



Circuitos Electrónicos (CELT)

Curso 2016-2017

Memoria final

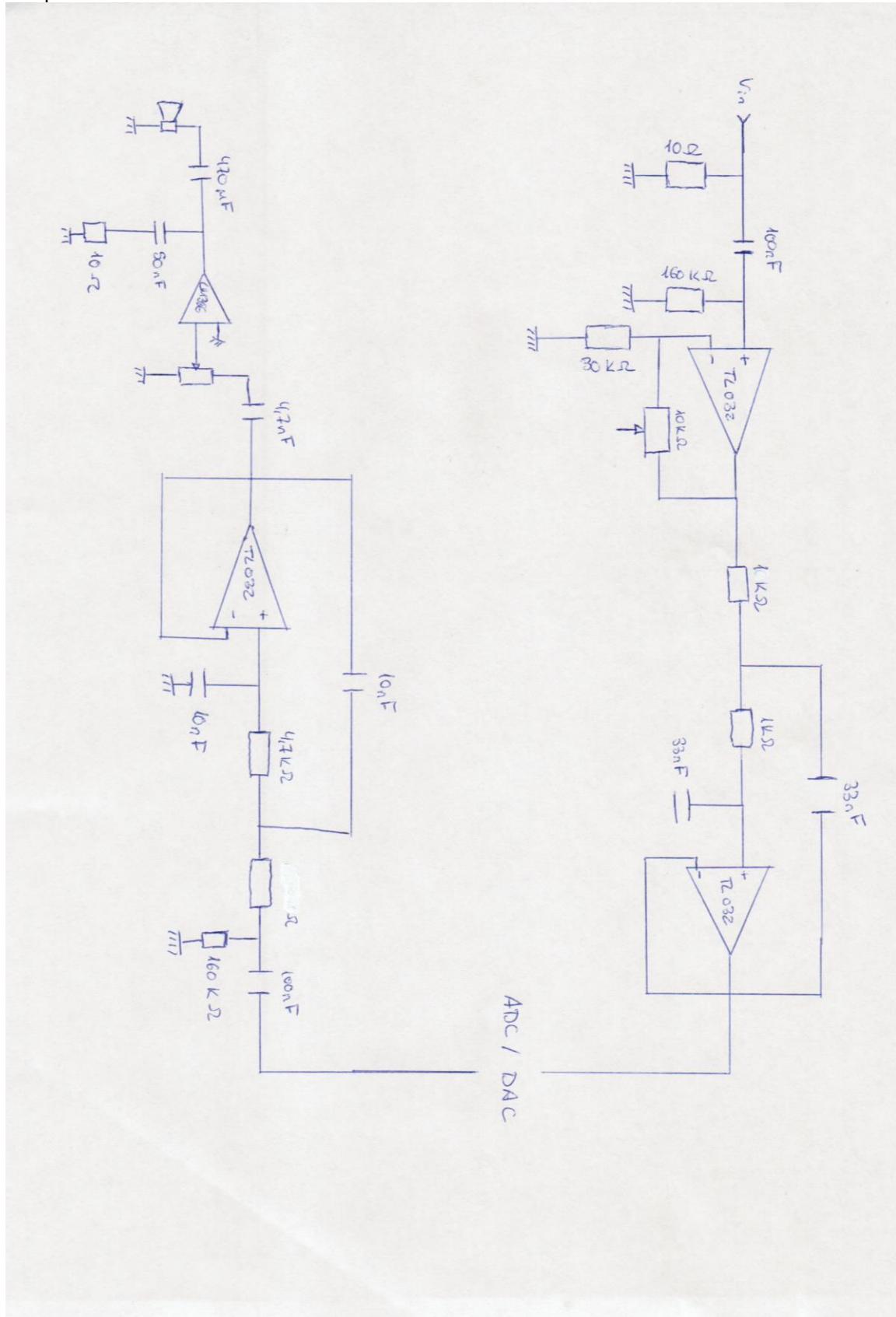
	Apellidos	Nombre
Alumno 1	Cepeda Zamorano	Álvaro
Alumno 2	García Sáez	Pablo

Código de pareja	VC-06
------------------	-------

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

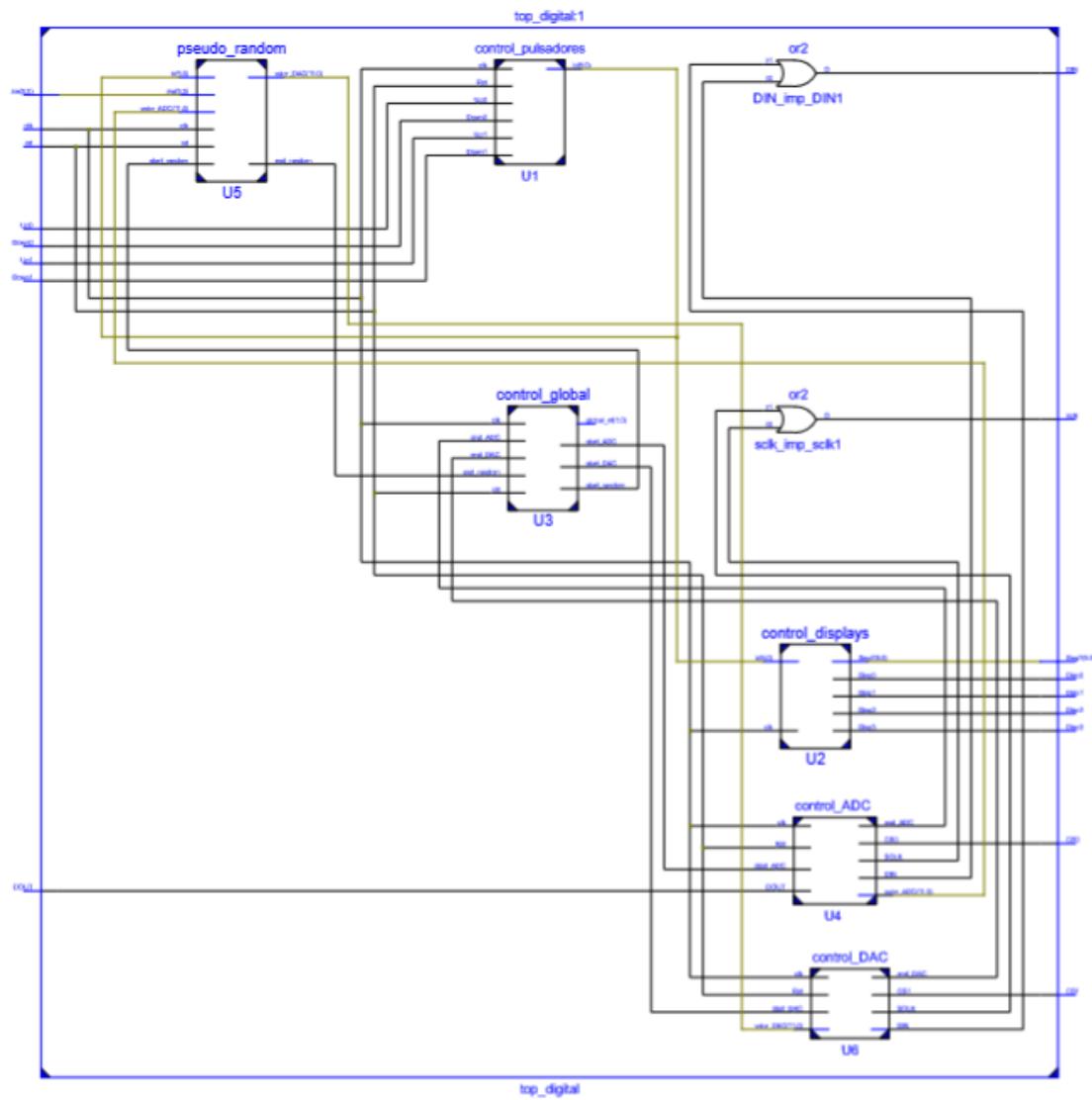
1. ESQUEMA COMPLETO DEL CIRCUITO ANALÓGICO

