSM

```
----------------------------------------------------------------------------
-------
-- AUTÓMATA DE MOORE
-- Máquina de estados de Moore que controla elfuncionamiento del módulo.
Recibe
-- las señales de los pulsadores y el interruptor, gestiona la cuenta de
100 ms,
-- y controla el valor del registro Reg_b.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MooreFSM is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           Up0 : in  STD_LOGIC;     -- pulsador que incrementa 1
           Down0 : in  STD_LOGIC;   -- pulsador que decrementa 1
           Up1 : in  STD_LOGIC;     -- pulsador que incrementa 10
           Down1 : in  STD_LOGIC;   -- pulsador que decrementa 10
           Rst : in  STD_LOGIC;     -- entrada del reset
           cnt_100ms : in  STD_LOGIC;     -- entrada proveniente del
contador_100ms
           rst_cnt : out  STD_LOGIC;      -- salida que resetea el contador
           en_cnt : out  STD_LOGIC; -- salida que habilita el contador
           rst_b : out  STD_LOGIC;  -- salida que resetea el registro
           en_b : out  STD_LOGIC;   -- salida que habilita el registro
           sel : out  STD_LOGIC_VECTOR (1 downto 0));-- salida del selector
end MooreFSM;

architecture Behavioral of MooreFSM is

type state_type is (Reset, Main, Up0a, Up0b, Up1a, Up1b, Down0a, Down0b,
Down1a, Down1b);  -- Estados del autómata
signal state_actual : state_type := Main;-- Estado actual inicial
signal state_siguiente : state_type := Main;    -- Estado siguiente inicial

begin -- Implementación de la arquitectura

process (clk) -- En este process se indica cuando se producen los cambios
de estado
begin
      if(clk'event and clk = '1') then          -- los cambios de estado se
producen en cada flanco de subida de reloj
            state_actual <= state_siguiente;
      end if;
end process;

process (state_actual, Rst, Up0, Down0, Up1, Down1, cnt_100ms) -- Cambios
de estado según el valor de las entradas
begin

            CASE state_actual IS

                  when Reset => -- Desde Reset se pasa directamente a Main
                        state_siguiente <= Main;
```

```
                when Main => -- En Main, si Rst es 1, cambia a estado
Reset, si no, según este pulsado un pulsador u otro vamos a su respectivo
estado
                        if (Rst='1') then
                              state_siguiente <= Reset;
                        elsif (Up0='1') then
                              state_siguiente <= Up0a;
                        elsif (Down0='1') then
                              state_siguiente <= Down0a;
                        elsif (Up1='1') then
                              state_siguiente <= Up1a;
                        elsif (Down1='1') then
                              state_siguiente <= Down1a;
                        else
                              state_siguiente <= Main;
                        end if;

                when Up0a => -- En Up0a, si no han pasado 100 ms,
seguimos en el estado; si pasan, cambia a Up0b
                        if (cnt_100ms='1') then
                              state_siguiente <= Up0b;
                        else
                              state_siguiente <= Up0a;
                        end if;

                when Up0b => -- En Up0b, se pasa directamente a Main
                        state_siguiente <= Main;

                when Up1a => -- En Up1a, si no han pasado 100 ms,
seguimos en el estado; si pasan, cambia a Up1b
                        if (cnt_100ms='1') then
                              state_siguiente <= Up1b;
                        else
                              state_siguiente <= Up1a;
                        end if;

                when Up1b => -- En Up1b, se pasa directamente a Main

                        state_siguiente <= Main;

                when Down0a => -- En Down0a, si no han pasado 100 ms,
seguimos en el estado; si pasan, cambia a Down0b
                        if (cnt_100ms='1') then
                              state_siguiente <= Down0b;
                        else
                              state_siguiente <= Down0a;
                        end if;

                when Down0b => -- En Down0b, se pasa directamente a Main
                        state_siguiente <= Main;

                when Down1a => -- En Down1a, si no han pasado 100 ms,
seguimos en el estado; si pasan, cambia a Down1b
                        if (cnt_100ms='1') then
                              state_siguiente <= Down1b;
                        else
                              state_siguiente <= Down1a;
                        end if;

                when Down1b => -- En Down1b, se pasa directamente a Main
                        state_siguiente <= Main;
```

```
                when others =>
                        state_siguiente <= Main;

        END CASE;
end process;

process (state_actual)  -- En este process se declara el valor de las
salidas en cada estado
begin
        rst_b <= '0'; en_b <= '0'; sel <= "00"; rst_cnt <= '1'; en_cnt
<= '0';     -- valores por defecto

        CASE state_actual IS

                when Reset =>
                        rst_b <= '1'; en_b <= '0'; sel <= "00"; rst_cnt <=
'1'; en_cnt <= '0';     --en Reset, valores por defecto, cambiando
únicamente rst_b a 1

                when Main =>
                        rst_b <= '0'; en_b <= '0'; sel <= "00"; rst_cnt <=
'1'; en_cnt <= '0';     -- en Main, valores por defecto

                when Up0a =>
                        rst_b <= '0'; en_b <= '0'; sel <= "00"; rst_cnt <=
'0'; en_cnt <= '1';     -- en Up0a, valores por defecto, cambiando rst_cnt
a 0 y en_cnt a 1

                when Up0b =>
                        rst_b <= '0'; en_b <= '1'; sel <= "00"; rst_cnt <=
'1'; en_cnt <= '0';     -- en Up0b, valores por defecto, cambiando
únicamente en_b a 1

                when Up1a =>
                        rst_b <= '0'; en_b <= '0'; sel <= "00"; rst_cnt <=
'0'; en_cnt <= '1';     -- en Up1a, valores por defecto, cambiando rst_cnt
a 0 y en_cnt a 1

                when Up1b =>
                        rst_b <= '0'; en_b <= '1'; sel <= "10"; rst_cnt <=
'1'; en_cnt <= '0';     -- en Up1b, valores por defecto, cambiando en_b a 1
y sel a 10

                when Down0a =>
                        rst_b <= '0'; en_b <= '0'; sel <= "00"; rst_cnt <=
'0'; en_cnt <= '1';     -- en Down0a, valores por defecto, cambiando
rst_cnt a 0 y en_cnt a 1

                when Down0b =>
                        rst_b <= '0'; en_b <= '1'; sel <= "01"; rst_cnt <=
'1'; en_cnt <= '0';     -- en Down0b, valores por defecto, cambiando en_b a
1 y sel a 01

                when Down1a =>
                        rst_b <= '0'; en_b <= '0'; sel <= "00"; rst_cnt <=
'0'; en_cnt <= '1';     -- en Down1a, valores por defecto, cambiando
rst_cnt a 0 y en_cnt a 1

                when Down1b =>
```

```
                        rst_b <= '0'; en_b <= '1'; sel <= "11"; rst_cnt <=
'1'; en_cnt <= '0';     -- en Down1b, valores por defecto, cambiando en_b a
1 y sel a 11

                  when others =>
                        rst_b <= '0'; en_b <= '0'; sel <= "00"; rst_cnt <=
'1'; en_cnt <= '0';

            END CASE;
end process;

end Behavioral;
```

4.1.2 Contador_100ms

```
----------------------------------------------------------------------------
-------
-- CONTADOR DE 100 MS
-- Contador que genera una señal que se activa con cada cuenta de 100 ms.
-- Cada vez que se realiza una pulsación, el autómata de Moore desactiva
-- la señal de reset del contador y habilita la cuenta. De esta forma,
-- cuando se activa la señal salida, el módulo indica que han
-- pasado 100 ms.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador_100ms is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           rst_cnt : in  STD_LOGIC; -- entrada del reset del contador
           en_cnt : in  STD_LOGIC;  -- entrada que habilita el contador
           cnt_100ms : out  STD_LOGIC);  -- salida de contador_100ms
end contador_100ms;

architecture Behavioral of contador_100ms is

signal contador : STD_LOGIC_VECTOR(22 downto 0) :=
"00000000000000000000000"; -- declaracion de la señal

begin

process (clk, rst_cnt)  -- En este process se indica la cuenta del contador
      begin
            if rst_cnt='1' then -- Cuando se activa la entrada del reset,
la salida se pone a 0
                  cnt_100ms <= '0';
                  contador <= (others => '0');
            elsif clk'event and clk='1' then
                  cnt_100ms <= '0';
                  if en_cnt='1' then
                        contador <= contador + '1';
                        if contador >= "10011000100101101000000" then --
valor máximo de la cuenta
                              cnt_100ms <= '1';
                        -- se pone la salida a 1
                              contador <= (others => '0');
                        end if;
                  end if;
            end if;
end process;

end Behavioral;
```

El valor de AAAAA que nos daba el enunciado era para indicar el número de ciclos de reloj, que hay en la placa FPGA, para llegar a 0.1 s. Al ser el procesador de la FPGA de 50MHz, el valor es de 5000000, que en binario hace AAAAA = "10011000100101101000000".

El valor de BBBBB es el número máximo de bits -1, para representar el valor de AAAAA en binario, el número de bits son 23, por tanto BBBBB = 22.

4.1.3 Reg_b

```
----------------------------------------------------------------------------
-------
-- REGISTRO
-- Este registro contiene en binario el valor del código introducido por el
-- usuario, b. En su funcionalidad se incorpora también que el registro sea
capaz
-- de variar su valor (en función de lo que indiquen las señales que
provienen de
-- la máquina de estados de Moore), y que no se excedan los límites máximos
-- y mínimos permitidos (0-63).
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Reg_b is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           rst_b : in  STD_LOGIC;   -- entrada que resetea el registro
           en_b : in  STD_LOGIC;    -- entrada que habilita el registro
           sel : in  STD_LOGIC_VECTOR (1 downto 0);  -- entrada del
selector
           b : out  STD_LOGIC_VECTOR (5 downto 0));  -- salida con el valor
del código introducido
end Reg_b;

architecture Behavioral of Reg_b is

signal s_Reg_b : STD_LOGIC_VECTOR(5 downto 0); -- declaracion de la señal
que sale del registro

begin

process (rst_b, clk)    -- En este process se modifica el valor de s_Reg_b
dependiendo del valor de la entrada sel
begin
      if clk'event and clk='1' then
            if (rst_b = '1') then
                  s_Reg_b <= "000000"; -- Asignación del valor por defecto
            end if;
            if en_b = '1' then
                  if ((sel = "00") and (s_Reg_b < "111111")) then
                        s_Reg_b <= s_Reg_b + "01"; -- Incremento +1 de la
señal
                  end if;
                  if ((sel = "01") and (s_Reg_b > "000000")) then
                        s_Reg_b <= s_Reg_b - "01"; -- Decremento -1 de la
señal
                  end if;
                  if ((sel = "10") and (s_Reg_b < "110110")) then
                        s_Reg_b <= s_Reg_b + "01010"; -- Incremento +10 de
la señal
                  end if;
                  if ((sel = "11") and (s_Reg_b > "001001")) then
                        s_Reg_b <= s_Reg_b - "01010"; -- Decremento -10 de
la señal
                  end if;
            end if;
```

```
      end if;
end process;

      b <= s_Reg_b; -- Asignación a la salida bdel valor obtenido de Reg_b
en el process

end Behavioral;
```

Los valores de CCCCC, DDDDD, EEEE, FFFFF y GGGGG indican:

El primero, el valor al que se inicializa b si pulsamos el reset, cuyo valor es 0 y en binario con 6 bits, CCCCC = ”000000”.

El valor de DDDDDD, indica el valor máximo que no puede superar b, si queremos incrementar en 1 su valor, siendo éste 63, por tanto en binario DDDDD = "111111".

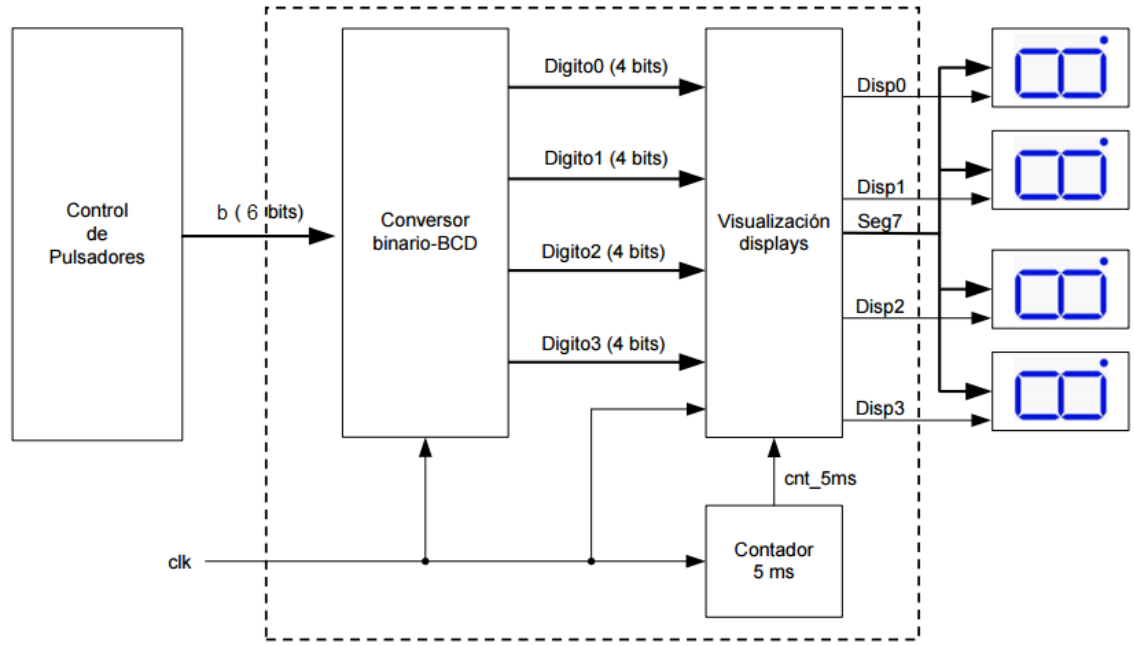
El valor de EEEEE , indica el valor mínimo que no puede superar b, si queremos decrementar en 1 su valor, siendo éste 0, por tanto en binario EEEEE = "000000".

El valor de FFFFF, indica el valor máximo que no puede superar b, si queremos incrementar en 100 su valor, siendo éste 54, por tanto en binario FFFFF = "110110".

Por último el valor GGGGG, indica el valor mínimo que no puede superar b, si queremos decrementar en 100 su valor, siendo éste 9, por tanto en binario GGGGG = "001001".

4.2 Control de los displays

Inserte aquí el código VHDL del módulo de control de los displays (descripción estructural) y sus correspondientes submódulos (conversor binario-BCD, contador 5 ms, y entidad de visualización) debidamente comentados. Utilice sub-apartados para cada uno de los componentes.




```
--------------------------------------------------------------------------------
-------
-- CONTROL DISPLAYS
-- Fichero TOP del módulo de control de los displays, que describe de forma
-- jerárquica la conexión de los módulos conversorBCD, contador_5ms y
-- visualizacion.
--------------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_displays is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           b : in  STD_LOGIC_VECTOR (5 downto 0);     -- entrada con el
valor del código introducido
           Disp0 : out  STD_LOGIC; -- salida encargada de activar el
display de las unidades
           Disp1 : out  STD_LOGIC; -- salida encargada de activar el
display de las decenas
           Disp2 : out  STD_LOGIC; -- salida encargada de activar el
display de las centenas
           Disp3 : out  STD_LOGIC; -- salida encargada de activar el
display de los millares
           Seg7 : out  STD_LOGIC_VECTOR (6 downto 0));      -- salida con el
valor del digito de BCD a 7 segmentos
end control_displays;

architecture Behavioral of control_displays is
```

```
component conversorBCD is
   Port ( --clk : in  STD_LOGIC;    -- entrada del reloj, no es necesaria,
se quita para evitar posibles errores
          b : in  STD_LOGIC_VECTOR (5 downto 0);     -- entrada con el
valor del código introducido
          D0 : out  STD_LOGIC_VECTOR (3 downto 0);  -- dígito que
corresponde a las unidades
          D1 : out  STD_LOGIC_VECTOR (3 downto 0);  -- dígito que
corresponde a las decenas
          D2 : out  STD_LOGIC_VECTOR (3 downto 0);  -- dígito que
corresponde a las centenas
          D3 : out  STD_LOGIC_VECTOR (3 downto 0)); -- dígito que
corresponde a los millares
end component;

component contador_5ms is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           cnt_5ms : out  STD_LOGIC);     -- salida de contador_5ms
end component;

component visualizacion is
     Port ( clk : in  STD_LOGIC;    -- entrada del reloj
           cnt_5ms : in  STD_LOGIC;-- entrada proveniente del
contador_5ms, para refrescar los displays
           Digito0 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a las unidades del valor
           Digito1 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a las decenas del valor
           Digito2 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a las centenas del valor
           Digito3 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a los millares del valor
           Disp0 : out  STD_LOGIC; -- salida encargada de activar el
display de las unidades
           Disp1 : out  STD_LOGIC; -- salida encargada de activar el
display de las decenas
           Disp2 : out  STD_LOGIC; -- salida encargada de activar el
display de las centenas
           Disp3 : out  STD_LOGIC; -- salida encargada de activar el
display de los millares
           Seg7 : out  STD_LOGIC_VECTOR (6 downto 0));      -- salida con el
valor del digito de BCD a 7 segmentos
end component;

      -- señales auxiliares para realizar el port map
      signal cnt_5ms : STD_LOGIC;
      signal Digito0, Digito1, Digito2, Digito3 : STD_LOGIC_VECTOR (3
downto 0);

begin

      U1 : conversorBCD port map (b, Digito0, Digito1, Digito2, Digito3);
      U2 : contador_5ms port map (clk, cnt_5ms);
      U3 : visualizacion port map (clk, cnt_5ms, Digito0, Digito1, Digito2,
Digito3, Disp0, Disp1, Disp2, Disp3, Seg7);

end Behavioral;
```