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



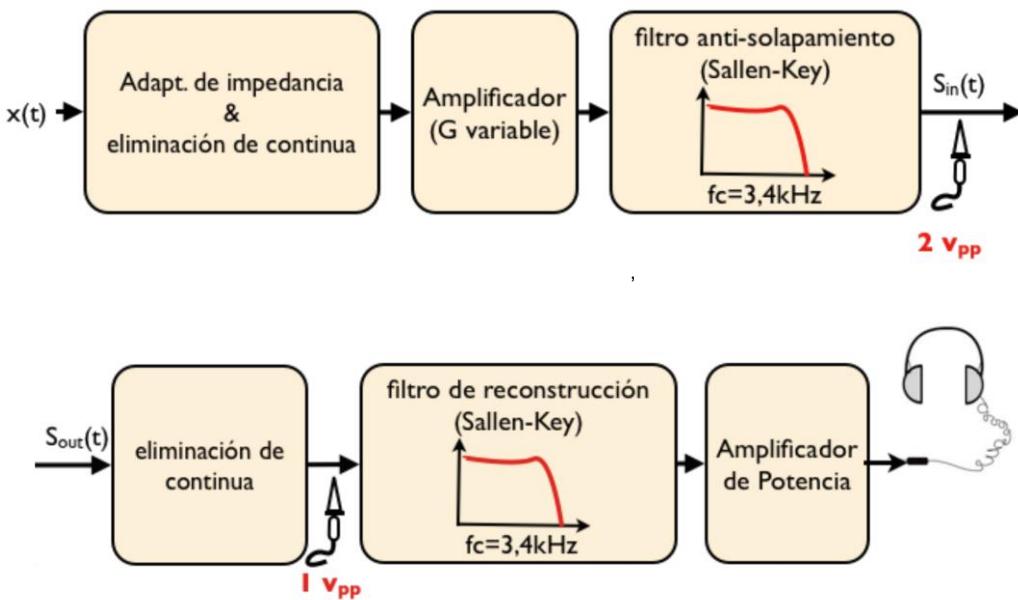
2. ESQUEMA COMPLETO DEL CIRCUITO DIGITAL

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



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) Se escoge una resistencia $R_o = 10 \Omega$ para simular la impedancia de entrada de los auriculares.

$$b) H_{PH}(s) = \frac{s}{s + \omega_c}, \quad 2\pi f_c = \frac{1}{R_o C_1}, \quad f_c = 10 \text{ Hz}$$

$$C_1 = 100 \text{ nF}$$

$$R_o = 160 \text{ k}\Omega$$

$$c) \left| \frac{V_o}{V_i} \right| = \frac{\omega R C}{\sqrt{1 + (\omega R C)^2}}$$

$$\left| \frac{V_o}{V_i} \right| \Big|_{f=5 \text{ Hz}} = 0,4491$$

$$\left| \frac{V_o}{V_i} \right| \Big|_{f=10 \text{ Hz}} = 0,7089$$

$$\left| \frac{V_o}{V_i} \right| \Big|_{f=100 \text{ Hz}} = 0,9950$$

d) Estos valores teóricos se aproximan a nuestros valores medidos

$$0,44, 0,69 \text{ y } 0,97$$

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) La función de transferencia del amplificador variable es:

$$A = \left(1 + \frac{R_2}{R_1} \right) \frac{1}{1 + (1 + R_2/R_1)/a} \approx 1 + \frac{R_2}{R_1}$$

El voltaje a la salida del amplificador no debe superar los 2 Vpp, por lo que tomando para R_2 un potenciómetro de 10 kΩ y un voltaje de 1,5 Vpp obtenemos:

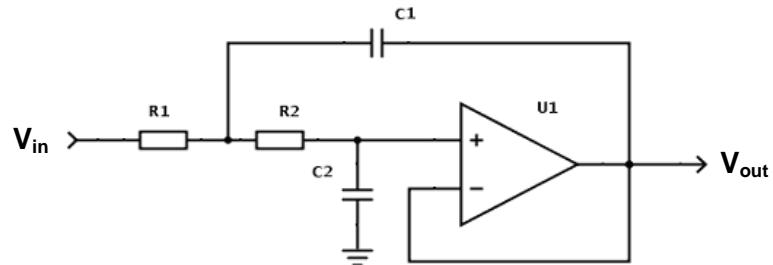
$$\frac{2}{1,5} = 1 + \frac{R_2}{R_1} \rightarrow R_1 = 30 \text{ k}\Omega$$

b) La ganancia es $A = 1 + \frac{R_2}{R_1}$

El valor mínimo se conseguirá poniendo $R_2 = 0$ y el máximo $R_2 = 10 \text{ k}\Omega$

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.



$$a) H_{LP} = \left| \frac{\omega_0^2}{-\omega_c^2 + j \frac{\omega_c}{Q} \cdot \omega_0 + \omega_0^2} \right|$$

$$f_c = 3400 \text{ Hz} \quad C_1 = C_2 \quad L_1 = L_2 \quad Q = 0,5$$

$$H_{LP} = \frac{1}{\sqrt{2}} \rightarrow \frac{\omega_0^2}{\sqrt{(-4,56 \cdot 10^8 + \omega_0^2)^2 + 1,8254 \cdot 10^9 \omega_0^2}} = \frac{1}{\sqrt{2}}$$

$$\frac{\omega_0^2}{\sqrt{\omega_0^4 + 2,079 \cdot 10^{17} - 9,12 \cdot 10^8 \omega_0^2 + 1,8254 \cdot 10^9 \omega_0^2}} = \frac{1}{\sqrt{2}}$$

$$\omega_0 = 33197,12 \frac{\text{rad}}{\text{s}} \rightarrow \omega_0 = \frac{1}{RC} \quad C = 33 \text{nF}$$

$$b) R = 1 \text{k}\Omega$$

$$\left| \frac{V_o}{V_i} \right| = \left| \frac{\omega_0^2}{-\omega_1^2 + j \frac{\omega_1}{Q} \cdot \omega_0 + \omega_0^2} \right|$$