4.2.1 ConversorBCD

```
----------------------------------------------------------------------------
-------
-- CONVERSOR BCD
-- Es el módulo que se encarga de convertir la señal b (representada
-- en binario y con valores entre 0 y 63) al formato BCD.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity conversorBCD is
    Port ( --clk : in  STD_LOGIC;   -- entrada del reloj, no es necesaria,
se quita para evitar posibles errores
           b : in  STD_LOGIC_VECTOR (5 downto 0);     -- entrada con el
valor del código introducido
           D0 : out  STD_LOGIC_VECTOR (3 downto 0);  -- dígito que
corresponde a las unidades
           D1 : out  STD_LOGIC_VECTOR (3 downto 0);  -- dígito que
corresponde a las decenas
           D2 : out  STD_LOGIC_VECTOR (3 downto 0);  -- dígito que
corresponde a las centenas
           D3 : out  STD_LOGIC_VECTOR (3 downto 0)); -- dígito que
corresponde a los millares
end conversorBCD;

architecture Behavioral of conversorBCD is

begin

process (b) -- En este process se obtiene el valor de las unidades y las
decenas del valor de entrada b

      VARIABLE resto : STD_LOGIC_VECTOR (5 downto 0);-- variable auxiliar
para realizar el algoritmo

begin
      D1 <= (others => '0'); -- VALORES POR DEFECTO
      resto := b;

      if b >= conv_std_logic_vector(10, 6) then -- conv_std_logic_vector(x,
y) convierte el valor x a binario con y bits
            D1 <= "0001";
            resto := b - conv_std_logic_vector(10, 6); -- resto = b - 10
      end if;

      if b >= conv_std_logic_vector(20, 6) then
            D1 <= "0010";
            resto := b - conv_std_logic_vector(20, 6); -- resto = b - 20
      end if;

      if b >= conv_std_logic_vector(30, 6) then
            D1 <= "0011";
            resto := b - conv_std_logic_vector(30, 6); -- resto = b - 30
      end if;

      if b >= conv_std_logic_vector(40, 6) then
            D1 <= "0100";
```

```
            resto := b - conv_std_logic_vector(40, 6); -- resto = b - 40
      end if;

      if b >= conv_std_logic_vector(50, 6) then
            D1 <= "0101";
            resto := b - conv_std_logic_vector(50, 6); -- resto = b - 50
      end if;

      if b >= conv_std_logic_vector(60, 6) then
            D1 <= "0110";
            resto := b - conv_std_logic_vector(60, 6); -- resto = b - 60
      end if;

      D0 <= resto(3 downto 0); -- Asignacion de salida a las unidades
                                                        --D1 corresponde
a las decenas y se asigna dentro del process

end process;

      D2 <= "0000";     -- D2 y D3 corresponderían a centenas y millares
      D3 <= "0000";     -- pero por la limitación de 0 a 63, los mantenemos
a 0

end Behavioral;
```

4.2.2 Contador_5ms

```
----------------------------------------------------------------------------
-------
-- CONTADOR 5 MS
-- Para multiplexar los datos de los 4 dígitos para utilizar
-- los 4 displays a la vez, hace falta una señal para que la
-- visualización sea correcta. Se ha seleccionado T = 5ms,
-- de forma que cada display se refresque cada 20 ms.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador_5ms is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           cnt_5ms : out  STD_LOGIC);     -- salida de contador_5ms
end contador_5ms;

architecture Behavioral of contador_5ms is

signal contador : STD_LOGIC_VECTOR(17 downto 0) := "000000000000000000"; --
declaracion de la señal

begin

process (clk)     -- En este process se indica la cuenta del contador
      begin
            if clk'event and clk='1' then
                  cnt_5ms <= '0';
                  contador <= contador + '1';
                  if contador >= "111101000010010000" then -- valor máximo
de la cuenta
                        cnt_5ms <= '1';
                  -- se pone la salida a 1
                        contador <= (others => '0');
                  end if;
            end if;
end process;

end Behavioral;
```

4.2.3 Visualizacion

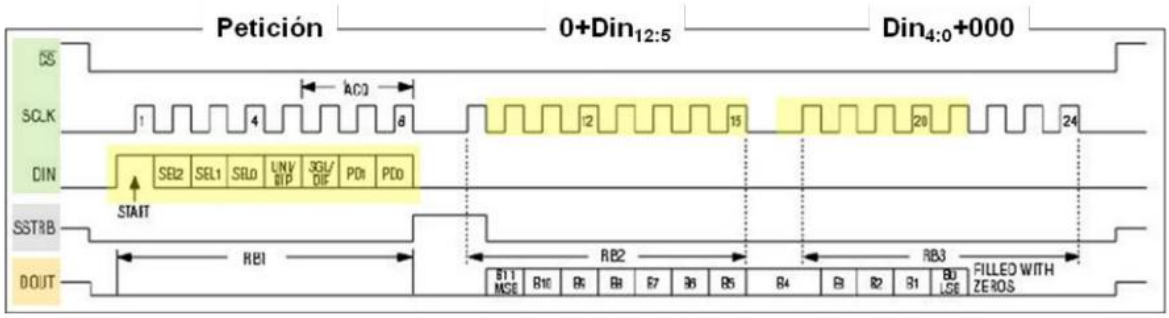
```
----------------------------------------------------------------------------
-------
-- VISUALIZACIÓN
-- Es el módulo encargado de mostrar los datos en los displays. Puesto que
la
-- señal de datos (Seg7) es compartida, es necesario activar cada 5 ms,
cada uno
-- de los displays y poner los datos correspondientes en el bus Seg7.
-- También es necesario convertir los datos de BCD a 7 segmentos.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity visualizacion is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           cnt_5ms : in  STD_LOGIC;-- entrada proveniente del
contador_5ms, para refrescar los displays
           Digito0 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a las unidades del valor
           Digito1 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a las decenas del valor
           Digito2 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a las centenas del valor
           Digito3 : in  STD_LOGIC_VECTOR (3 downto 0);     -- dígito en BCD
que corresponde a los millares del valor
           Disp0 : out  STD_LOGIC; -- salida encargada de activar el
display de las unidades
           Disp1 : out  STD_LOGIC; -- salida encargada de activar el
display de las decenas
           Disp2 : out  STD_LOGIC; -- salida encargada de activar el
display de las centenas
           Disp3 : out  STD_LOGIC; -- salida encargada de activar el
display de los millares
           Seg7 : out  STD_LOGIC_VECTOR (6 downto 0));     -- salida con el
valor del digito de BCD a 7 segmentos
end visualizacion;

architecture Behavioral of visualizacion is

signal sel : STD_LOGIC_VECTOR(1 downto 0) := "00"; -- señal que se encarga
de activar un display u otro
signal digitoBCD : STD_LOGIC_VECTOR(3 downto 0); -- señal que usamos para
pasar de BCD a 7 segmentos

begin

process (clk) -- en este process se cambia el valor de sel cada 5 ms
begin
 if clk'event and clk='1' then      -- en cada flanco de subida del reloj
      if cnt_5ms = '1' then               -- cuando la entrada cnt_5ms está
activada
            sel <= sel + "01";
      end if;
 end if;
end process;
```

```
process (sel, Digito0, Digito1, Digito2, Digito3) -- process en el que se
activa un display u otro dependiendo del valor de sel
      begin
            digitoBCD <= digito0; -- asignación por defecto
            Disp0 <= '1'; -- por defecto (activas a nivel bajo)
            Disp1 <= '1';
            Disp2 <= '1';
            Disp3 <= '1';

                  CASE sel IS

                        when "00" =>
                              Disp0 <= '0';
                              digitoBCD <= Digito0;

                        when "01" =>
                              Disp1 <= '0';
                              digitoBCD <= Digito1;

                        when "10" =>
                              Disp2 <= '1';
                              digitoBCD <= Digito2;

                        when "11" =>
                              Disp3 <= '1';
                              digitoBCD <= Digito3;

                        when others => Disp0 <= '1';

      END CASE;
end process;

with digitoBCD select Seg7 <= -- with select para indicar el valor de Seg7
dependiendo del valor de digitoBCD (activas a nivel bajo)
 "1000000" when "0000",
 "1111001" when "0001",
 "0100100" when "0010",
 "0110000" when "0011",
 "0011001" when "0100",
 "0010010" when "0101",
 "0000010" when "0110",
 "1111000" when "0111",
 "0000000" when "1000",
 "0011000" when "1001",
 "1111111" when others;

end Behavioral;
```

4.3 Generación del bloque de control del ADC

Inserte aquí el código VHDL de la entidad del control del ADC, debidamente comentado, justificando su funcionamiento.

En particular, explique el funcionamiento de los contadores 1, 2 y 3, y cómo se generan las señales CS0, SCLK y DIN.




```
--------------------------------------------------------------------------------
-------
-- CONTROL ADC
-- Es el encargado de enviar y recibir todas las señales que deben llegar
-- al ADC. Este bloque debe realizar 2 cosas:
-- En primer lugar, enviar el byte de petición por medio de las señales
-- CS0, SCLK y DIN.
-- En segundo, recibir la respuesta a través de la señal DOUT y entregar el
-- valor digital que se reciba a la siguiente etapa.
--------------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_ADC is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
           Rst : in  STD_LOGIC;     -- señal de reset
           start_ADC : in  STD_LOGIC;     -- señal que habilita el módulo
           DOUT : in  STD_LOGIC;    -- señal serie que se recibe desde el
ADC
           end_ADC : out  STD_LOGIC;      -- señal que da por terminado la
operación del módulo
           CS0 : out  STD_LOGIC;    --  señal de control para el ADC
           SCLK : out  STD_LOGIC;   -- señal de control para el ADC
           DIN : out  STD_LOGIC;    -- señal de petición par el ADC
           valor_ADC : out  STD_LOGIC_VECTOR (11 downto 0));      -- último
valor binario completo recibido desde el ADC
end control_ADC;

architecture Behavioral of control_ADC is

 -- Declaración de las señales necesarias
signal en_cnt : STD_LOGIC := '0';  -- señal que habilita el contador de 1
us
signal fin_de_cuenta : STD_LOGIC := '0'; -- señal que indica que ha pasado
1 us
signal contador : STD_LOGIC_VECTOR  (5 downto 0) := "000000";     -- señal
que lleva la cuenta para 1 us
```

```
signal byte_peticion_ADC : STD_LOGIC_VECTOR (7 downto 0) := "00000000";
      -- señal que porta el byte de peticion
signal bit_DIN : STD_LOGIC_VECTOR (3 downto 0) := "0000";  -- señal que
cuenta los bits transmitidos
signal st : STD_LOGIC_VECTOR (1 downto 0) := "00";   -- señal que indica el
estado del autómata
signal dato_ADC : STD_LOGIC_VECTOR (15 downto 0) := "0000000000000000";
      -- señal que porta la recepcion de bits
signal aux : STD_LOGIC := '0';      -- señal que equivale a SCLK
signal salida : STD_LOGIC_VECTOR (11 downto 0) := "000000000000";      --
señal que equivale a valor_ADC

begin

 -- Procesos (contadores 1 y 2) y otras asignaciones de señales de la
entidad

process(clk)      -- en este process se produce la cuenta de 1 us
      begin
            if clk'event and clk='1' then
                  if en_cnt = '1' then
                        fin_de_cuenta <= '0';
                        contador <= contador + '1';
                        if contador >= "110010" then -- valor máximo de la
cuenta
                              fin_de_cuenta <= '1';
                              contador <= (others => '0');
                        end if;
                  end if;
            end if;
end process;

process (clk)     -- este process equivale a un biestable del que resulta
SCLK
      begin
            if clk'event and clk='1' then
                  if fin_de_cuenta = '1' then
                        aux <= not(aux);
                  end if;
            end if;
end process;

process (Rst, clk)      -- en este process se entrega el byte de peticion,
se reciben los datos serie y se envia el dato de salida
      begin
            if (Rst = '1') then -- Poner a cero todas las señales del
proceso
                  st <= "00";
                  bit_DIN <= "0000";
                  byte_peticion_ADC <= "00000000";
                  dato_ADC <= "0000000000000000";
                  end_ADC <= '0';

            elsif clk'event and clk = '1' then

                  end_ADC <= '0';   -- necesario para que la señal dure
sólo un pulso de reloj

                  CASE st IS

                        when "00" => -- Reposo (Estado 00)
```

```
                              en_cnt <= '0'; -- Inicializar contadores
                              byte_peticion_ADC <= "10010111"; -- Precarga
del byte de peticion
                              if start_ADC = '1' then -- Selección del
canal 0 en modo bipolar (-1.3, 2.5V)
                                    st <= "01";
                              end if;

                        when "01" => -- Entrega de la peticion (Estado 01)
                              en_cnt <= '1';
                                    if (fin_de_cuenta = '1' and aux = '1')
then  -- Flanco de bajada de SCLK
                                          byte_peticion_ADC <=
byte_peticion_ADC(6 downto 0) & '0'; -- Desplaza petición
                                          bit_DIN <= bit_DIN + "01";
                                          if bit_DIN >= "0111" then -- Se
ha enviado la peticion completa
                                                st <= "10";
                                                bit_DIN <= (others => '0');
                                          end if;
                                    end if;

                        when "10" => -- Recepcion de los datos serie
(Estado 10)
                              en_cnt <= '1';
                              if (fin_de_cuenta = '1' and aux = '0') then -
- Flanco de subida de SCLK
                                    dato_ADC <= dato_ADC(14 downto 0) &
DOUT; -- Recepcion serie
                              end if;
                              if (fin_de_cuenta = '1' and aux = '1') then -
- Flanco de bajada de SCLK
                                    bit_DIN <= bit_DIN + "01";
                                          if bit_DIN = "1111" then -- Se ha
recibido el dato completo
                                                st <= "11";
                                                bit_DIN <= (others => '0');
                                          end if;
                              end if;

                        when "11" => -- Entrega del dato de salida, y final
(Estado 11)
                              st <= "00";
                              salida <= dato_ADC(14 downto 3);    -- se
pasan a valor_ADC los bits significativos del conversor
                              end_ADC <= '1';

                        when others =>
                              st <= "00";

                  END CASE;
      end if;
end process;

-- asignación de salidas
SCLK <= aux;
DIN <= byte_peticion_ADC(7) when st = "01" else '0'; -- se va pasando por
DIN el byte de peticion
CS0 <= '0' when (st = "01" or st = "10") else '1';
valor_ADC <= salida;
```

```
end Behavioral;
```

Los tres contadores funcionan sólo en los estados 01 y 10. Todos los contadores están a 0 durante el estado de reposo, y se activan durante los estados de envío y recepción de datos. El primer contador debe generar una señal (fin_de_cuenta) cada 1 µs para que el segundo contador cambie cada 1 µs.

El segundo contador recibe la señal de fin de cuenta, y cambia su estado generando la señal SCLK. Este segundo contador podría identificarse como un biestable.

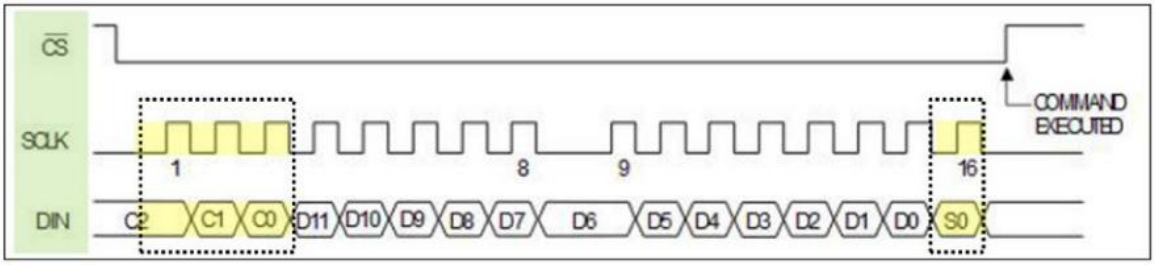
El tercer contador recibe la señal fin_de_cuenta del primer contador y la señal SCLK del segundo, e incrementa el número de los bits de petición enviados a través de la señal DIN (hasta 8), actualizando el valor de dicha señal en cada flanco de bajada de SCLK.

La señal CS0 es una señal de control, de manera que siempre que esté a 1, DIN va a estar a 0.

4.4 Generación del bloque de control del DAC

Inserte aquí el código VHDL de la entidad del control del DAC, debidamente comentado, justificando su funcionamiento.

En particular, explique el funcionamiento de los contadores que haya utilizado, y la generación de las señales CS1, SCLK y DIN.




```
--------------------------------------------------------------------------------
-------
-- CONTROL DAC
-- Es el encargado de generar todas las señales digitales que necesita
-- el DAC para realizar la conversión
--------------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_DAC is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
           Rst : in  STD_LOGIC;     -- señal de reset
           start_DAC : in  STD_LOGIC;     -- señal que habilita el módulo
           valor_DAC : in  STD_LOGIC_VECTOR (11 downto 0); -- valor binario
a enviar al DAC
           end_DAC : out  STD_LOGIC;           -- señal que da por
terminado la operación del módulo
           CS1 : out  STD_LOGIC;    --  señal de control para el DAC
           SCLK : out  STD_LOGIC;   -- señal de control para el DAC
           DIN : out  STD_LOGIC);    -- dato que se envía al DAC
end control_DAC;

architecture Behavioral of control_DAC is

 -- Declaración de las señales necesarias
signal en_cnt : STD_LOGIC := '0';  -- señal que habilita el contador de 1
us
signal fin_de_cuenta : STD_LOGIC := '0'; -- señal que indica que ha pasado
1 us
signal contador : STD_LOGIC_VECTOR  (5 downto 0) := "000000";    -- señal
que lleva la cuenta para 1 us
signal byte_transmision_DAC : STD_LOGIC_VECTOR (15 downto 0) :=
"0000000000000000";     -- señal que porta el byte de transmisión
signal bit_DIN : STD_LOGIC_VECTOR (3 downto 0) := "0000";  -- señal que
cuenta los bits transmitidos
signal st : STD_LOGIC_VECTOR (1 downto 0) := "00";   -- señal que indica el
estado del autómata
signal aux : STD_LOGIC := '0';      -- señal que equivale a SCLK
```

```
begin

 -- Procesos (contadores 1 y 2) y otras asignaciones de señales de la
entidad

process(clk)      -- en este process se produce la cuenta de 1 us
      begin
            if clk'event and clk='1' then
                  if en_cnt = '1' then
                        fin_de_cuenta <= '0';
                        contador <= contador + '1';
                        if contador >= "110010" then -- valor máximo de la
cuenta
                              fin_de_cuenta <= '1';
                              contador <= (others => '0');
                        end if;
                  end if;
            end if;
end process;

process (clk)     -- este process equivale a un biestable del que resulta
SCLK
      begin
            if clk'event and clk='1' then
                  if fin_de_cuenta = '1' then
                        aux <= not(aux);
                  end if;
            end if;
end process;

process (Rst, clk)      -- en este process se entrega los bytes de
transmisión y se envia el dato de salida
      begin
            if (Rst = '1') then -- Poner a cero todas las señales del
proceso
                  st <= "00";
                  bit_DIN <= "0000";
                  byte_transmision_DAC <= "0000000000000000";
                  end_DAC <= '0';

            elsif clk'event and clk = '1' then

                  end_DAC <= '0';   -- necesario para que la señal dure
sólo un pulso de reloj

                  CASE st IS

                        when "00" => -- Reposo (Estado 00)

                              en_cnt <= '0'; -- Inicializar contadores
                              byte_transmision_DAC <= "000" & valor_DAC &
'0';
                              if start_DAC = '1' then
                                    st <= "01";
                              end if;

                        when "01" => -- Entrega de la peticion (Estado 01)
                              en_cnt <= '1';
                              if (fin_de_cuenta = '1' and aux = '1') then -
- Flanco de bajada de SCLK
```

```
                                    byte_transmision_DAC <=
byte_transmision_DAC(14 downto 0) & '0'; -- desplaza transmision
                                    bit_DIN <= bit_DIN + "01";
                                    if bit_DIN >= "1111" then -- Se ha
enviado la peticion completa
                                          st <= "11";
                                          bit_DIN <= (others => '0');
                                    end if;
                              end if;

                        when "11" => -- Entrega del dato de salida, y final
(Estado 11)
                              st <= "00";
                              end_DAC <= '1';

                        when others =>
                              st <= "00";

                  END CASE;
      end if;
end process;

-- asignación de salidas
SCLK <= aux;
CS1 <= '0' when (st = "01") else '1';
DIN <= byte_transmision_DAC(15) when st = "01" else '0';    -- se va pasando
por DIN el byte de transmisión

end Behavioral;
```

Es un caso similar al del bloque de control del ADC, pero algo más sencillo:

Los tres contadores funcionan sólo en el estado 01. Todos los contadores están a 0 durante el estado de reposo y se activan durante el estado 01 para controlar la temporización de envío de los datos. El primer contador debe generar una señal (fin_de_cuenta) cada 1 µs para que el segundo contador cambie cada 1 µs.

El segundo contador recibe la señal de fin de cuenta, y cambia su estado generando la señal SCLK. Este segundo contador podría identificarse como un biestable.

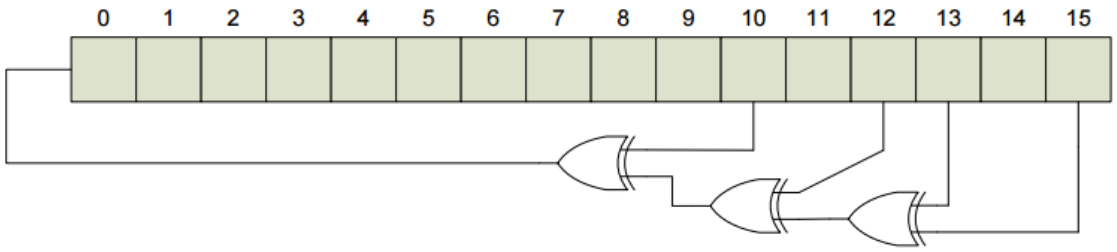
El tercer contador recibe la señal fin_de_cuenta del primer contador y la señal SCLK del segundo, e incrementa el número de los bits de petición enviados a través de la señal DIN (hasta 8), actualizando el valor de dicha señal en cada flanco de bajada de SCLK.

La señal CS1 es una señal de control, de manera que siempre que esté a 1, DIN va a estar a 0.

4.5 Generación del bloque de encriptación

Inserte aquí el código VHDL de la entidad de encriptación, debidamente comentado, justificando su funcionamiento.

En particular, explique el funcionamiento del contador pseudo-aleatorio, y cómo afecta a la salida del sistema.




```
----------------------------------------------------------------------------
-------
-- ENCRIPTACIÓN
-- Es el encargado de comparar el código predefinido de cifrado
-- (fijado en los interruptores) con el código introducido por
-- el usuario (generado mediante los pulsadores).
-- En caso de que sean iguales, la señal de entrada se transmitirá
-- a la salida.
-- En caso de que no lo sean, se entrega al DAC la salida de un
-- generador pseudo-aleatorio de 16 bits.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity pseudo_random is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
           Rst : in  STD_LOGIC;     -- señal de reset
           start_random : in  STD_LOGIC; -- señal que habilita el módulo
           sw : in  STD_LOGIC_VECTOR (5 downto 0);    -- código predefinido
de cifrado, especificado por los interruptores
           b : in  STD_LOGIC_VECTOR (5 downto 0);     -- código introducido
por el usuario a través de los pulsadores
           valor_ADC : in  STD_LOGIC_VECTOR (11 downto 0); -- último valor
binario completo recibido desde el ADC
           end_random : out  STD_LOGIC;         -- señal que da por
terminado la operación del módulo
           valor_DAC : out  STD_LOGIC_VECTOR (11 downto 0));     -- valor
binario a enviar al DAC
end pseudo_random;

architecture Behavioral of pseudo_random is

-- Declaración de las señales necesarias
signal registro : STD_LOGIC_VECTOR (15 downto 0) := "0000000010000001";
      -- representa el registro de desplazamiento
signal aux : STD_LOGIC_VECTOR (11 downto 0) := "000000000000";   -- para
sumarle el offset a la entrada de datos
signal semilla : STD_LOGIC;   -- valor inicial para el registro
```

```
begin

aux <= valor_ADC + 2048;      -- suma del offset necesario
semilla <= (((registro(15) XOR registro(13)) XOR registro(12)) XOR
registro(10));    -- registro de desplazamiento

process (Rst, clk)      -- en este process se comparan los códigos y se
asignan las salidas
      begin
            if Rst = '1' then
                  valor_DAC <= "000000000000";
            elsif clk'event and clk='1' then
                  if start_random = '1' then
                        if (b = sw) then -- si coinciden los valores, la
salida es igual a la entrada
                              valor_DAC <= aux;
                              end_random <= '1';
                        else  -- si no coinciden, se genera una secuencia
pseudoaleatoria
                              valor_DAC <= registro (11 downto 0);
                              end_random <= '1';
                        end if;
                  end if;
            end if;
end process;

process (clk)     -- process que sirve para que los valores del registro
estén en continuo cambio
      begin
      if clk'event and clk='1' then
            registro <= registro(14 downto 0) & semilla;
      end if;
end process;

end Behavioral;
```

El bloque de encriptación de la señal es el encargado de comparar el código predefinido de cifrado (fijado en los interruptores) con el código introducido por el usuario (generado mediante los pulsadores). En caso de que sean iguales, la señal de entrada se transmitirá a la salida. En caso de que no lo sean, se entrega al DAC la salida de un generador pseudo-aleatorio de 16 bits.

La arquitectura para el generador pseudo-aleatorio es un registro de desplazamiento con realimentación lineal (LFSR) de 16 bits, de los cuales se entregan al DAC los 12 bits menos significativos. Este tipo de generadores tienen la característica de que pasan por casi todos los posibles valores del registro antes de que se repita el estado (con la excepción del caso en que todos los bits del registro sean 0, que mantiene su valor), y son muy fácilmente implementables.

En caso de que ambos códigos sean iguales, no habrá ninguna consecuencia para la salida.

En caso contrario, la señal de salida se verá afectada por mucho ruido que la distorsionará.

4.6 Entidad jerárquica TOP

Inserte aquí el código VHDL de la entidad TOP (descripción estructural) debidamente comentado. Explique también la operación del sistema en función de los distintos estados.

```
----------------------------------------------------------------------------
-------
-- ENTIDAD JERÁRQUICA TOP
-- Versión final del fichero jerarquía del subsistema digital.
-- Contiene los bloques de control de pulsadores, control de displays,
-- control del ADC, control del DAC, control global y encriptación.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top_digital is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           Rst : in  STD_LOGIC;     -- entrada del reset
           Up0 : in  STD_LOGIC;     -- pulsador que incrementa 1
           Down0 : in  STD_LOGIC;   -- pulsador que decrementa 1
           Up1 : in  STD_LOGIC;     -- pulsador que incrementa 10
           Down1 : in  STD_LOGIC;   -- pulsador que decrementa 1
           DOUT : in  STD_LOGIC;    -- señal serie que se recibe desde el
ADC
           sw : in  STD_LOGIC_VECTOR (5 downto 0);    -- código predefinido
de cifrado, especificado por los interruptores              DOUT : in
STD_LOGIC;
           Disp0 : out  STD_LOGIC; -- salida encargada de activar el
display de las unidades
           Disp1 : out  STD_LOGIC; -- salida encargada de activar el
display de las decenas
           Disp2 : out  STD_LOGIC; -- salida encargada de activar el
display de las centenas
           Disp3 : out  STD_LOGIC; -- salida encargada de activar el
display de los millares
           Seg7 : out  STD_LOGIC_VECTOR (6 downto 0);-- salida con el valor
del digito de BCD a 7 segmentos
           CS0 : out  STD_LOGIC;    --  señal de control para el ADC
           CS1 : out  STD_LOGIC;    --  señal de control para el DAC
           SCLK : out  STD_LOGIC;   -- señal de control para el DAC
           DIN : out  STD_LOGIC);    -- dato que se envía al DAC
end top_digital;

architecture Behavioral of top_digital is

component control_pulsadores is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
           Rst : in  STD_LOGIC;     -- entrada del reset
           Up0 : in  STD_LOGIC;     -- pulsador que incrementa 1
           Down0 : in  STD_LOGIC;   -- pulsador que decrementa 1
           Up1 : in  STD_LOGIC;     -- pulsador que incrementa 10
           Down1 : in  STD_LOGIC;   -- pulsador que decrementa 10
           b : out  STD_LOGIC_VECTOR (5 downto 0));  -- salida con el valor
del código introducido
end component;

component control_displays is
    Port ( clk : in  STD_LOGIC;     -- entrada del reloj
```

```
            b : in  STD_LOGIC_VECTOR (5 downto 0);     -- entrada con el
valor del código introducido
            Disp0 : out  STD_LOGIC; -- salida encargada de activar el
display de las unidades
            Disp1 : out  STD_LOGIC; -- salida encargada de activar el
display de las decenas
            Disp2 : out  STD_LOGIC; -- salida encargada de activar el
display de las centenas
            Disp3 : out  STD_LOGIC; -- salida encargada de activar el
display de los millares
            Seg7 : out  STD_LOGIC_VECTOR (6 downto 0));     -- salida con el
valor del digito de BCD a 7 segmentos
end component;

component control_ADC is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
            Rst : in  STD_LOGIC;     -- señal de reset
            start_ADC : in  STD_LOGIC;     -- señal que habilita el módulo
            DOUT : in  STD_LOGIC;    -- señal serie que se recibe desde el
ADC
            end_ADC : out  STD_LOGIC;      -- señal que da por terminado la
operación del módulo
            CS0 : out  STD_LOGIC;    --  señal de control para el ADC
            SCLK : out  STD_LOGIC;   -- señal de control para el ADC
            DIN : out  STD_LOGIC;    -- señal de petición par el ADC
            valor_ADC : out  STD_LOGIC_VECTOR (11 downto 0));     -- último
valor binario completo recibido desde el ADC
end component;

component control_DAC is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
            Rst : in  STD_LOGIC;     -- señal de reset
            start_DAC : in  STD_LOGIC;     -- señal que habilita el módulo
            valor_DAC : in  STD_LOGIC_VECTOR (11 downto 0); -- valor binario
a enviar al DAC
            end_DAC : out  STD_LOGIC;           -- señal que da por
terminado la operación del módulo
            CS1 : out  STD_LOGIC;    --  señal de control para el DAC
            SCLK : out  STD_LOGIC;   -- señal de control para el DAC
            DIN : out  STD_LOGIC);    -- dato que se envía al DAC
end component;


component control_global is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
            Rst : in  STD_LOGIC;     -- señal de reset
            end_ADC : in  STD_LOGIC;       -- señal que da por terminado la
operación del módulo ADC
            end_random : in  STD_LOGIC;          -- señal que da por
terminado la operación del módulo de encriptación
            end_DAC : in  STD_LOGIC;       -- señal que da por terminado la
operación del módulo DAC
            start_ADC : out  STD_LOGIC;    -- señal que habilita el módulo
ADC
            start_random : out  STD_LOGIC;-- señal que habilita el módulo
de encriptación
            start_DAC : out  STD_LOGIC;    -- señal que habilita el módulo
DAC
            global_st : out  STD_LOGIC_VECTOR (1 downto 0));-- señal que
indica el estado del autómata
end component;
```

```
component pseudo_random is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
           Rst : in  STD_LOGIC;     -- señal de reset
           start_random : in  STD_LOGIC; -- señal que habilita el módulo
           sw : in  STD_LOGIC_VECTOR (5 downto 0);   -- código predefinido
de cifrado, especificado por los interruptores
           b : in  STD_LOGIC_VECTOR (5 downto 0);    -- código introducido
por el usuario a través de los pulsadores
           valor_ADC : in  STD_LOGIC_VECTOR (11 downto 0); -- último valor
binario completo recibido desde el ADC
           end_random : out  STD_LOGIC;        -- señal que da por
terminado la operación del módulo
           valor_DAC : out  STD_LOGIC_VECTOR (11 downto 0));     -- valor
binario a enviar al DAC
end component;

-- señales auxiliares para realizar el port map
signal b : STD_LOGIC_VECTOR(5 downto 0);
signal start_ADC, start_DAC, start_random, end_ADC, end_DAC, end_random :
STD_LOGIC;
signal valor_ADC, valor_DAC: STD_LOGIC_VECTOR(11 downto 0);
signal global_st : STD_LOGIC_VECTOR(1 downto 0);
signal DIN_ADC, DIN_DAC, SCLK_ADC, SCLK_DAC : STD_LOGIC;   -- para la
salida compartida

begin

      with global_st select DIN <=  -- como la salida es compartida se
multiplexa en función del estado del autómata
            DIN_ADC when "01",
            DIN_DAC when "10",
            '0' when others;

      with global_st select SCLK <= -- como la salida es compartida se
multiplexa en función del estado del autómata
            SCLK_ADC when "01",
            SCLK_DAC when "10",
            '0' when others;

      U1 : control_pulsadores port map (clk, Rst, Up0, Down0, Up1, Down1,
b);
      U2 : control_displays port map (clk, b, Disp0, Disp1, Disp2, Disp3,
Seg7);
      U3 : control_ADC port map (clk, Rst, start_ADC, DOUT, end_ADC, CS0,
SCLK_ADC, DIN_ADC, valor_ADC);
      U4 : control_DAC port map (clk, Rst, start_DAC, valor_DAC, end_DAC,
CS1, SCLK_DAC, DIN_DAC);
      U5 : control_global port map (clk, Rst, end_ADC, end_random, end_DAC,
start_ADC, start_random,
                  start_DAC, global_st);
      U6 : pseudo_random port map (clk, Rst, start_random, sw, b,
valor_ADC, end_random, valor_DAC);

end Behavioral;
```

```
#Señal de reloj del sistema
NET "clk" LOC = "M6"; # Reloj del sistema de 50 MHz

# PULSADORES
NET "Up0"  LOC = "G12";
NET "Down0"  LOC = "C11";
NET "Up1"  LOC = "M4";
NET "Down1"  LOC = "A7";

# RESET
NET "Rst"  LOC = "P11";

# SWITCHES
NET "sw<0>" LOC = "K3"; #
NET "sw<1>" LOC = "B4"; #
NET "sw<2>" LOC = "G3"; #
NET "sw<3>" LOC = "F3"; #
NET "sw<4>" LOC = "E2"; #
NET "sw<5>" LOC = "N3"; #

# SEÑALES DE ACTIVACIÓN DE LOS DISPLAYS
NET "Disp3" LOC = "K14"; # SEÑAL = AN0
NET "Disp2" LOC = "M13"; # SEÑAL = AN1
NET "Disp1" LOC = "J12"; # SEÑAL = AN2
NET "Disp0" LOC = "F12"; # SEÑAL = AN3

# 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

# ENTRADA ADC - DAC
NET "DOUT" LOC = "C6"; # "E5"

# SALIDAS ADC - DAC
NET "CS0" LOC = "C13"; # ADC        "S7"
NET "CS1" LOC = "D12"; # DAC        "S8"
NET "SCLK" LOC = "C12"; #           "S5"
NET "DIN" LOC = "A13"; #            "S6"
```

```
----------------------------------------------------------------------------
-------
-- CONTROL GLOBAL
-- Es el encargado de gestionar cuándo tiene que comenzar el
-- procesamiento de cada uno de los bloques que dependen de él.
-- Utiliza un protocolo de comunicación asíncrona basado en una
-- señal de inicio y otra señal de fin (cada bloque comenzará su
-- procesamiento cuando haya terminado el bloque inmediatamente anterior).
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_global is
    Port ( clk : in  STD_LOGIC;     -- señal de reloj
           Rst : in  STD_LOGIC;     -- señal de reset
           end_ADC : in  STD_LOGIC;       -- señal que da por terminado la
operación del módulo ADC
           end_random : in  STD_LOGIC;          -- señal que da por
terminado la operación del módulo de encriptación
           end_DAC : in  STD_LOGIC;       -- señal que da por terminado la
operación del módulo DAC
           start_ADC : out  STD_LOGIC;    -- señal que habilita el módulo
ADC
           start_random : out  STD_LOGIC;-- señal que habilita el módulo
de encriptación
           start_DAC : out  STD_LOGIC;    -- señal que habilita el módulo
DAC
           global_st : out  STD_LOGIC_VECTOR (1 downto 0));-- señal que
indica el estado del autómata
end control_global;

architecture Behavioral of control_global is

-- Declaración de las señales necesarias
signal contador : STD_LOGIC_VECTOR (12 downto 0) := "0000000000000";   --
señal que lleva la cuenta para 125 us
signal cnt_125us : STD_LOGIC := '0';      -- señal que indica que han
pasado 125 us
signal s_global_st : STD_LOGIC_VECTOR (1 downto 0) := "00";-- señal que
indica el estado del autómata

begin

process (clk)     -- este process cuenta 125 us
      begin
            if clk'event and clk='1' then
                  cnt_125us <= '0';
                  contador <= contador + '1';
                  if contador >= "1100001101010" then -- valor máximo de la
cuenta
                        cnt_125us <= '1';
                        contador <= (others => '0');
                  end if;
            end if;
end process;

global_st <= s_global_st;
```

```
start_ADC <= cnt_125us; -- debido a que cada 125 us se pasa al estado 01,
activandose start_ADC

process(clk, s_global_st, Rst, end_ADC, end_random, end_DAC, cnt_125us)
      -- en este process se va cambiando de estado en función de que las
entradas avisen de que se acaban los estados anteriores
      begin
            if Rst = '1' then -- Poner a cero las señales
                  s_global_st <= "00";
                  start_DAC <= '0';
                  start_random <= '0';

            elsif clk'event and clk='1' then
                  start_DAC <= '0'; -- incialización
                  start_random <= '0';    -- incialización

                  CASE s_global_st IS

                        when "00" =>

                              if (cnt_125us = '0') then
                                    s_global_st <= "00";
                              else  -- si han pasado 125 us, se acaba
reposo (01), el estado siguente es ADC (01)
                                    s_global_st <= "01";
                              end if;

                        when "01" =>

                              if (end_ADC = '0') then
                                    s_global_st <= "01";
                              else  -- si se acaba ADC (01), el estado
siguente es encriptación (11)
                                    start_random <= '1';
                                    s_global_st <= "11";
                              end if;

                        when "11" =>

                              if (end_random = '0') then
                                    s_global_st <= "11";
                              else  -- si se acaba encriptación (11), el
estado siguente es DAC (10)
                                    start_DAC <= '1';
                                    s_global_st <= "10";
                              end if;

                        when "10" =>

                              if (end_DAC = '0') then
                                    s_global_st <= "10";
                              else  -- si se acaba DAC (01), el estado
siguente es reposo (00)
                                    s_global_st <= "00";
                              end if;

                        when others =>
                              s_global_st <= "00";

                  END CASE;
            end if;
```

```
end process;

end Behavioral;
```

El bloque de control global es el encargado de gestionar cuándo tiene que comenzar el procesamiento de cada uno de los bloques que dependen de él (control del bloque del ADC, bloque de encriptación y control del bloque del DAC).