$$\left| \frac{V_o}{V_i} \right|_{f_1 = 1000 \text{ Hz}} = 0,965$$

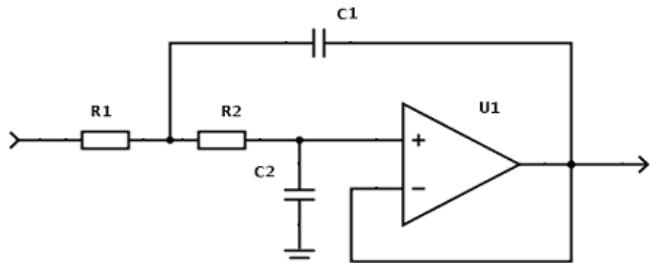
$$\left| \frac{V_o}{V_i} \right|_{f_1 = 3400 \text{ Hz}} = 0,707$$

$$\left| \frac{V_o}{V_i} \right|_{f_1 = 10 \text{ kHz}} = 0,218$$

c) Estos valores teóricos se aproximan a los valores medidas; 0,94, 0,75 y 0,18.

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.



$$a) H_{LP} = \frac{\omega_0^2}{-\omega_c^2 + j \frac{\omega_c}{Q} \omega_0 + \omega_0^2}$$

$f_c = 3400 \text{ Hz}$
 $Q = 0,35$

$$\left| \frac{\omega_0^2}{-\omega_c^2 + j \frac{6800\pi}{0,35} \omega_0 + \omega_0^2} \right| = \frac{1}{\sqrt{2}}$$

$$\frac{\omega_0^2}{\sqrt{\omega_0^4 + 2,07 \cdot 10^{17} + 2,8 \cdot 10^9 \omega_0^2}} = \frac{1}{\sqrt{2}} \rightarrow \omega_0 = 53591,7 \text{ rad/s}$$

$$C_1 = C_2 \quad R_1 = m R_2 \quad Q = \frac{\sqrt{m}}{m+1} = 0,35 \rightarrow m = \frac{0,166}{6}$$

$$\omega_0 = \frac{1}{\sqrt{m} \cdot R \cdot C}$$

Estableciendo los condensadores iguales y de valor
10 nF

$$R_2 = \frac{1}{\omega_0 \sqrt{m} \cdot C} \approx 780 \Omega$$

$$R_1 = m R_2 = 4,7 \text{ k}\Omega$$

b)

$$\left| \frac{V_2}{V_1} \right| = \frac{\omega_0^2}{-\omega_1^2 + j \frac{\omega_1}{Q} \omega_0 + \omega_0^2}$$

$$\left| \frac{V_2}{V_1} \right|_{f=1000 \text{ Hz}} = 0,96$$

$$\left| \frac{V_2}{V_1} \right|_{f=3400 \text{ Hz}} = 0,705$$

$$\left| \frac{V_2}{V_1} \right|_{f=10 \text{ kHz}} = 0,297$$

c) Estos valores teóricos se aproximan con gran exactitud a los valores medidos, siendo estos 0,96 ; 0,75 y 0,29

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)

$$R = 10 \text{ k}\Omega$$

$$f_c = 3,4 \text{ kHz} = \frac{1}{2\pi RC} \rightarrow C = 4,7 \text{ nF}$$

b) Siguiendo las especificaciones del proyecto el valor máximo de entrada es de 1Vpp

c) Debido a la atenuación de la ganancia obtenida en los diagramas de Bode

d) La amplificación máxima de esta etapa es dos veces la entrada, lo cual comprobamos que se cumple.

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  
    begin  
        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;  
        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  
          "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

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity control_pulsadores is
5     Port ( clk : in STD_LOGIC;                                --reloj FPGA
6             Rst : in STD_LOGIC;                                --reset del registro
7             Up0 : in STD_LOGIC;                                --Incremento de 1
8             Down0 : in STD_LOGIC;                             --Decremento de 1
9             Up1 : in STD_LOGIC;                                --Incremento de 10
10            Down1 : in STD_LOGIC;                            --Decremento de 10
11            b : out STD_LOGIC_VECTOR (5 downto 0);          --Valor del código metido por los pulsadores en binario
12 end control_pulsadores;
13
14 architecture Behavioral of control_pulsadores is
15
16 --Declaración de los componentes
17 Component contador_100ms is
18     Port(clk,rst_cnt,en_cnt : in STD_LOGIC;
19           cnt_100ms : out STD_LOGIC);
20 End component;
21
22 Component Reg_b is
23     Port(clk,rst_b,en_b : in STD_LOGIC;
24           sel : in STD_LOGIC_VECTOR (1 downto 0);
25           b : out STD_LOGIC_VECTOR (5 downto 0));
26 End component;
27
28 Component MooreFSM is
29     Port(clk,Up0,Down0,Up1,Down1,Rst,cnt_100ms : in STD_LOGIC;
30           rst_cnt,en_cnt,rst_b,en_b : out STD_LOGIC;
31           sel : out STD_LOGIC_VECTOR (1 downto 0));
32 End component;
33
34 --Declaración de señales
35 signal rst_100ms, en_100ms, cnt_100ms, rst_moore, en_moore : STD_LOGIC;
36 signal sel_moore : STD_LOGIC_VECTOR (1 downto 0);
37
38 -- Implementación de la arquitectura
39 begin
40 U1 : contador_100ms port map (clk,rst_100ms,en_100ms,cnt_100ms);
41 U2 : MooreFSM port map(clk,Up0,Down0,Up1,Down1,Rst,cnt_100ms,rst_100ms,en_100ms,rst_moore,en_moore,sel_moore);
42 U3 : Reg_b port map(clk,rst_moore,en_moore,sel_moore,b);
43
44 end Behavioral;

```

Contador 100ms

```

1 -----
2 --CONTADOR 100MS
3
4 --Contador que genera una señal que se activa con cada cuenta de 100 ms.
5 -----
6 library IEEE;
7 use IEEE.STD_LOGIC_1164.ALL;
8 use IEEE.STD_LOGIC_ARITH.ALL;
9 use IEEE.STD_LOGIC_UNSIGNED.ALL;
10
11 entity contador_100ms is
12     Port ( clk : in STD_LOGIC; --Reloj FPGA
13             rst_cnt : in STD_LOGIC; --reset del contador
14             en_cnt : in STD_LOGIC; -- enable del contador
15             cnt_100ms : out STD_LOGIC); --fin de la cuenta de 100ms
16 end contador_100ms;
17
18 architecture Behavioral of contador_100ms is
19 signal contador : STD_LOGIC_VECTOR(22 downto 0):= (others => '0'); --Declaracion de la señal con 23 bits
20
21 begin
22     process (rst_cnt, clk)
23     begin
24         if rst_cnt='1' then --Reset asíncrono
25             cnt_100ms <= '0';--Fin de cuenta a 0
26             contador <= (others => '0');--Inicializa el contador
27         elsif clk'event and clk='1' then
28             cnt_100ms <= '0';--Fin de cuenta a 0
29             if en_cnt='1' then--Si el enable es uno empieza a contar
30                 contador <= contador + '1';--Suma de un ciclo
31             end if;
32             if contador >= 5000000 then -- valor máximo de la cuenta = 0.1 (20ns)
33                 cnt_100ms <= '1';--Fin de cuenta(Ha llegado al máximo)
34                 contador <= (others => '0');--Contador de ciclos a 0
35             end if;
36         end if;
37     end process;
38 end Behavioral;

```

El valor AAAA cuenta el numero de periodos que tienen que pasar de clk para que se llegue a 100 ms. Este valor se obtiene gracias a la frecuencia del reloj de la FPGA.

El valor BBBB es el número de bits necesarios que tiene que tener la señal contador para poder representar el valor AAAA en binario.

MooreFSM