Utiliza un protocolo de comunicación asíncrona basado en una señal de inicio y otra señal de fin, es decir, cada bloque comenzará su procesamiento cuando haya terminado el bloque inmediatamente anterior.

Puesto que en las especificaciones se indica que la frecuencia de muestreo deben ser 8 kHz, un contador interno generará cada 125 µs la señal de inicio del primero de los bloques, por tanto cada 125 µs el autómata ha pasado por cada uno de sus cuatro estados (00 = Reposo, 01 = ADC, 11 = encriptación y 10 = DAC).

4.7 Simulaciones adicionales

Incluya aquí las simulaciones adicionales que haya realizado con los ficheros de *testbench* que haya utilizado para las pruebas (pulsadores, displays, control ADC, DAC, etc.). ***Incluya las simulaciones de verificación de cada entidad en sub-apartados diferentes.***

Por cada uno:

- Incluir el código del fichero de *testbench* debidamente comentado.
- Incluir la captura de pantalla del simulador ISIM donde se aprecie de forma suficiente el funcionamiento del módulo.
- Incluir un breve comentario sobre sus conclusiones acerca de la simulación.

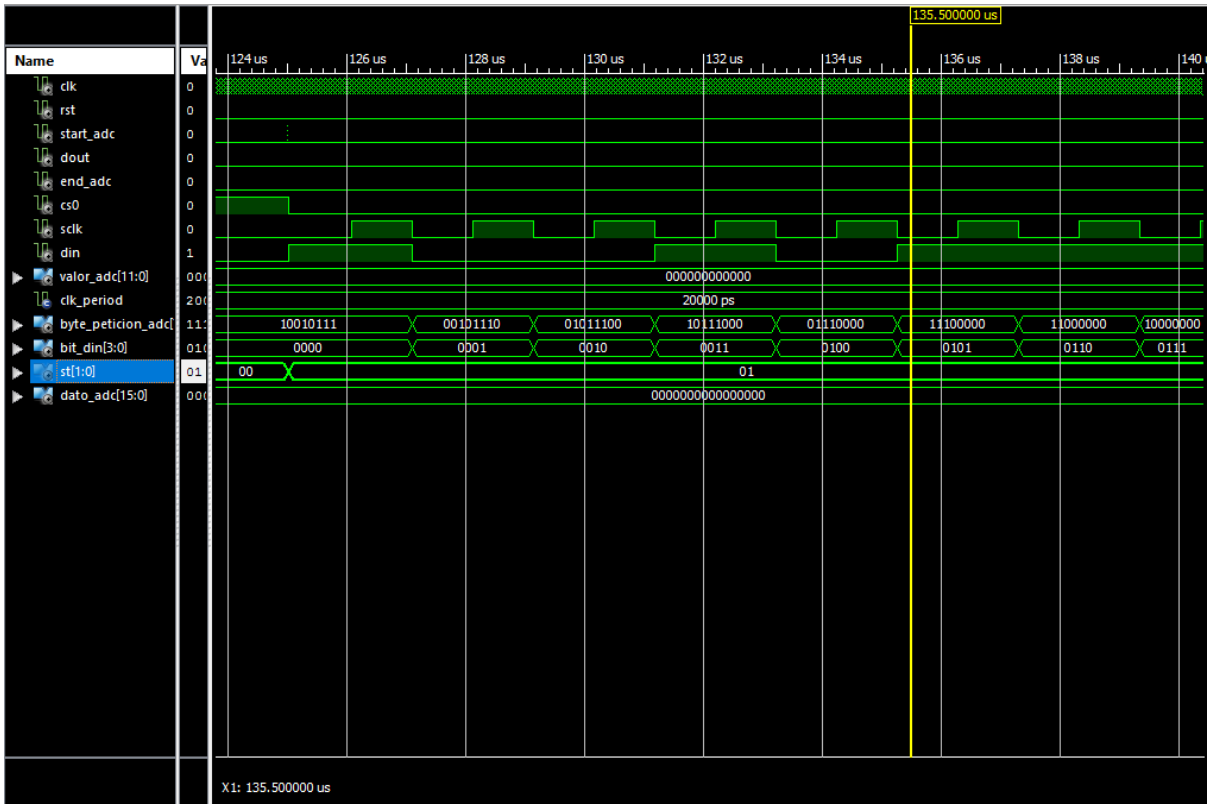
4.7.1 Control ADC

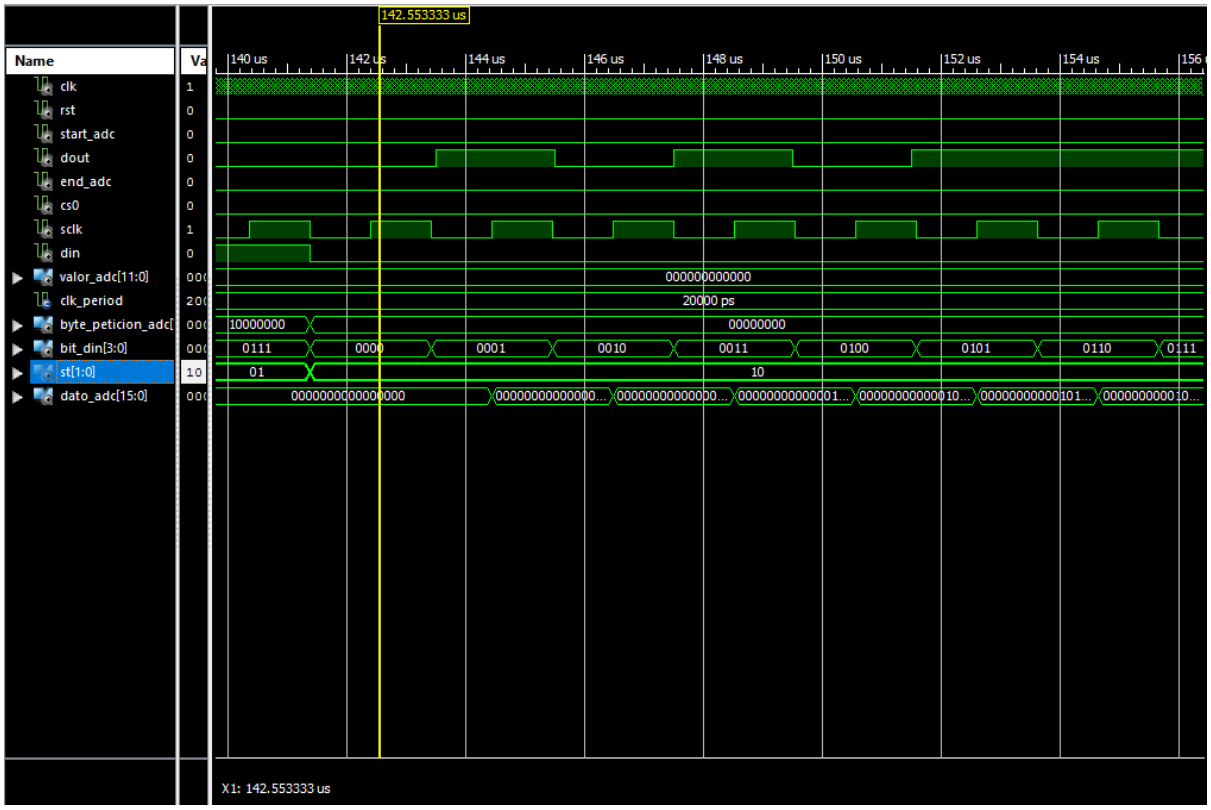
```
--------------------------------------------------------------------------------
-----
-- TEST BENCH DE CONTROL ADC
--------------------------------------------------------------------------------
-----
library STD;
use std.textio.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_textio.all;
use ieee.std_logic_unsigned.all;

ENTITY control_ADC_tb IS
END control_ADC_tb;

ARCHITECTURE behavior OF control_ADC_tb IS

    COMPONENT control_ADC
    PORT(clk : IN  std_logic;         -- señal de reloj
         Rst : IN  std_logic;         -- señal de reset
         start_ADC : IN  std_logic;-- señal que habilita el módulo
         DOUT : IN  std_logic;      -- señal serie que se recibe desde el
ADC
         end_ADC : OUT  std_logic;  -- señal que da por terminado la
operación del módulo
         CS0 : OUT  std_logic;      --  señal de control para el ADC
         SCLK : OUT  std_logic;     -- señal de control para el ADC
         DIN : OUT  std_logic;      -- señal de petición par el ADC
         valor_ADC : OUT  std_logic_vector(11 downto 0));  -- último valor
binario completo recibido desde el ADC
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal Rst : std_logic := '0';
   signal start_ADC : std_logic := '0';
   signal DOUT : std_logic := '0';

      --Outputs
```

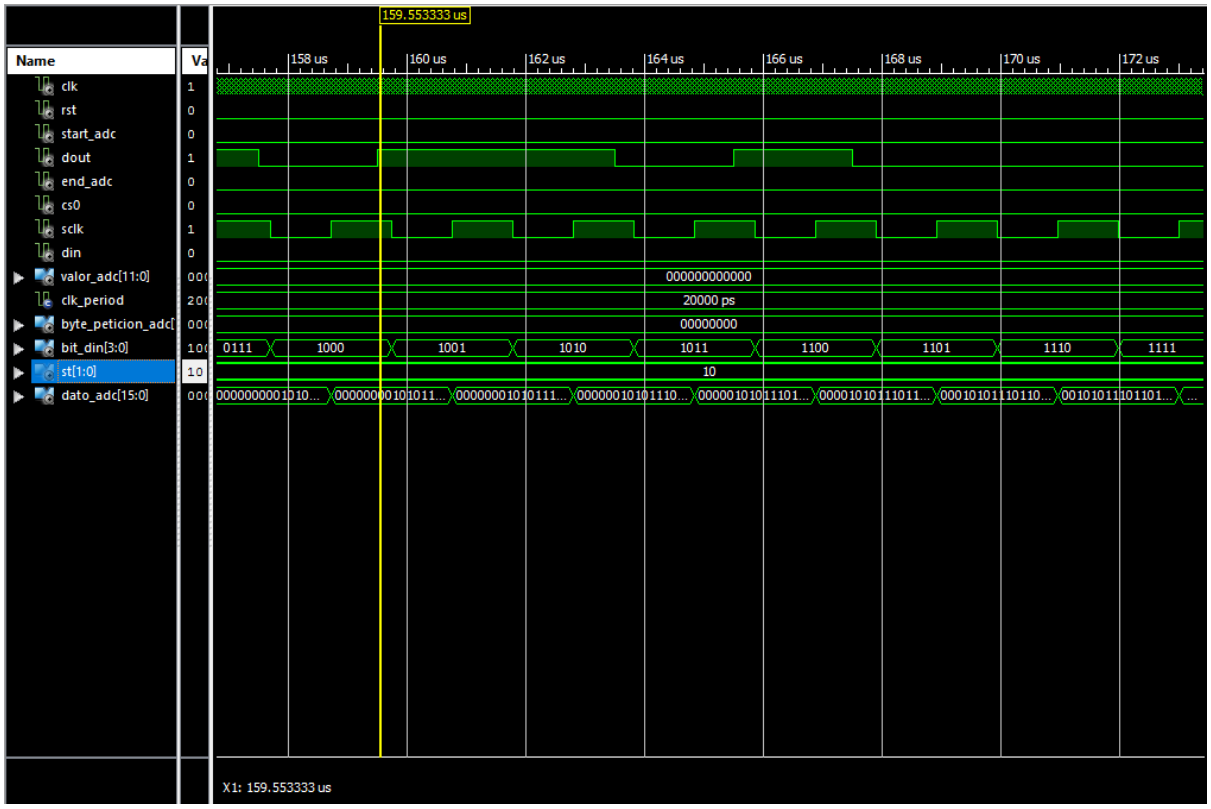
```
   signal end_ADC : std_logic;
   signal CS0 : std_logic;
   signal SCLK : std_logic;
   signal DIN : std_logic;
   signal valor_ADC : std_logic_vector(11 downto 0);

   -- Clock period definitions
   constant clk_period : time := 20 ns;          -- 50Mhz

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   unit_control_ADC: control_ADC PORT MAP (
          clk => clk,
          Rst => Rst,
          start_ADC => start_ADC,
          DOUT => DOUT,
          end_ADC => end_ADC,
          CS0 => CS0,
          SCLK => SCLK,
          DIN => DIN,
          valor_ADC => valor_ADC
        );

   -- Clock process definitions
   clk_process :process
   begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
   end process;

      -- start_ADC process
      start_ADC_process :process
   begin
            start_ADC <= '0';
            wait for 125us;
            start_ADC <= '1';
            wait for 20ns;
            start_ADC <= '0';
   end process;

   -- Stimulus process
   stim_proc: process
   begin

            -- insert stimulus here
            DOUT <='0'; -- valor cualquiera inicial
            rst<='1';   -- se ponen todas las señales a 0
            wait for 500 ns;
            rst <='0';  -- a la espera de start_ADC para empezar el proceso
            wait for 143 us;
            DOUT <='1'; -- se van metiendo valores a DOUT
            wait for 2 us;    -- 2 us es el valor del periodo de SCLK
            DOUT <='0';
            wait for 2 us;
            DOUT <='1';
            wait for 2 us;
            DOUT <='0';
            wait for 2 us;
```

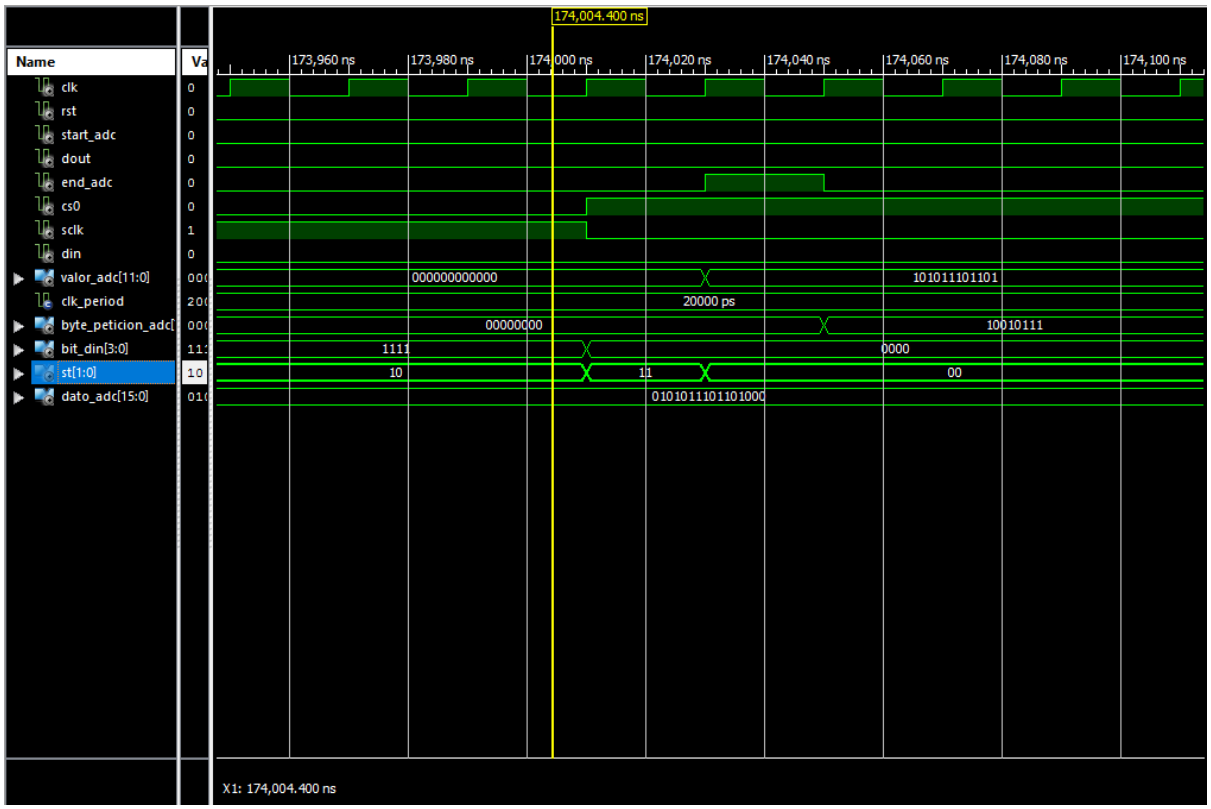
```
            DOUT <='1';
            wait for 2 us;
            DOUT <='1';
            wait for 2 us;
            DOUT <='1';
            wait for 2 us;
            DOUT <='0';
            wait for 2 us;
            DOUT <='1';
            wait for 2 us;
            DOUT <='1';
            wait for 2 us;
            DOUT <='0';
            wait for 2 us;
            DOUT <='1';
            wait for 2 us;
            DOUT <='0';
            wait;
    end process;

END;
```