```

1  --AUTÓMATA DE MOORE
2
3  --Máquina de estados (Finite State Machine, FSM) de Moore que controla el
4  --funcionamiento del módulo. Recibe las señales de los pulsadores (Up0, Down0, Up1 y Down1) y el
5  --interruptor (Rst), gestiona la cuenta de 100 ms. (mediante las señales rst_cnt, en_cnt, y cnt_100ms), y
6  --controla el valor del registro Reg_b (mediante las señales rst_b, en_b y sel).
7  -----
8  library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
10 use IEEE.STD_LOGIC_ARITH.ALL;
11 use IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13 entity MooreFSM is
14     Port ( clk : in STD_LOGIC; --Reloj FPGA
15
16         Up0 : in STD_LOGIC; -- incremento1 del valor (pulsador 0)
17         Down0 : in STD_LOGIC; -- decremento1 del valor (pulsador 1)
18         Up1 : in STD_LOGIC; -- incremento10 del valor (pulsador 2)
19         Down1 : in STD_LOGIC; -- decremento10 del valor (pulsador 3)
20         Rst : in STD_LOGIC; -- reset del registro (Sw 7)
21         cnt_100ms : in STD_LOGIC; -- Fin de la cuenta de 100ms
22         rst_cnt : out STD_LOGIC; -- Inicializa contador 100ms
23         en_cnt : out STD_LOGIC; -- Habilita contador 100ms
24         rst_b : out STD_LOGIC; -- Inicializa registro de frecuencia
25         en_b : out STD_LOGIC; -- Enable de incremento/decremento
26         sel : out STD_LOGIC_VECTOR (1 downto 0); -- Incremento/Decremento del registro
27     end MooreFSM;
28
29 architecture Behavioral of MooreFSM is
30
31 type state_type is (st_Reset, st_Main, st_Up0a,st_Up0b,st_Down0a, st_Down0b, st_Upla,st_Uplb,st_Downia,st_Down1b); --Estados
32 signal state : state_type; --Señal que lleva el estado actual del sistema
33
34 begin -- Implementación de la arquitectura
35
36     process (clk)
37     begin
38         if clk'event and clk='1' then
39             rst_b <= '0'; en_b <= '0'; sel(0)<= '0'; -- valores por defecto
40             rst_cnt <= '1'; en_cnt <= '0';
41             CASE state IS
42                 when st_Reset => --Estado de Reset
43                     rst_b <= '1'; state <= st_Main;
44                 when st_Main => --Estado principal
45                     rst_b <= '0'; en_b <= '0'; sel<= "00"; rst_cnt <= '1'; en_cnt <= '0'; -- valores por defecto
46                     if (Rst='1') then --Reset
47                         state<= st_Reset;
48                     end if;
49                     if (Up0='1')then
50                         state <= st_Up0a; --Estado en el que espera al contador para sumar 1
51                     end if;
52                     if (Down0='1') then
53                         state<= st_Down0a; --Estado en el que espera al contador para restar 1
54                     end if;
55                     if (Up1='1')then
56                         state <= st_Upla; --Estado en el que espera al contador para sumar 10

```

```

57         end if;
58         if (Down1='1')then
59             state <= st_Down1a; --Estado en el que espera al contador para restar 10
60         end if;
61
62         when st_Up0a =>
63             rst_cnt<= '0'; en_cnt<='1';
64             if (cnt_100ms ='0') then
65                 state<=st_Up0a;
66             else state <=st_Up0b; --Si pasan 100 ms pasa al estado de suma unidad
67             end if;
68
69         when st_Up0b =>
70             en_b<='1'; sel<="00"; --Activa la suma unidad
71             state<= st_Main;
72
73
74         when st_Down0a =>
75             rst_cnt<= '0'; en_cnt<='1';
76             if (cnt_100ms ='0') then
77                 state<=st_Down0a;
78             else state <=st_Down0b; --Si pasan 100 ms pasa al estado de resta unidad
79             end if;
80
81         when st_Down0b =>
82             en_b<='1'; sel<="01"; --Activa la resta unidad
83             state<= st_Main;
84
85         when st_Upla =>
86             rst_cnt<= '0'; en_cnt<='1';
87             if (cnt_100ms ='0') then
88                 state<=st_Upla;
89             else state <=st_Uplb; --Si pasan 100 ms pasa al estado de sumar 10
90             end if;
91
92         when st_Uplb =>
93             en_b<='1'; sel<="10"; --Activa la suma de 10
94             state<= st_Main;
95
96
97         when st_Down1a =>
98             rst_cnt<= '0'; en_cnt<='1';
99             if (cnt_100ms ='0') then
100                 state<=st_Down1a;
101             else state <=st_Down1b; ----Si pasan 100 ms pasa al estado de resta 10
102             end if;
103
104         when st_Down1b =>
105             en_b<='1'; sel<="11"; --Activa la resta de 10
106             state<= st_Main;
107
108
109         when others => state<= st_Main;-- Cualquier otro caso al estado inicial
110     END CASE;
111     end if;
112 end process;

```

Reg_b