A continuación enseñamos los cronogramas que verifican el correcto funcionamiento del bloque de control del ADC:




142.553333 us
Name
clk
rst
start_adc
dout
end_adc
cs0
sclk
din
valor_adc[11:0]
clk_period
byte_peticion_adc[
bit_din[3:0]
st[1:0]
dato_adc[15:0]
140 us
142 us
144 us
146 us
148 us
150 us
152 us
154 us
000000000000
20000 ps
10000000
00000000
0111
0000
0001
0010
0011
0100
0101
0110
0111
01
10
X1: 142.553333 us


159.553333 us
Name
clk
rst
start_adc
dout
end_adc
cs0
sclk
din
valor_adc[11:0]
clk_period
byte_peticion_adc[
bit_din[3:0]
st[1:0]
dato_adc[15:0]
158 us
160 us
162 us
164 us
166 us
168 us
170 us
172 us
000000000000
20000 ps
00000000
0111
1000
1001
1010
1011
1100
1101
1110
1111
10
X1: 159.553333 us



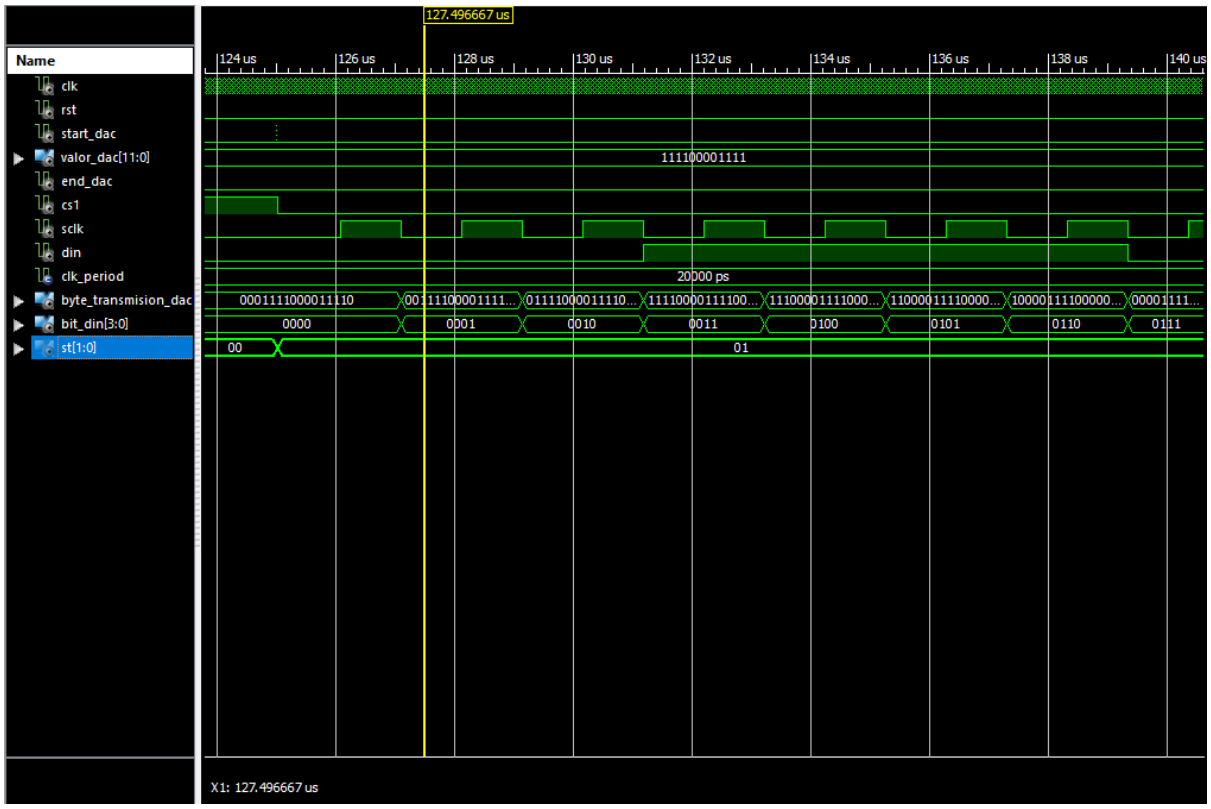
En la primera imagen se observa que hasta que la señal start_ADC no se pone a 1, la señal CS0 está a 1 y el autómata está en el estado de reposo (00). Entonces se suceden los tres estados restantes (01, 10, 11). Durante los estados 01 y 10 se ve cómo va la cuenta tanto de la transmisión de bits de petición, como de la recepción de bits en serie mediante la señal bit_DIN. El estado 11 dura un solo flanco de reloj, donde se asigna el valor de salida a valor_ADC y se pone la señal end_ADC a 1 durante un flanco de reloj.

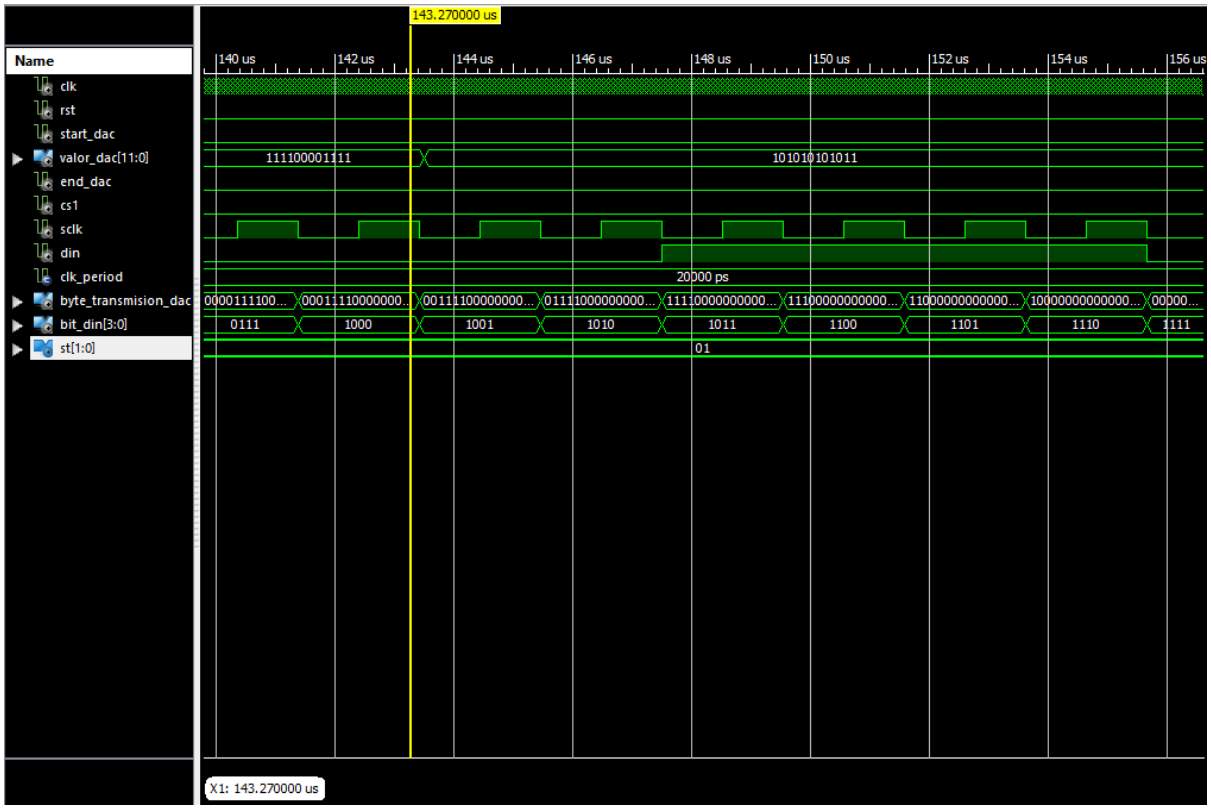
4.7.2 Control DAC

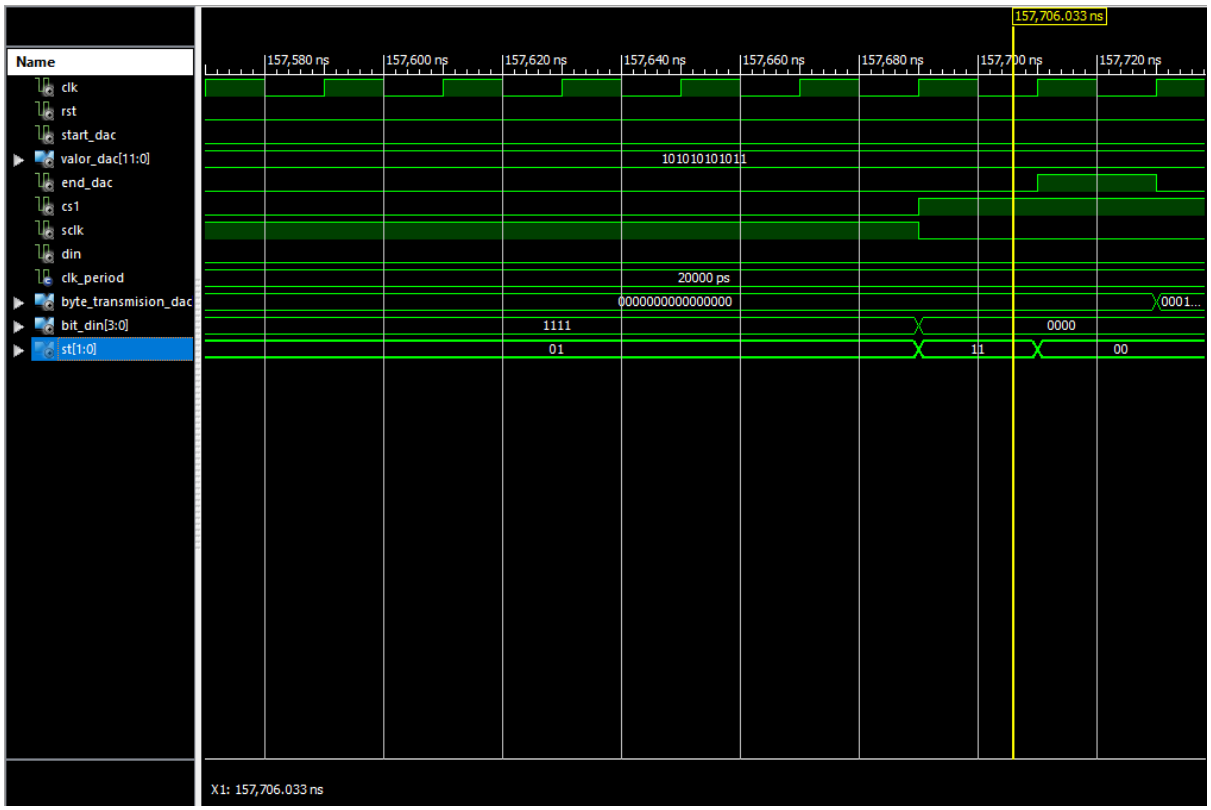
```
--------------------------------------------------------------------------------
-- TEST BENCH DE CONTROL DAC
--------------------------------------------------------------------------------
library STD;
use std.textio.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_textio.all;
use ieee.std_logic_unsigned.all;

ENTITY control_DAC_tb IS
END control_DAC_tb;

ARCHITECTURE behavior OF control_DAC_tb IS

    COMPONENT control_DAC
    PORT(clk : IN  std_logic; -- señal de reloj
         Rst : IN  std_logic; -- señal de reset
         start_DAC : IN  std_logic; -- señal que habilita el módulo
         valor_DAC : IN  std_logic_vector(11 downto 0);    -- valor binario
a enviar al DAC
         end_DAC : OUT  std_logic;        -- señal que da por terminado la
operación del módulo
         CS1 : OUT  std_logic;      --  señal de control para el DAC
         SCLK : OUT  std_logic;     -- señal de control para el DAC
         DIN : OUT  std_logic);      -- dato que se envía al DAC
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal Rst : std_logic := '0';
   signal start_DAC : std_logic := '0';
   signal valor_DAC : std_logic_vector(11 downto 0) := (others => '0');

      --Outputs
   signal end_DAC : std_logic;
   signal CS1 : std_logic;
   signal SCLK : std_logic;
   signal DIN : std_logic;

   -- Clock period definitions
   constant clk_period : time := 20 ns;          -- 50Mhz

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: control_DAC PORT MAP (
          clk => clk,
          Rst => Rst,
          start_DAC => start_DAC,
          valor_DAC => valor_DAC,
          end_DAC => end_DAC,
          CS1 => CS1,
          SCLK => SCLK,
```

```
          DIN => DIN
        );

   -- Clock process definitions
   clk_process :process
   begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
   end process;

      -- start_DAC process
      start_DAC_process :process
   begin
            start_DAC <= '0';
            wait for 125us;
            start_DAC <= '1';
            wait for 20ns;
            start_DAC <= '0';
   end process;

   -- Stimulus process
   stim_proc: process
   begin

      -- insert stimulus here
            valor_DAC <="111100001111";   -- un valor inicial cualquiera a
observar
            rst<='1';   -- se ponen todas las señales a 0
            wait for 500 ns;
            rst <='0';  -- a la espera de start_DAC para empezar el proceso
            wait for 143 us;
            valor_DAC <="101010101011";   -- otro valor cualquiera a
observar
      wait;
   end process;

END;
```

A continuación enseñamos los cronogramas que verifican el correcto funcionamiento del bloque de control del DAC:


127.496667 us
Name
clk
rst
start_dac
valor_dac[11:0]
end_dac
cs1
sclk
din
clk_period
byte_transmision_dac
bit_din[3:0]
st[1:0]
124 us
126 us
128 us
130 us
132 us
134 us
136 us
138 us
140 us
111100001111
20000 ps
0001111000011110
0011110000111...
0111100001111...
1111000011110...
1110000111100...
1100001111000...
1000011110000...
0000111100000...
0000
0001
0010
0011
0100
0101
0110
0111
00
01
X1: 127.496667 us


143.270000 us
Name
clk
rst
start_dac
valor_dac[11:0]
end_dac
cs1
sclk
din
clk_period
byte_transmision_dac
bit_din[3:0]
st[1:0]
140 us
142 us
144 us
146 us
148 us
150 us
152 us
154 us
156 us
111100001111
101010101011
20000 ps
0000111100...
0001111000000...
0011110000000...
0111100000000...
1111000000000...
1110000000000...
1100000000000...
1000000000000...
0000...
0111
1000
1001
1010
1011
1100
1101
1110
1111
01
X1: 143.270000 us



En la primera imagen se observa que hasta que la señal start_DAC no se pone a 1, la señal CS1 está a 1 y el autómata está en el estado de reposo (00). Entonces se suceden los dos estados restantes (01, 11). Durante el estado 01 se ve cómo va la cuenta de la transmisión de bits de petición mediante la señal bit_DIN. El estado 11 dura un solo flanco de reloj, donde se asigna el valor de salida a valor_DAC y se pone la señal end_DAC a 1 durante un flanco de reloj.

5. MEJORAS

Si ha realizado mejoras incluya un apartado por cada una de ellas.

Si se trata de circuitos analógicos:

- Detalle los cálculos de diseño.
- Si ha realizado alguna medida incluya las capturas de pantalla o valores obtenidos

Si se trata de módulos VHDL:

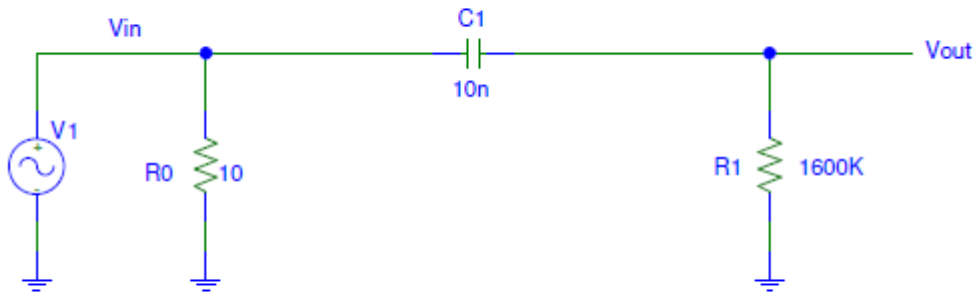
- Incluya un diagrama de bloques completo de la mejora
- Incluya el código correspondiente a cada módulo debidamente comentado.

5.1 Simulación de los módulos analógicos con 5Spice

Antes de detallar nuestras simulaciones nos gustaría dejar constancia de que todos los valores utilizados son comerciales. Es decir, no todos se corresponden con los valores teóricos.

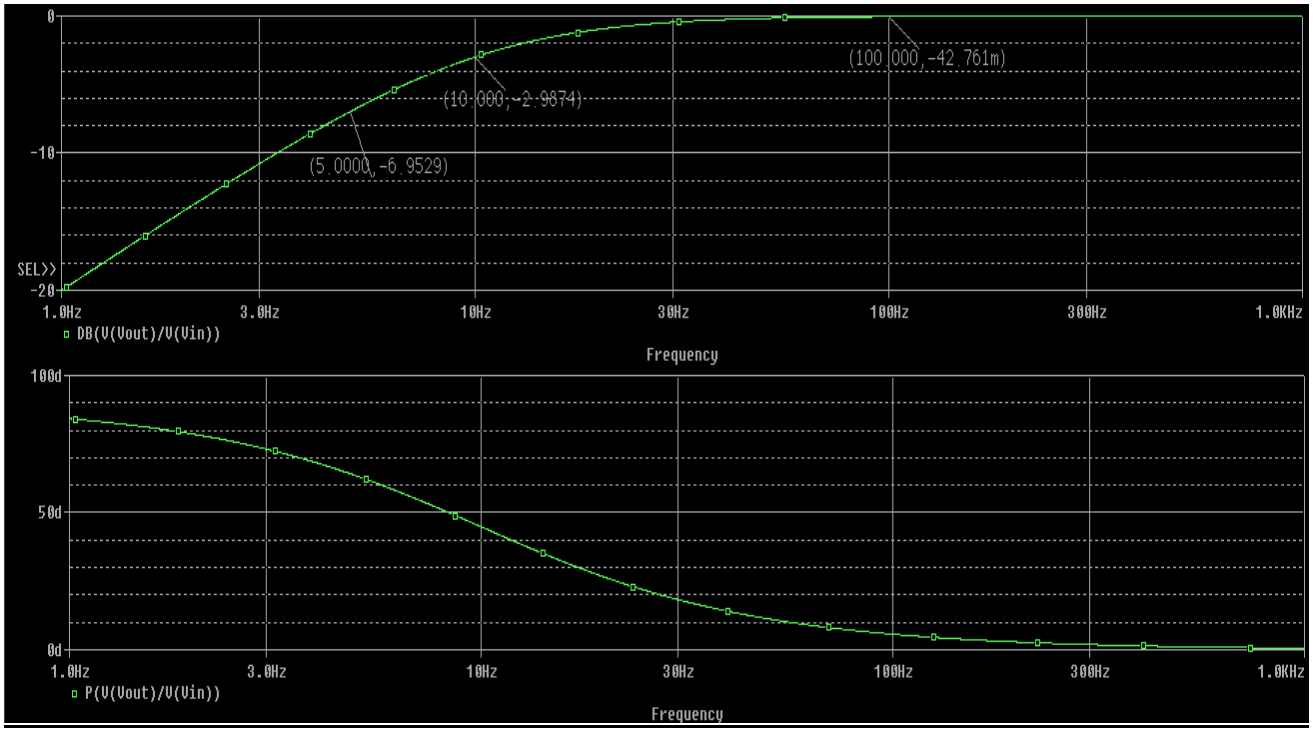
A la hora de hacer las simulaciones en el barrido temporal metemos como señal de entrada del circuito una señal de una amplitud de 0.1V (0.2Vpp), esperando obtener así una señal de salida acorde al máximo estipulado para el auricular.

5.1.1 Bloque de adaptación de impedancias y eliminación de la componente continua.



El valor de la resistencia y del condensador están detallados en la parte analógica de esta memoria (Pag 5-6).

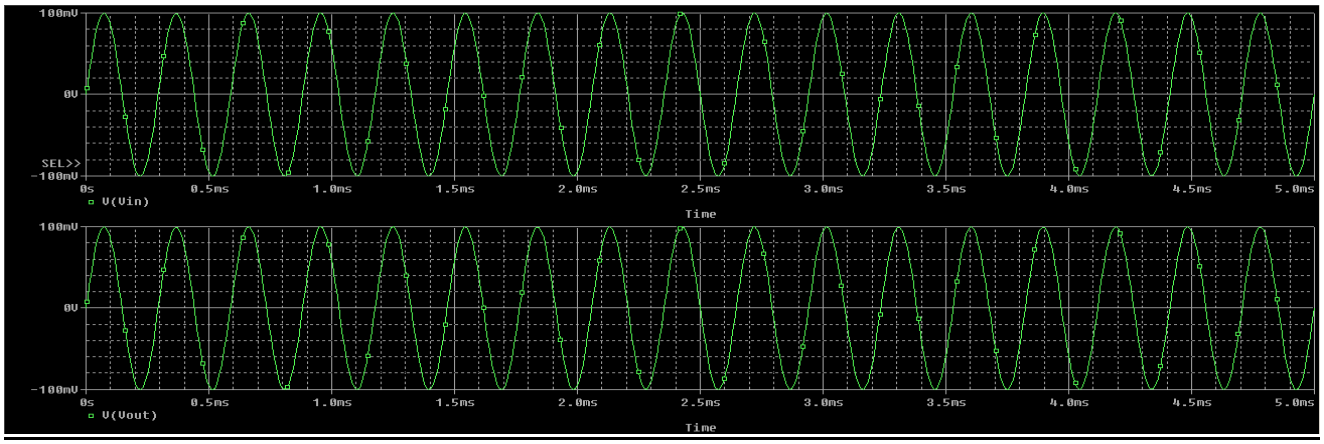
Simulación en AC

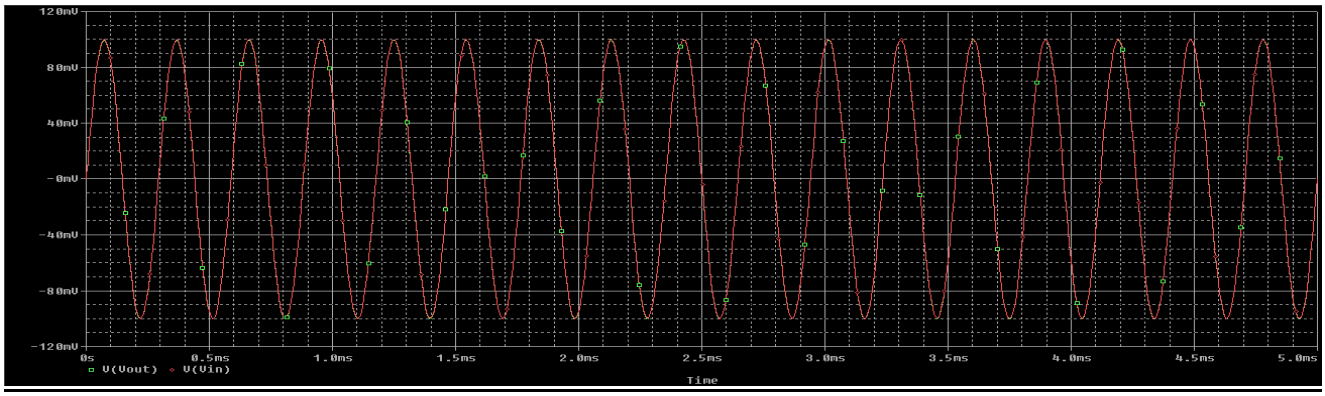


Como se puede comprobar, la simulación se corresponde con los cálculos teóricos, obteniendo así un filtro paso alto con una frecuencia de corte de 10Hz.

Los valores indicados hacen referencia a las ganancias pedidas en el apartado de analógica.

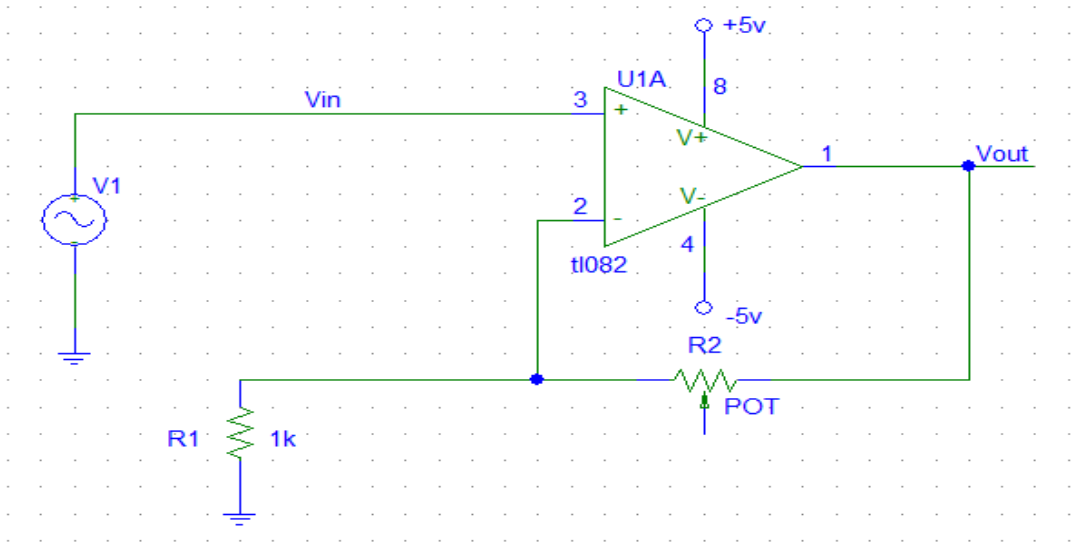
Simulación Temporal





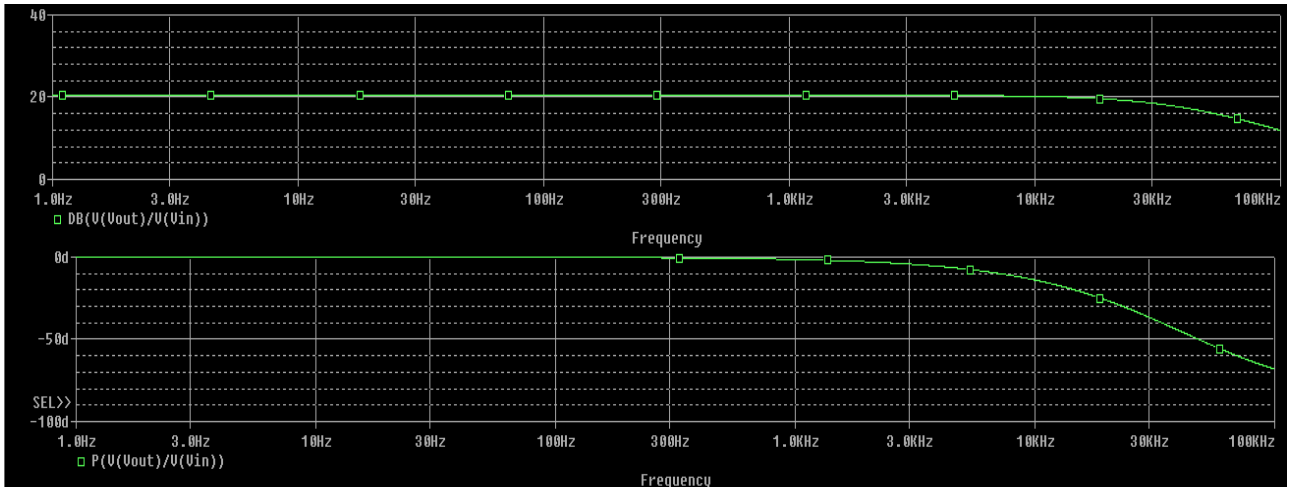
Como se puede observar, en esta parte del circuito no se aprecia tanto una variación de amplitud como de desfase. Teniendo una señal de entrada y salida similares.

5.1.2 Bloque amplificador de ganancia variable

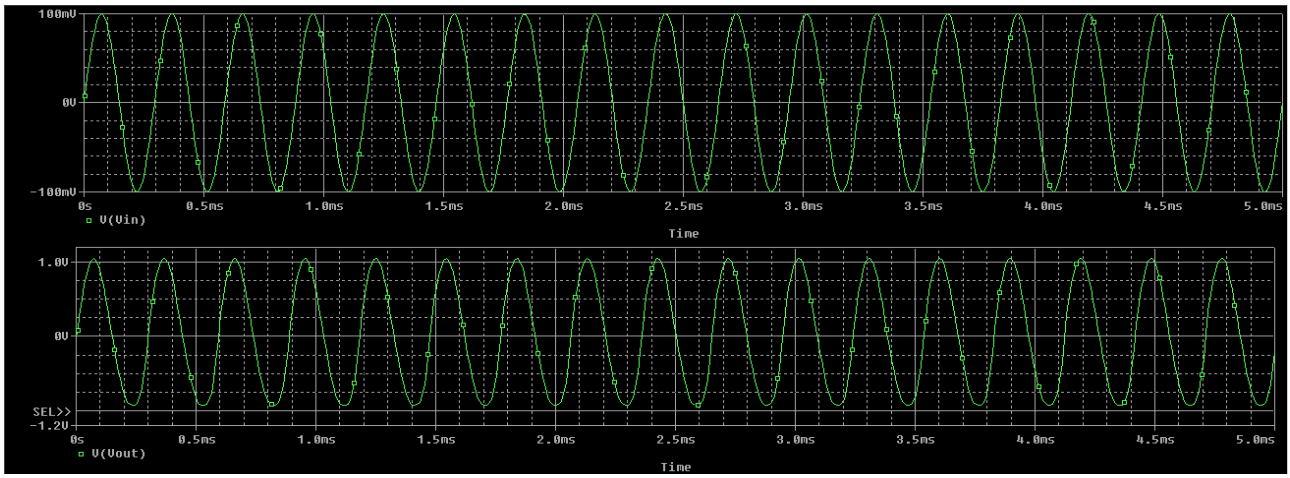


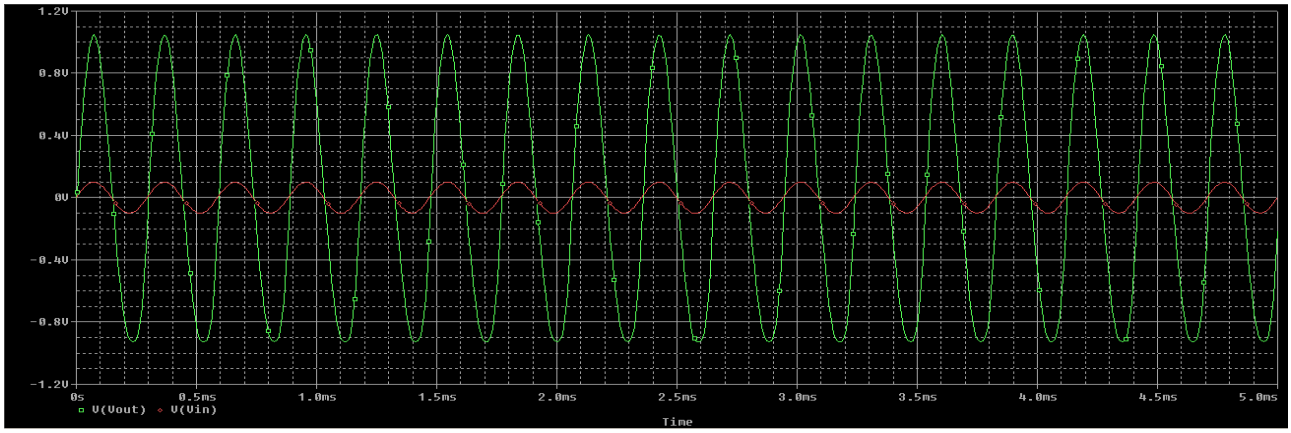
El valor de las resistencias y del los condensadores están detallados en la parte analógica de esta memoria (Pag 7). El amplificador está alimentado a ±Vcc.

Simulación en AC



Simulación Temporal

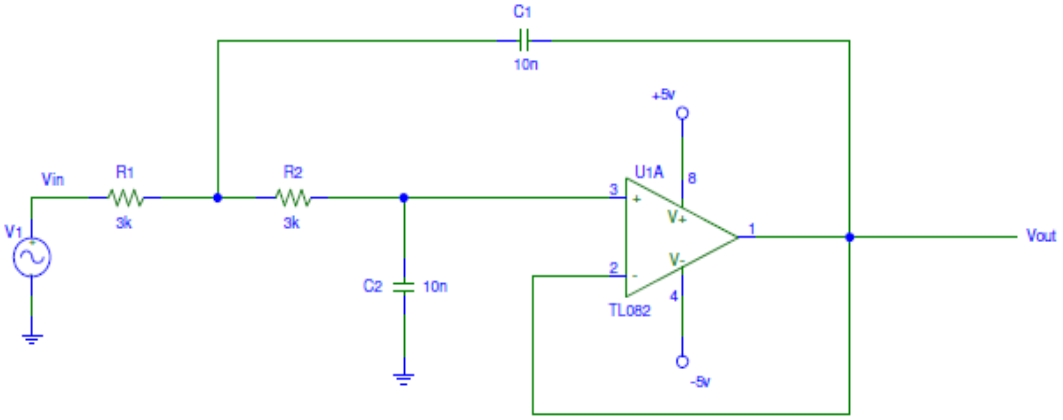




Como se puede observar, se produce en esta parte del circuito una ganancia de la señal de 20 dB, obteniendo así una señal de salida de 2Vpp (valor máximos pedido).

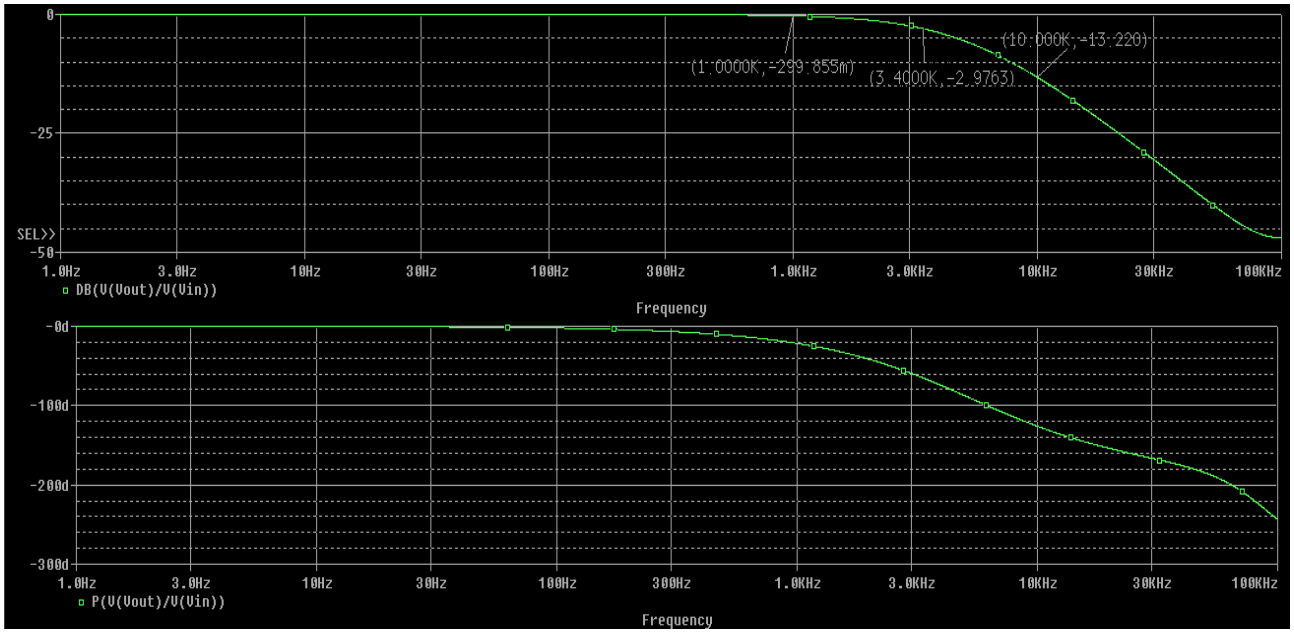
El desfase en este módulo es nulo.