```

1 -----
2 --REG_b
3 --Este registro contiene en binario el valor del código introducido por el usuario, b. En su
4 --funcionalidad se incorpora también que el registro sea capaz de variar su valor (en función de lo que
5 --indiquen las señales que provienen de la máquina de estados de Moore), y que no se excedan los
6 --límites máximos y mínimos permitidos (0-63)
7 -----
8 library IEEE;
9 use IEEE.STD_LOGIC_1164.ALL;
10 use IEEE.STD_LOGIC_UNSIGNED.ALL;
11 use IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13 entity Reg_b is
14     Port ( clk : in STD_LOGIC; --Reloj
15             rst_b : in STD_LOGIC;-- reset de B
16             en_b : in STD_LOGIC;-- enable de B
17             sel : in STD_LOGIC_VECTOR (1 downto 0); -- Incremento/Decremento del registro
18             b : out STD_LOGIC_VECTOR (5 downto 0)); --Valor del código de usuario
19 end Reg_b;
20
21 architecture Behavioral of Reg_b is
22 signal s_Reg_b : STD_LOGIC_VECTOR(5 downto 0);
23
24 begin
25     process (rst_b, clk)
26     begin
27         if (rst_b = '1') then
28             s_Reg_b <= "000000"; -- Asignación del valor por defecto
29         elsif clk'event and clk='1' then
30             if en_b = '1' then
31                 if ((sel = "00") and (s_Reg_b < "111111")) then
32                     s_Reg_b <= s_Reg_b + "01"; -- Incremento +1 de la señal
33                 end if;
34                 if ((sel = "01") and (s_Reg_b > "000000")) then
35                     s_Reg_b <= s_Reg_b - "01"; -- Decremento -1 de la señal
36                 end if;
37                 if ((sel = "10") and (s_Reg_b < "110110")) then
38                     s_Reg_b <= s_Reg_b + "01010"; -- Incremento +10 de la señal
39                 end if;
40                 if ((sel = "11") and (s_Reg_b > "001001")) then
41                     s_Reg_b <= s_Reg_b - "01010"; -- Decremento -10 de la señal
42                 end if;
43             end if;
44         end if;
45     end process;
46
47     b<=s_Reg_b;--Señal final
48 end Behavioral;

```

CCCC inicializa el valor de s_Reg_b a 0.

Los valores DDDD, EEEE... representan los valores límites que puede tener la señal s_Reg_b por los cuales no se podrá sumar o restar más a partir de esos valores. Siendo:

DDDD = 63

EEEE = 0

FFFF = 54

GGGG = 9

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

```

1 -----+
2 --CONTROL DE DISPLAYS
3 --El bloque de control de los displays se encarga de mostrar el valor de la frecuencia de la señal
4 --deseada.
5 -----
6 library IEEE;
7 use IEEE.STD_LOGIC_1164.ALL;
8
9 entity control_displays is
10    Port ( clk : in STD_LOGIC;--reloj
11           b : in STD_LOGIC_VECTOR (5 downto 0);--señal
12           sw : in STD_LOGIC_VECTOR (5 downto 0);
13           Disp0 : out STD_LOGIC;--Display0
14           Disp1 : out STD_LOGIC;--Display1
15           Disp2 : out STD_LOGIC;--Display2
16           Disp3 : out STD_LOGIC;--Display3
17           Seg7 : out STD_LOGIC_VECTOR (6 downto 0)); --conjunto de LEDs que se activan en el display seleccionado
18 end control_displays;
19
20 architecture Behavioral of control_displays is
21
22 --Declaración de los componentes
23 component contador_5ms is
24    Port(
25       clk : in STD_LOGIC; --reloj
26       cnt_5ms: out STD_LOGIC); -- salida: activa cada 5ms
27   end component;
28
29 component conversorBCD is
30    Port ( clk: in STD_LOGIC; --reloj
31           b : in STD_LOGIC_VECTOR (5 downto 0);
32           sw : in STD_LOGIC_VECTOR (5 downto 0);
33           D0 : out STD_LOGIC_VECTOR (3 downto 0); --Digito0
34           D1 : out STD_LOGIC_VECTOR (3 downto 0); --Digito1
35           D2 : out STD_LOGIC_VECTOR (3 downto 0); --Digito2
36           D3 : out STD_LOGIC_VECTOR (3 downto 0));--Digito3
37   end component;
38
39 component Visualizacion is
40    Port (
41       clk : in STD_LOGIC; --reloj
42       cnt_5ms : in STD_LOGIC; --se activa durante un flanko de reloj cuando la cuenta llega a 5 ms
43       Digito0, Digito1, Digito2, Digito3 : in STD_LOGIC_VECTOR (3 downto 0); -- valores BCD a presentar
44       Disp0, Disp1, Disp2, Disp3 : out STD_LOGIC; -- salidas encargadas de habilitar los displays
45       Seg7 : out STD_LOGIC_VECTOR (6 downto 0)); -- conjunto de LEDs que se activan en el display seleccionado
46   end component;
47
48 --Declaración de las señales
49
50 signal Dig1, Dig2, Dig3, Dig0 : STD_LOGIC_VECTOR(3 downto 0);
51 signal cnt : STD_LOGIC;
52
53 --Implementación de la arquitectura
54 begin
55
56 U1: contador_5ms port map (clk, cnt);
57 U2: conversorBCD port map (clk,b, sw, Dig0,Dig1,Dig2,Dig3);
58 U3: visualizacion port map ( clk,cnt,Dig0,Dig1,Dig2,Dig3,Disp0,Disp1,Disp2,Disp3,Seg7);
59
60 end Behavioral;
61
62

```

Contador 5ms

```
1 -----  
2 -- CONTADOR 5ms  
3 -- Puesto que el módulo de visualización de los displays necesita multiplexar los  
4 --datos de los 4 dígitos para utilizar los 4 displays a la vez, hace falta una señal que indique cuándo hay  
5 --que mostrar el siguiente dato. Para que la visualización sea correcta se ha seleccionado T = 5ms., de  
6 --forma que cada display se refresque cada 20 ms.  
7 -----  
8 library IEEE;  
9 use IEEE.STD_LOGIC_1164.ALL;  
10 use IEEE.STD_LOGIC_UNSIGNED.ALL;  
11 use IEEE.STD_LOGIC_ARITH.ALL;  
12  
13 entity contador_5ms is  
14     Port(  
15         clk : in STD_LOGIC; -- entrada: reloj  
16         cnt_5ms: out STD_LOGIC); -- salida: activa cada 5ms  
17 end contador_5ms;  
18  
19 architecture arch_contador_5ms of contador_5ms is  
20     signal contador : STD_LOGIC_VECTOR(17 downto 0) := "00000000000000000000"; --Señal que guarda las cuentas|  
21 begin  
22     process(clk)  
23     begin  
24         if clk'event and clk='1' then  
25             cnt_5ms <= '0';  
26             contador <= contador + '1';  
27             if contador > 249999 then  
28                 cnt_5ms <= '1';  
29                 contador <= (others => '0');  
30             end if;  
31         end if;  
32     end process;  
33 end arch_contador_5ms;
```

Conversor BCD

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity conversorBCD is
7      Port ( clk : in STD_LOGIC;
8             b : in STD_LOGIC_VECTOR (5 downto 0);
9             sw : in STD_LOGIC_VECTOR (5 downto 0);
10            D0 : out STD_LOGIC_VECTOR (3 downto 0);
11            D1 : out STD_LOGIC_VECTOR (3 downto 0);
12            D2 : out STD_LOGIC_VECTOR (3 downto 0);
13            D3 : out STD_LOGIC_VECTOR (3 downto 0));
14 end conversorBCD;
15
16 architecture Behavioral of conversorBCD is
17
18
19 begin
20
21 process (b)
22     VARIABLE resto : STD_LOGIC_VECTOR (5 downto 0);
23     VARIABLE resto_sw : STD_LOGIC_VECTOR (5 downto 0);
24
25 begin
26
27     D0 <= (others => '0');      resto := b;      --VALORES POR DEFECTO
28     D1 <= (others => '0');      resto := b;
29     D2 <= (others => '0');      resto_sw := sw;
30     D3 <= (others => '0');      resto_sw := sw;
31
32
33     if b >= conv_std_logic_vector(10, 6) then
34         D1 <= "0001";
35         resto := b -conv_std_logic_vector(10, 6);    --resto = data_in -10
36     end if;
37
38     if b >= conv_std_logic_vector(20, 6) then
39         D1 <= "0010";
40         resto := b -conv_std_logic_vector(20, 6);    --resto = data_in -20
41     end if;
42
43     if b >= conv_std_logic_vector(30, 6) then
44         D1 <= "0011";
45         resto := b -conv_std_logic_vector(30, 6);    --resto = data_in -30
46     end if;
47
48     if b >= conv_std_logic_vector(40, 6) then
49         D1 <= "0100";
50         resto := b -conv_std_logic_vector(40, 6);    --resto = data_in -40
51     end if;
52
53     if b >= conv_std_logic_vector(50, 6) then
54         D1 <= "0101";
55         resto := b -conv_std_logic_vector(50, 6);    --resto = data_in -50
56     end if;