5.1.3 Filtro antisolapamiento



El valor de las resistencias y del los condensadores están detallados en la parte analógica de esta memoria (Pag 8-9). El amplificador está alimentado a ±Vcc.

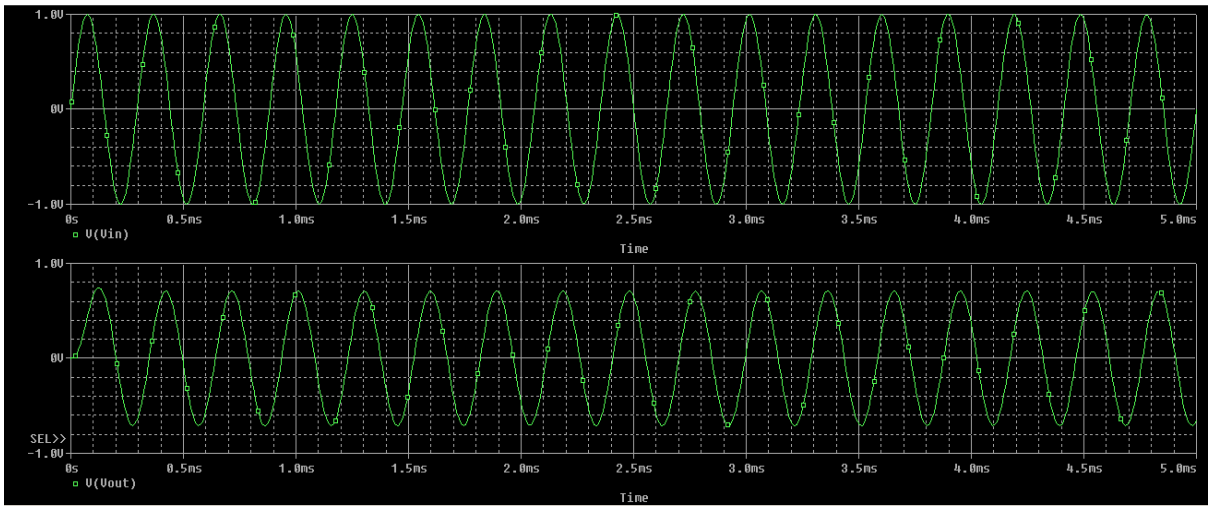
Simulación en AC.

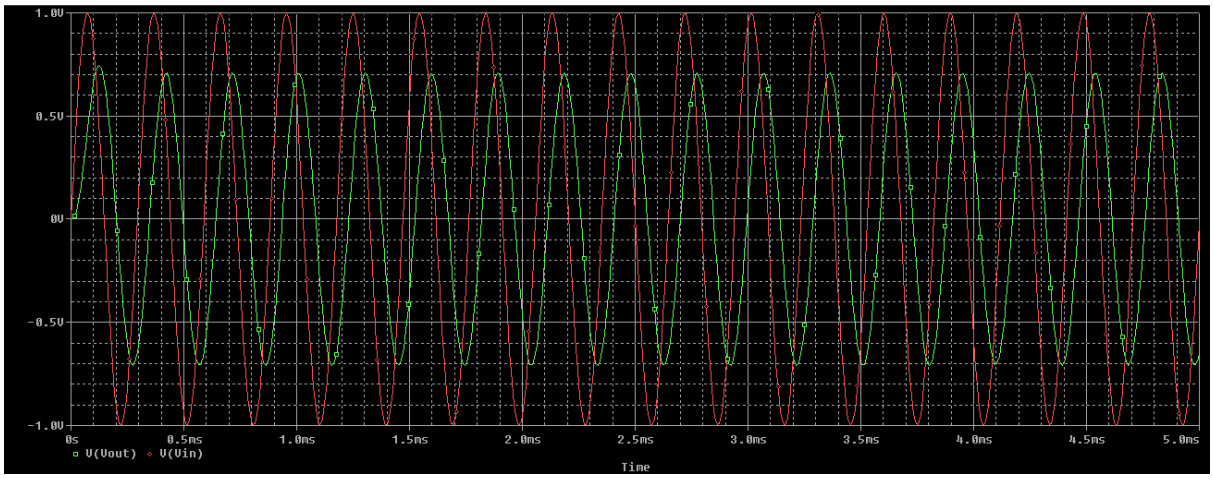


Como se puede comprobar, la simulación corresponde con los cálculos teóricos, obteniendo así un filtro paso bajo con una frecuencia de corte de 3.4KHz.

Los valores indicados hacen referencia a las ganancias pedidas en el apartado de analógica.

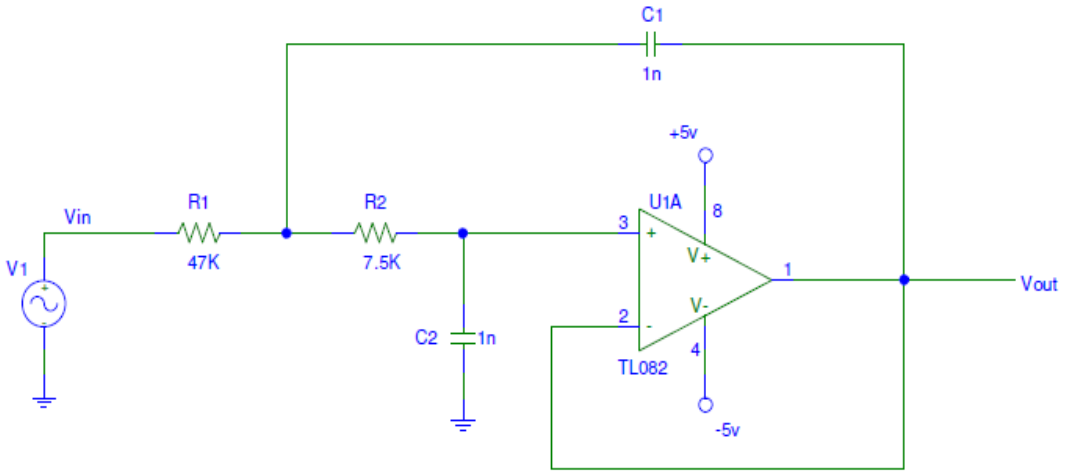
Simulación Temporal





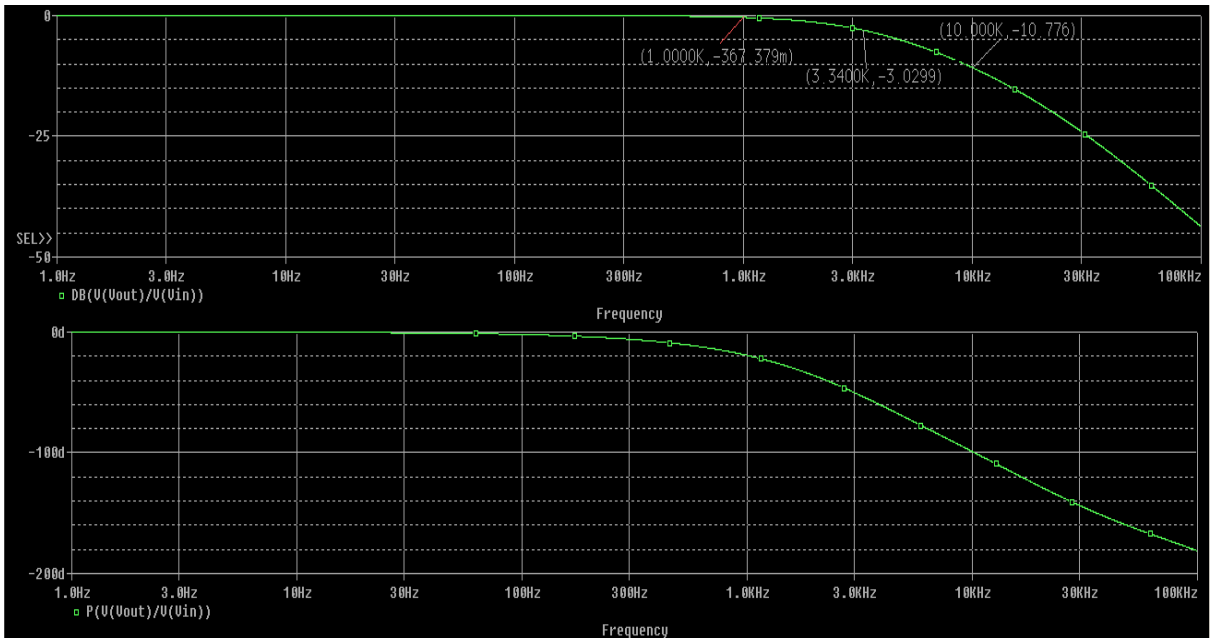
Como se puede observar y como le corresponde a esta sección del circuito, para una señal de entrada con una frecuencia de 3400Hz (frecuencia de corte del filtro) se obtiene una señal de salida atenuada y con un cierto desfase.

5.1.4 Filtro de reconstrucción



El valor de las resistencias y de los condensadores están detallados en la parte analógica de esta memoria (Pag 10-11). El amplificador está alimentado a ±Vcc.

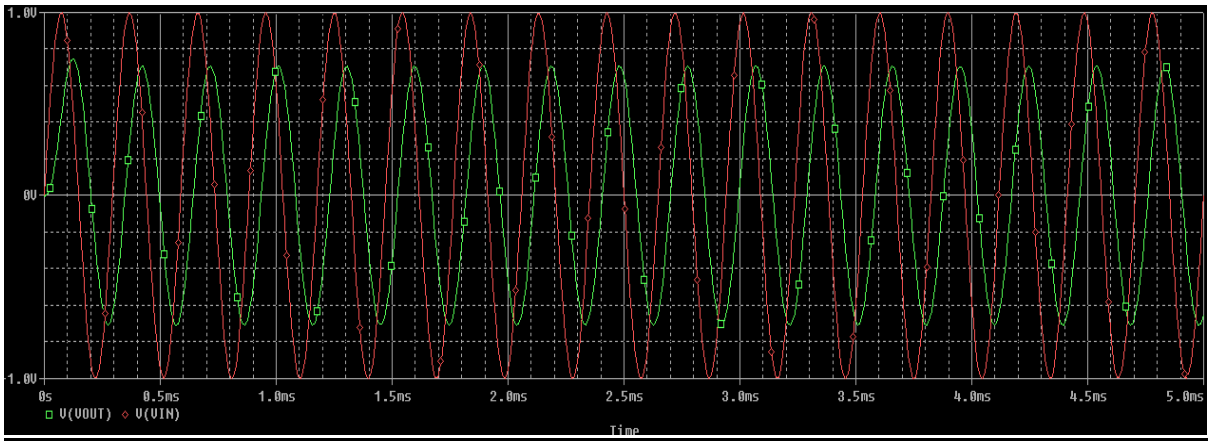
Simulación en AC



Como se puede comprobar, la simulación corresponde con los cálculos teóricos, obteniendo así un filtro paso bajo con una frecuencia de corte de 3.4KHz.

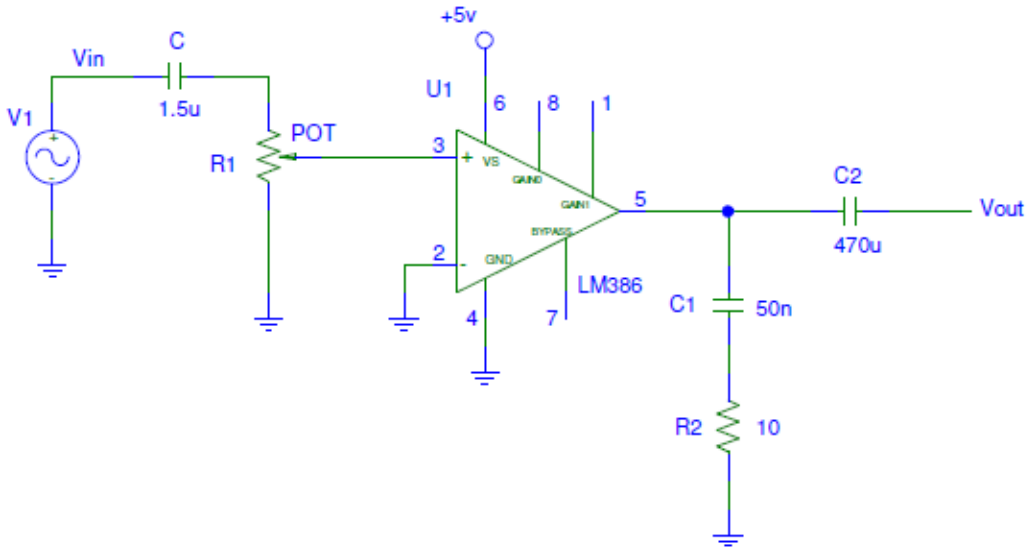
Los valores indicados hacen referencia a las ganancias pedidas en el apartado de analógica.

Simulación Temporal



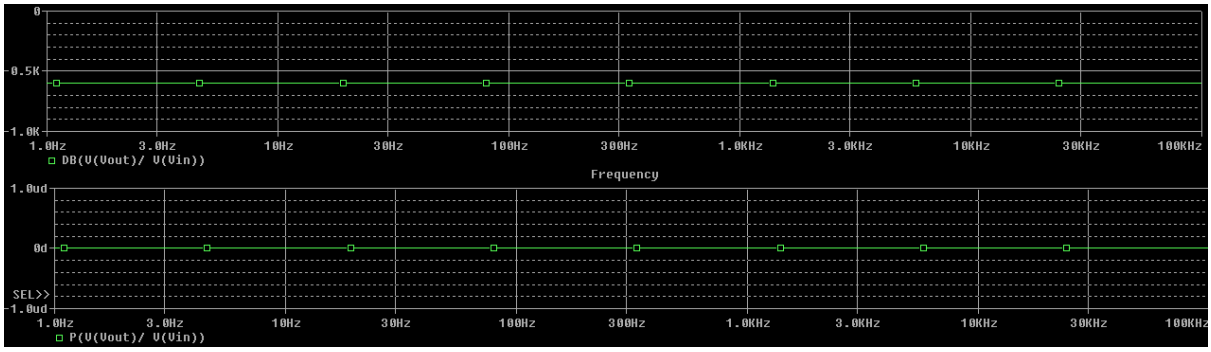
Como se puede observar y como le corresponde a esta sección del circuito, para una señal de entrada con una frecuencia de 3400Hz (frecuencia de corte del filtro) se obtiene una señal de salida atenuada y con un cierto desfase (correspondiente a un filtro con dos polos).

5.1.5 Amplificador de audio de salida basado en el LM386

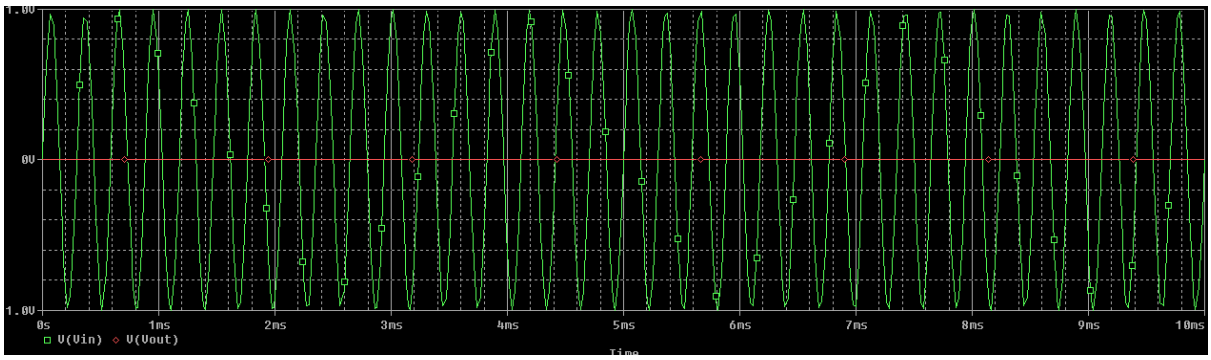


El valor de las resistencias y de los condensadores están detallados en la parte analógica de esta memoria (Pag 12). El amplificador está alimentado a ±Vcc.