```

```
57      if b >= conv_std_logic_vector(60, 6) then
58          D1 <= "0110";
59          resto := b -conv_std_logic_vector(60, 6);    --resto = data_in -60
60      end if;
61
62      D0 <= resto(3 downto 0);    --Asignacion de salida
63
64      --Ahora para los interruptores
65
66      if sw >= conv_std_logic_vector(10, 6) then
67          D3 <= "0001";
68          resto_sw := sw -conv_std_logic_vector(10, 6);    --resto_sw = data_in -10
69      end if;
70
71      if sw >= conv_std_logic_vector(20, 6) then
72          D3 <= "0010";
73          resto_sw := sw -conv_std_logic_vector(20, 6);    --resto_sw = data_in -20
74      end if;
75
76      if sw >= conv_std_logic_vector(30, 6) then
77          D3 <= "0011";
78          resto_sw := sw -conv_std_logic_vector(30, 6);    --resto_sw = data_in -30
79      end if;
80
81      if sw >= conv_std_logic_vector(40, 6) then
82          D3 <= "0100";
83          resto_sw := sw -conv_std_logic_vector(40, 6);    --resto_sw = data_in -40
84      end if;
85
86      if sw >= conv_std_logic_vector(50, 6) then
87          D3 <= "0101";
88          resto_sw := sw -conv_std_logic_vector(50, 6);    --resto_sw = data_in -50
89      end if;
90
91      if sw >= conv_std_logic_vector(60, 6) then
92          D3 <= "0110";
93          resto_sw := sw -conv_std_logic_vector(60, 6);    --resto_sw = data_in -60
94      end if;
95
96      D2 <= resto_sw(3 downto 0);    --Asignacion de salida
97  end process;
98
99  end Behavioral;
```

Visualización

```

1 -----  

2 -- VISUALIZACION  

3 -- Es el modulo encargado de mostrar los datos en los displays. Puesto que la  

4 --señal de datos (Seg7) es compartida, es necesario activar cada 5 ms, cada uno de los displays y  

5 --poner los datos correspondientes en el bus Seg7.  

6 -----  

7 library IEEE;  

8 use IEEE.STD_LOGIC_1164.ALL;  

9  

10 use IEEE.STD_LOGIC_UNSIGNED.ALL;  

11  

12 entity Visualizacion is  

13   Port (  

14     clk : in STD_LOGIC;      --reloj  

15     cnt_5ms : in STD_LOGIC; --se activa durante un flanko de reloj cuando la cuenta llega a 5 ms  

16     Dígito0, Dígito1,Dígito2,Dígito3 : in STD_LOGIC_VECTOR (3 downto 0); --valores BCD a presentar  

17     Disp0, Disp1, Disp2, Disp3 : out STD_LOGIC; --salidas encargadas de habilitar los displays  

18     Seg7 : out STD_LOGIC_VECTOR (6 downto 0)); --conjunto de LEDs que se activan en el display seleccionado  

19  

20 end Visualizacion;  

21  

22  

23 architecture arch_Visualizacion of Visualizacion is  

24 signal sel : STD_LOGIC_VECTOR (1 downto 0) := "00";-- señal para activar display  

25 signal DígitoBCD : STD_LOGIC_VECTOR (3 downto 0);-- señal almacenar dígito  

26 signal Disp2_aux, Disp3_aux :STD_LOGIC;  

27  

28 begin  

29  

30   process ( clk, cnt_5ms, sel, Dígito0, Dígito1, Dígito2, Dígito3)  --PROCESO PARA ACTIVAR LOS DISPLAYS  

31 begin  

32   DígitoBCD <= Dígito0;      -- valores por defecto  

33   Disp2_aux <= '1'; Disp3_aux <= '1'; Disp0 <= '1'; Disp1 <= '1'; Disp2 <= '1'; Disp3 <= '1';  

34  

35   if clk'event and clk='1' then  

36     if (cnt_5ms = '1') then  

37  

38       CASE sel is  

39         when "00" =>  

40           sel<="01";      -- activavamos el display0 y dígitoBCD=dígito0  

41  

42         when "01" =>  

43           sel<="10";      -- activavamos el display1 y dígitoBCD=dígito1  

44  

45         when "10" =>  

46           sel<="11";      -- activavamos el display2 y dígitoBCD=dígito2  

47  

48         when "11" =>  

49           sel<="00";      -- activavamos el display3 y dígitoBCD=dígito3  

50  

51         when others =>    sel<="00";  

52  

53       end case;  

54       end if;  

55     end if;

```

```

56      case sel is
57        when "00" =>
58          Disp0<='0';
59          -- activamos el display0 y digitoBCD=digito0
60          DigitoBCD <= Dígito0;
61        when "01" =>
62          Disp1<='0';
63          -- activamos el display1 y digitoBCD=digito1
64          DigitoBCD <= Dígito1;
65        when "10" =>
66          Disp2<= '0';
67          -- activamos el display2 y digitoBCD=digito2
68          DigitoBCD <= Dígito2;
69        when "11" =>
70          Disp3 <= '0';
71          -- activamos el display3 y digitoBCD=digito3
72          DigitoBCD <= Dígito3;
73        when others => Disp0 <= '1';
74
75      end case;
76
77
78
79      CASE DigitoBCD is
80        when "0000" => Seg7 <= "00000001";
81        when "0001" => Seg7 <= "10011111";
82        when "0010" => Seg7 <= "00100101";
83        when "0011" => Seg7 <= "00001100";
84        when "0100"=> Seg7 <= "10011000";
85        when "0101"=> Seg7 <= "01000100";
86        when "0110"=> Seg7 <= "01000000";
87        when "0111" => Seg7 <= "00011111";
88        when "1000" => Seg7 <= "00000000";
89        when "1001" => Seg7 <= "00001100";
90        when others => Seg7 <= "11111111";
91      END CASE;
92
93    end process;
94  end arch_Visualizacion;

```

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.

El primer contador es un contador de 1 microsegundo que genera la señal fin de cuenta, haciendo así que el segundo contador altere su valor entre 0 y 1 cada vez que le llega esta señal fin de cuenta, generando así la señal SCLK. Cuando SCLK y fin de cuenta coinciden el tercer contador incremente el número de los bits de petición enviados a través del DIN, llegando este hasta 8. Dicho DIN se encarga de enviar los bits de petición uno a uno al ADC. La señal del DIN se actualiza en los flancos de bajada de SCLK.

CS0 es una señal activa a nivel bajo durante el tiempo que dura el envío del byte de petición y la recepción de los dos bytes de datos.

```

control_ADC.vhd                                         Fri Dec 09 23:37:41 2016
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
7
8
9  entity control_ADC is
10     Port ( clk : in STD_LOGIC;      --Reloj del sistema
11            Rst : in STD_LOGIC;      --Interruptor 7
12            start_ADC : in STD_LOGIC; --Inicio de operación del bloque
13            DOUT : in STD_LOGIC;    --señal serie que se recibe desde el ADC
14            end_ADC : out STD_LOGIC; -- Fin de operación del bloque
15            --Señales de control y petición para el ADC
16            CS0 : out STD_LOGIC;
17            SCLK : out STD_LOGIC;
18            DIN : out STD_LOGIC;
19
20            valor_ADC : out STD_LOGIC_VECTOR (11 downto 0));--Último valor binario
21  completo recibido desde el ADC
22 end control_ADC;
23
24 ARCHITECTURE behavior OF control_ADC IS
25
26 type state_type is (st_00,st_01,st_10,st_11);           --Estados
27 signal state : state_type := st_00;                      --Señal de estado
28 signal contador : STD_LOGIC_VECTOR(5 downto 0):= (others => '0'); --Cuenta el
numero de períodos transcurridos
29 signal fin_de_cuenta : STD_LOGIC;                         --Indica que ha
pasado un micro segundo
30 signal en_cnt : STD_LOGIC;                                --Activa la cuenta
31 signal bit_DIN : STD_LOGIC_VECTOR(3 downto 0);           --Señal de
control. Asegura que se trasmite el número correcto de bits por bloque.
32 signal SCLK_aux : STD_LOGIC := '0';                      --Señal auxiliar
del contador selk
33 signal byte_peticion_ADC : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');--Byte de
petición del ADC
34 signal dato_ADC : STD_LOGIC_VECTOR(15 downto 0);          --Salida generada
35
36
37 BEGIN
38
39 process (clk, en_cnt, contador,state) --contador lus
40 begin
41
42     IF clk'event and clk='1' then
43         fin_de_cuenta <= '0';
44         if en_cnt = '1' then           --Si el contador está activo
45             contador <= contador + 1;   --Empieza a contar ciclos
46         end if;
47         if contador >= "110010" then  --Valor máximo de la cuenta(50)
48             fin_de_cuenta <= '1';     --Termina la cuenta(Llega a 50)
49             contador <= (others => '0'); --Contador a 00
50         end if;
51         if rst = '1' then           --Si no llega a 50 ciclos no termina la
52             fin_de_cuenta <= '0';

```

```

control_ADC.vhd                                         Fri Dec 09 23:40:44 2016
cuenta
53         contador <= (others => '0');
54     end if;
55     end if;
56     end process;
57 -- Recibe la señal de fin de cuenta del primer contador y la señal SCLK,
58 -- e incrementa el número de los bits de petición enviados a través de la señal DIN
59 (hasta 8),
60 -- actualizando el valor de dicha señal en cada flanco de bajada de SCLK
61 process (clk)
62 begin
63     if (clk'event and clk = '1') then
64         if (fin_de_cuenta= '1' and sclk_aux = '0') then      --Si es 0 pasa a 1
65             sclk_aux <= '1';
66         elsif (fin_de_cuenta = '1' and sclk_aux = '1') then   --Si es 1 pasa a 0
67             sclk_aux <= '0';
68         end if;
69     end if;
70 end process;
71 sclk <= sclk_aux;
72
73
74 PROCESS (Rst, clk, sclk_aux, fin_de_cuenta, bit_din)
75 BEGIN
76
77     IF (Rst = '1') THEN          --Reset asincrono, Inicializa las señales a su
estado por defecto
78         dato_ADC <=(others => '0');
79         state<=st_00;
80         end_ADC <= '0';
81         en_cnt <= '0';
82         valor_ADC <= "000000000000";
83         bit_DIN <= "0000";
84
85     ELSIF clk'event and clk = '1' THEN
86         CASE state IS
87             WHEN st_00 =>           -- Reposo (Estado st_00)
88                 end_ADC <= '0';    -- Señales a su valor por defecto
89                 byte_peticion_ADC <= "10010111"; -- Precarga del byte de peticion
90
91             IF start_ADC = '1' THEN      --Si tiene permiso para empezar estado el
01 pasa a este estado
92                 state <= st_01;
93             END IF;
94
95             WHEN st_01 =>           -- Entrega de la peticion (Estado st_01)
96                 end_ADC <= '0';
97                 en_cnt <= '1';
98                 IF (fin_de_cuenta = '1' and SCLK_aux = '1') THEN
99                     byte_peticion_ADC <= byte_peticion_ADC(6 downto 0) & '0'; -- Desplaza
petición
100                bit_DIN <= bit_DIN + "01";--Cuenta el numero de desplazamientos
101                IF bit_DIN >= "0111" THEN -- Se ha enviado la peticion completa
102                    state <= st_10;
103                    bit_DIN <= (others => '0');--Señal de control a 0
104                END IF;
105            END IF;

```

```

control_ADC.vhd                                         Fri Dec 09 23:41:09 2016
106      WHEN st_10 => -- Recepcion de los datos serie (Estado st_10)
107          en_cnt <= '1';--Activa la cuenta
108          end_ADC <= '0';
109          IF (fin_de_cuenta = '1' and SCLK_aux = '0') THEN -- Flanco de subida de SCLK
110              dato_ADC <= dato_ADC(14 downto 0) & DOUT; -- Recepcion serie
111          END IF;
112          IF (fin_de_cuenta = '1' and SCLK_aux = '1') THEN -- Flanco de bajada de SCLK
113              bit_DIN <= bit_DIN + "01";--Actualiza la señal de control
114              IF bit_DIN = "1111" THEN -- Se ha recibido el dato completo
115                  state <= st_11; --Pasa al estado final
116                  bit_DIN <= (others => '0');
117              END IF;
118          END IF;
119          WHEN st_11 => -- Entrega del dato de salida, y final (Estado st_11)
120              state <= st_00;--Vuelve al principio
121              en_cnt <= '0';
122              valor_ADC <= dato_ADC(14 downto 3);-- Último valor binario completo
123              recibido desde el ADC
124              end_ADC<= '1'; --Termina el adc
125
126      WHEN others =>
127          state <= st_00;
128          end_ADC <= '0';
129      END CASE;
130  END IF;
131  END PROCESS;
132  DIN <= byte_peticion_ADC(7) when ((state = st_01) or (state = st_10)) else '0';
133  --Genera la señal din final
134  CS0 <= '0' when ((state = st_01) or (state = st_10)) else '1';
END behavior;

```

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

El DAC funciona exactamente igual que el ADC explicado en el apartado anterior, hemos usado la misma secuencia de contadores, por lo tanto SCLK tiene el mismo funcionamiento.