Simulación en AC



Simulación Temporal

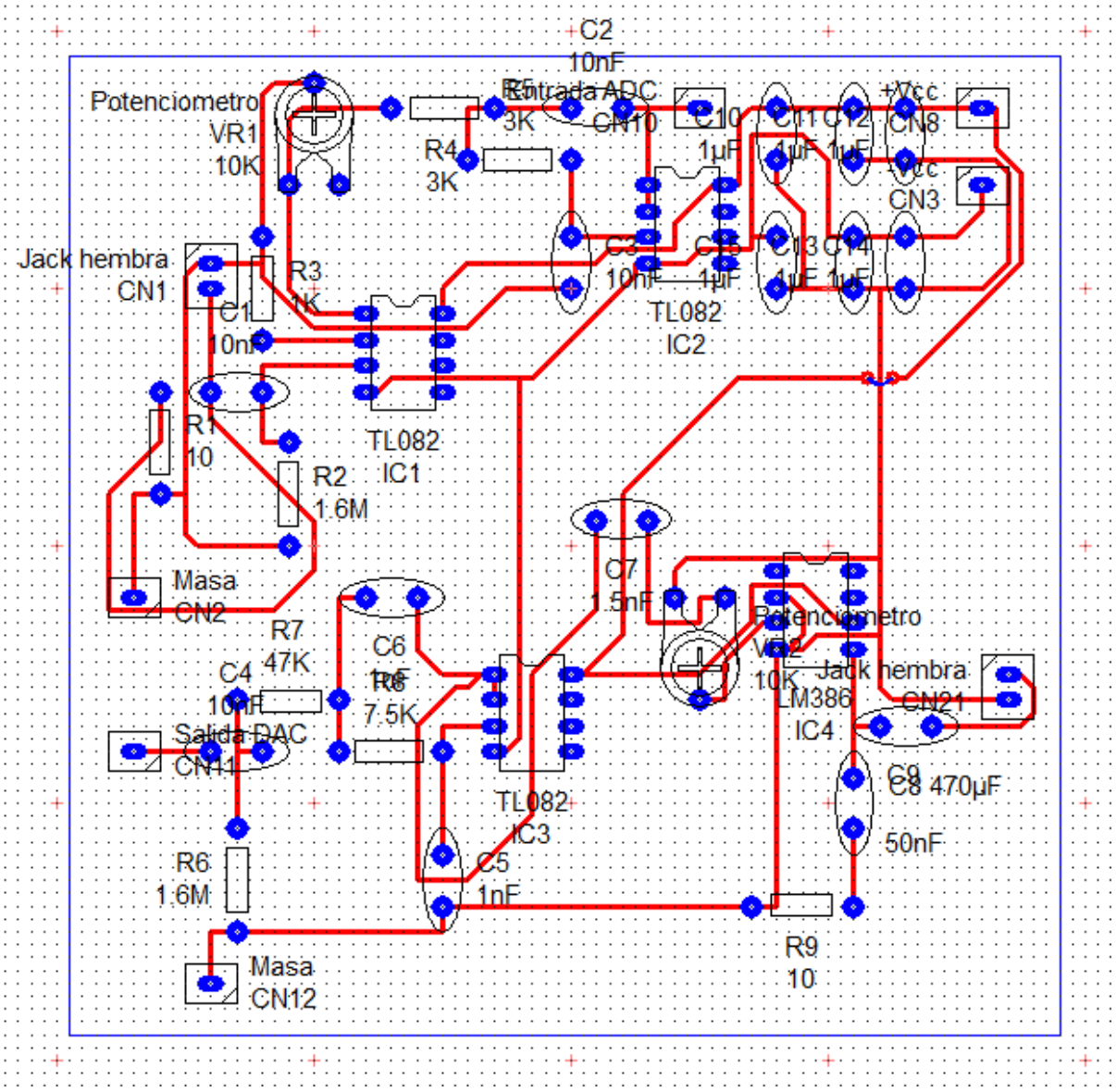


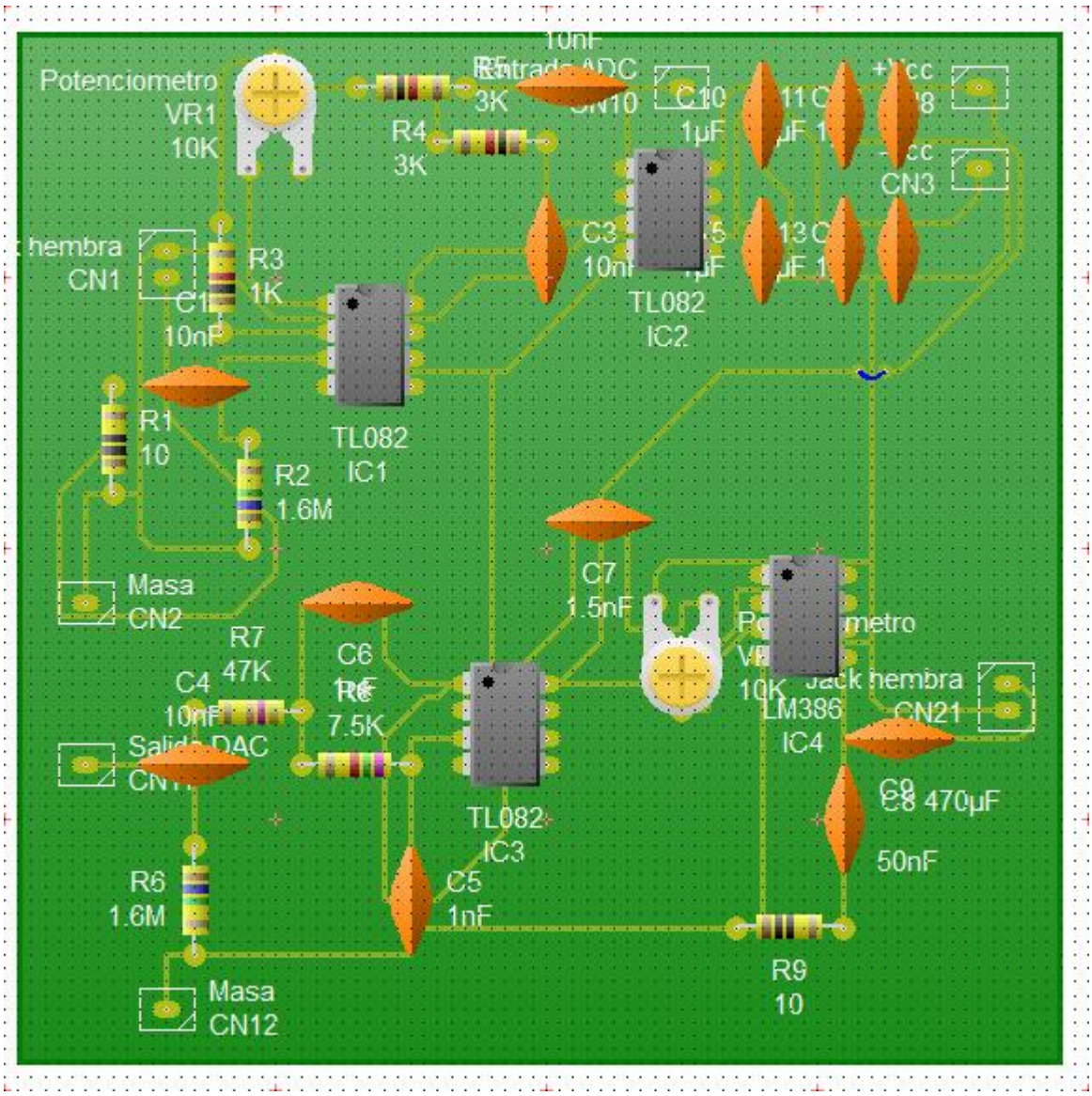
La simulación mediante la herramienta Pspice ha resultado imposible. Esto es debido a que el amplificador LM386 acarrea un error en el programa Probe y no consigue simularlo.

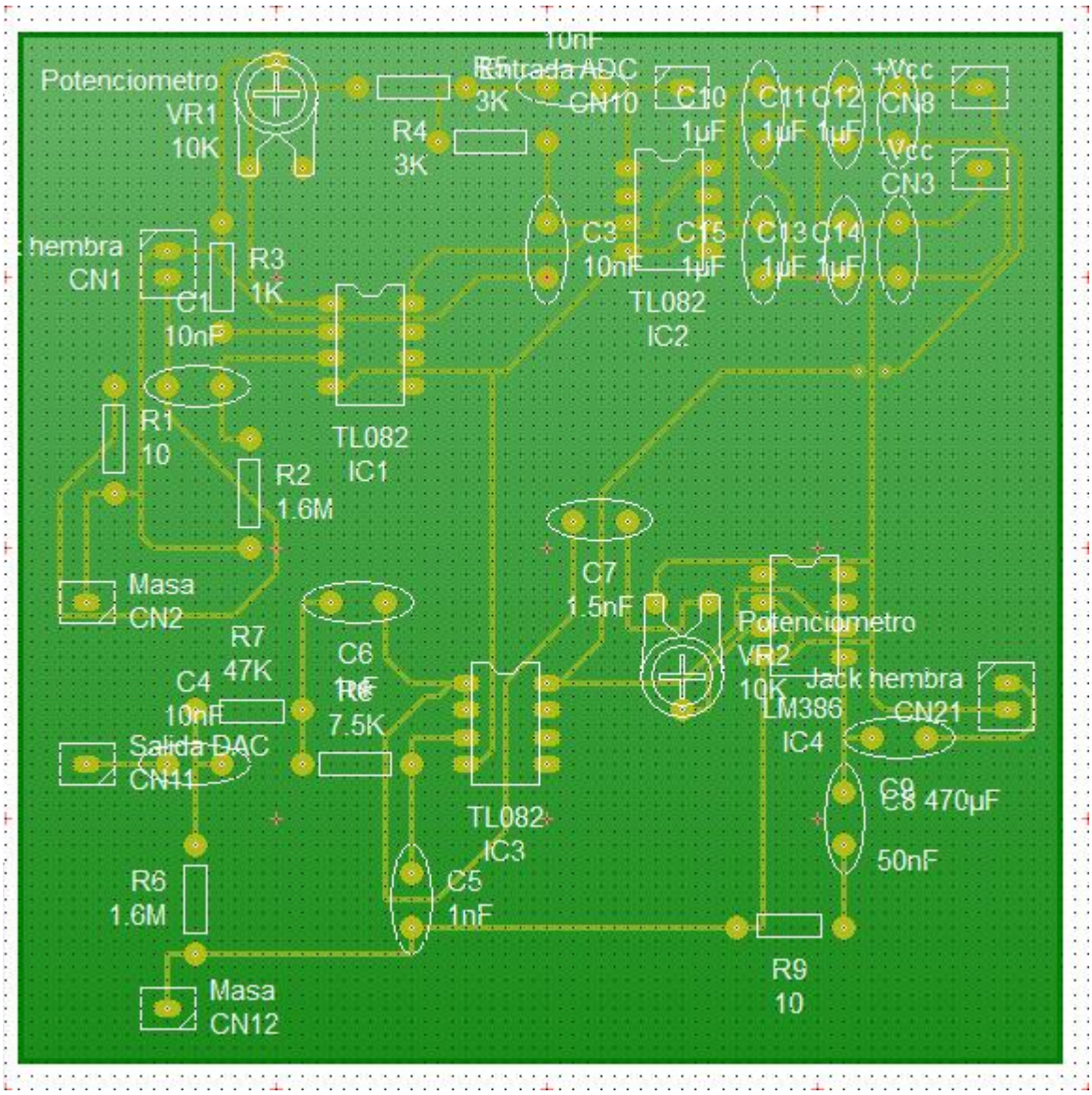
Las imágenes han sido implementadas mediante otro amplificador (sin conseguir el resultado esperado).

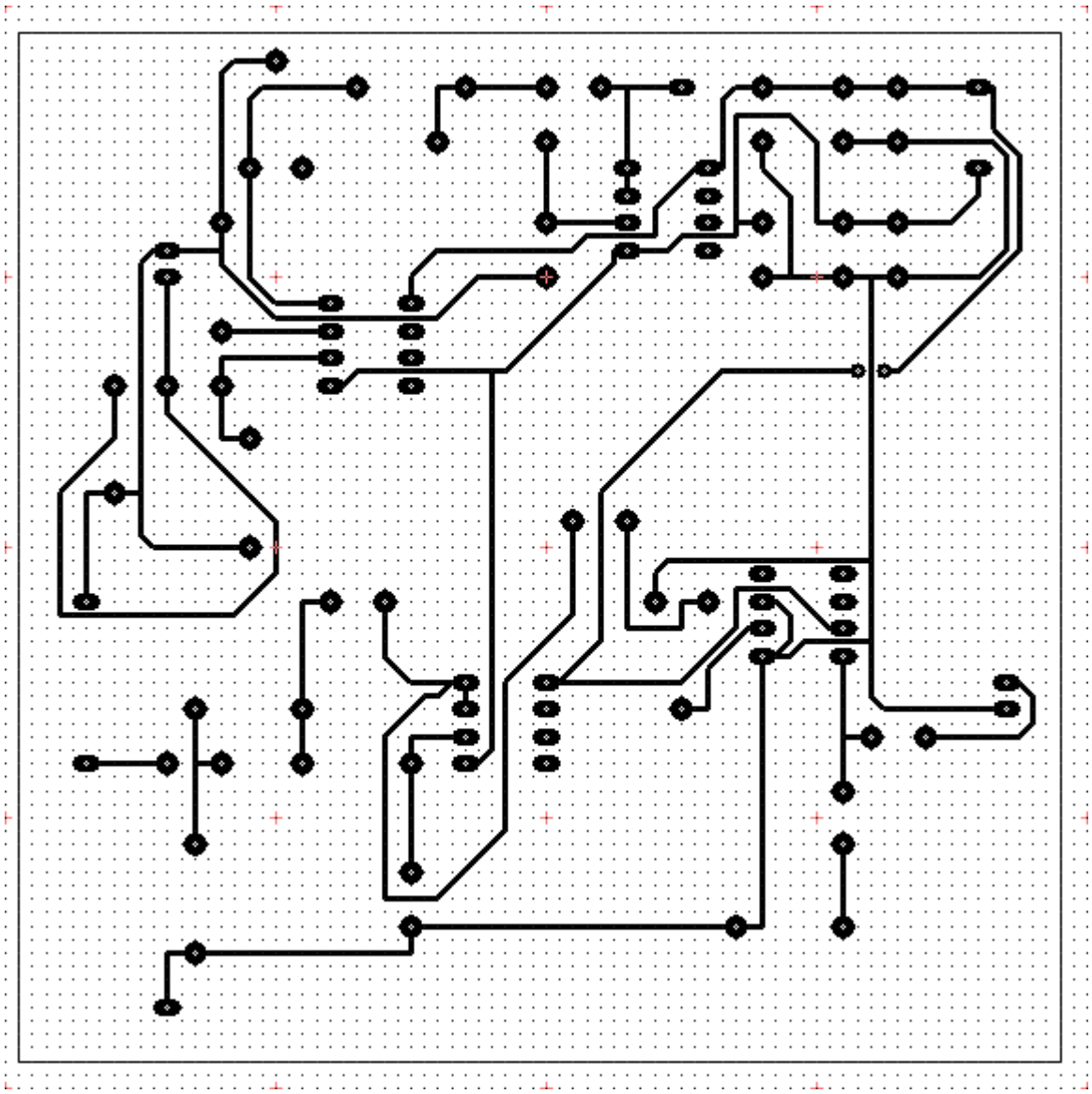
5.2 Realización de los sistemas analógicos en PCB

A continuación observamos la imagen de la PCB con los distintos componentes y sus conexiones:




Potenciometro
VR1
10K
R5
3K
R4
3K
10nF
C10
1µF
+Vcc
-Vcc
CN3
hembra
CN1
R3
1K
C1
10nF
C3
10nF
TL082
IC2
TL082
IC1
R1
10
R2
1.6M
Masa
CN2
C7
1.5nF
R7
47K
C6
C4
10nF
7.5K
Salida DAC
CN
TL082
IC3
C5
1nF
R6
1.6M
Masa
CN12
LM386
IC4
Jack hembra
CN21
C8 470µF
50nF
R9
10


Potenciometro
VR1
10K
R4
3K
10nF
Entrada ADC
CN10
C10
1µF
C11
1µF
C12
1µF
+Vcc
CN8
+Vcc
CN3
hembra
CN1
R3
1K
C1
10nF
C3
10nF
C15
1µF
C13
1µF
C14
1µF
TL082
IC2
R1
10
TL082
IC1
R2
1.6M
C7
1.5nF
Masa
CN2
R7
47K
C6
1nF
C4
10nF
7.5K
Potenciometro
VR2
10K
Jack hembra
CN21
LM386
IC4
Salida DAC
CN11
C8 470µF
50nF
TL082
IC3
R6
1.6M
C5
1nF
R9
10
Masa
CN12



-Especificaciones:
La PCB tiene un tamaño de (98 x 97) [mm]

El módulo del adaptador de impedancia y el filtro de eliminación de la componente continua está formado por 2 resistencias (R1 y R2) y 1 condensador (C1). La frecuencia de corte del filtro paso alto es 10 Hz.

El módulo del amplificador variable está compuesto por 1 resistencia (R3), 1 potenciómetro (VR1) y un amplificador TL082 de 8 pines.

El módulo del filtro paso bajo antisolapamiento Sallen-Key está compuesto por 2 resistencias del mismo valor (R4 y R5), 2 condensadores del mismo valor (C2 y C3) y un amplificador TL082 de 8 pines. La frecuencia de corte del filtro paso bajo es 3400 Hz.

El módulo del filtro de eliminación de la componente continua está formado por 1 resistencia (R6) y 1 condensador (C4). La frecuencia de corte del filtro paso alto es 10 Hz.

El módulo del filtro paso bajo de reconstrucción Sallen-Key está compuesto por 2 resistencias (R7 y R8), 2 condensadores del mismo valor (C5 y C6) y un amplificador TL082 de 8 pines. La frecuencia de corte del filtro paso bajo es 3400 Hz.

El módulo de la Etapa de Salida está compuesto por 1 condensador (C7), 1 potenciómetro (VR2), 1 resistencia (R9), un amplificador LM386 y 2 condensadores (C8 y C9).

Un circuito de control, formado por tres pares de condensadores de desacoplo (100 µF, 100nF, 100 pF).

Una pista que estará conectada a Tierra.

Una pista que estará conectada a +Vcc (Alimentación de los amplificadores).

Una pista que estará conectada a –Vcc (Alimentación de los amplificadores).

Los valores de los componentes están detallados en la parte analógica de esta memoria (Págs. 5 – 12).

-Problemas:
Los componentes se han puesto de esa forma ya que creemos que es la más adecuada y la más cómoda para la interconexión entre ellos y que no haya pistas que se crucen, aunque al incluir el circuito de control (condensadores de desacoplo) ha sido imposible evitar un cruce.

Debido a que el servicio que había en la escuela para fabricar circuitos impresos ha sido suprimido, el resultado que presentamos ha sido producto exclusivamente nuestro, con los medios que teníamos a nuestro alcance. Se puede ver que el resultado no es el mejor que podría haberse obtenido, pero se ha hecho lo mejor que se ha podido.