DIN tiene un funcionamiento similar, pero en este caso se encarga de transmitir los bits de datos provenientes del ADC al DAC.

CS1 es similar a CS0, pero activa a nivel alto durante la duración de la transmisión.

```

control_DAC.vhd                                         Fri Dec 09 23:47:08 2016
1  library IEEE;
2  use IEEE.STD.LOGIC_1164.ALL;
3  use IEEE.STD.LOGIC.ARITH.ALL;
4  use IEEE.STD.LOGIC.UNSIGNED.ALL;
5  entity control_DAC is
6      Port ( clk : in STD_LOGIC; --Reloj del sistema
7              Rst : in STD_LOGIC; --Reset del sistema
8              start_DAC : in STD_LOGIC; --Da permiso al DAC de empezar
9              valor_DAC : in STD_LOGIC_VECTOR (11 downto 0); --Datos del encriptador
10             end_DAC : out STD_LOGIC; --Señal de finalizacion del bloque DAC
11             CS1 : out STD_LOGIC; --señales de control del DAC
12             SCLK : out STD_LOGIC;
13             DIN : out STD_LOGIC); --envio de los datos del DAC
14 end control_DAC;
15 architecture Behavioral of Control_DAC is
16 type STATE_TYPE is (st_00,st_01,st_11);
17 --Declaracion de señales auxiliares
18 signal contador : STD_LOGIC_VECTOR(5 downto 0) := "000000"; --Cuenta los ciclos del reloj
19 signal fin_de_cuenta : STD_LOGIC; --Indica cuando el contador llega al maximo
20 signal sclk_aux : STD_LOGIC := '0'; --Señal auxiliar entre contadores
21 signal state : state_type := st_00;--Señal de estado
22 signal bit_DIN : STD_LOGIC_VECTOR(3 DOWNTO 0) := (others=>'0');--Señal de control de
datos transmitidos
23 signal en_cnt : STD_LOGIC; --Activa los contadores
24 signal dato_DAC : STD_LOGIC_VECTOR(15 DOWNTO 0) := (others=>'0'); --Bytes procedentes
de del DAC
25 begin
26 process(clk) --Contador de lus
27 begin
28 if clk'event and clk='1' then
29     fin_de_cuenta <= '0';
30     if (en_cnt='1') then --Si el contador esta activado empiza a sumar
31         contador <= contador + '1';
32     end if;
33     if contador >= "110010" then --Valor maximo de la cuenta(Al llegar pasa un
micro segundo)
34         fin_de_cuenta <= '1';
35         contador <= (others => '0');
36     end if;
37     if rst='1' then
38         fin_de_cuenta <= '0';
39         contador <= (others => '0');
40     end if;
41 end if;
42 end process;
43 --Recibe la señal de fin de cuenta, y cambia su estado generando la señal SCLK.
44 process (clk)
45 begin
46 if clk'event and clk='1' then
47     if( fin_de_cuenta = '1' and sclk_aux ='0')then --Si es 0 pasa a 1
48         sclk_aux<='1';
49     end if;
50     if (fin_de_cuenta = '1' and sclk_aux ='1')then --Si es 1 pasa a 0
51         sclk_aux<='0';
52     end if;
53 end if;
54 end process;
55 sclk<=sclk_aux;--Salida sclk generada

```

```

control_DAC.vhd                                         Fri Dec 09 23:47:29 2016
56  -- Recibe la señal de fin de cuenta del primer contador y la señal SCLK,
57  -- e incrementa el número de los bits de petición enviados a través de la señal DIN
58  -- actualizando el valor de dicha señal en cada flanco de bajada de SCLK
59  PROCESS (Rst, clk)
60  BEGIN
61    IF (Rst = '1') THEN
62      end_DAC<='0';
63      --SCLK <='0';
64      --CS1 <= '1'; --0
65      --DIN <='0';
66      en_cnt <= '0';
67      --fin_de_cuenta <= '0';
68      bit_DIN <= "0000";
69      dato_DAC <= "00000000000000000000";
70
71  ELSIF clk'event and clk = '1' THEN
72  CASE state IS
73    WHEN st_00 =>      --Estado de reposo
74      end_DAC<='0';
75      en_cnt <= '0';
76      dato_DAC <= "000" & valor_DAC & '0'; --Lo que se envia con 0 de control
77      IF (start_DAC = '1') THEN --Si tiene permiso para empezar el bloque empieza
78        state <= st_01;
79      END IF;
80    WHEN st_01=>-- Envío de los datos (Estado st_01)
81      end_DAC<='0';
82      en_cnt <= '1';
83      IF (fin_de_cuenta = '1' and sclk_aux = '1') THEN
84        dato_DAC <= dato_DAC(14 downto 0) & '0';--Lo desplaza para sacar la secuencia
85        bit a bit por el DIN
86        bit_DIN <= bit_DIN + "01";--Actualiza la señal de control
87        IF bit_DIN >= "1111" THEN -- Al llegar a este valor se ha enviado todo
88          state <= st_11;
89          bit_DIN <= (others => '0'); --Señal de control a 0
90        END IF;
91      END IF;
92    WHEN st_11 => -- Entrega del dato de salida, y final (Estado st_11)
93      state <= st_00;
94      end_DAC<='1';--Termina el bloque
95    WHEN others =>
96      end_DAC<='0';
97      state <= st_00;
98  END CASE;
99  END IF;
100 END PROCESS;
101 DIN <= dato_DAC(15) when (state= st_01) else '0'; --Con cada bucle genera un valor
para Din
102 CS1 <= '0' when (state = st_01) else '1';
103 end Behavioral;

```

4.5 Generación del bloque de encriptación

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

```

pesudo_random.vhd                                         Fri Dec 09 23:45:39 2016
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  entity pseudo_random is
6      Port ( clk : in STD_LOGIC;                                --Reloj del sistema
7             Rst : in STD_LOGIC;                                --Reset del sistema
8             start_random : in STD_LOGIC;                      --Inicio del bloque
9             sw : in STD_LOGIC_VECTOR (5 downto 0);           --Código predefinido de
   cifrado en los interruptores
10            b : in STD_LOGIC_VECTOR (5 downto 0);           --Código introducido por
   el usuario a través de los pulsadores
11            valor_ADC : in STD_LOGIC_VECTOR (11 downto 0);  --Último valor binario
   completo recibido del ADC
12            end_random : out STD_LOGIC;                     --Final del bloque
13            valor_DAC : out STD_LOGIC_VECTOR (11 downto 0); --Valor binario a enviar
   al DAC.
14  end pseudo_random;
15 architecture Behavioral of pseudo_random is
16  --Declaración de señales auxiliares
17  signal XOR_A : STD_LOGIC := '0';                         --Señal auxiliar posición 13
   XOR posición 15
18  signal XOR_B : STD_LOGIC := '0';                         --Señal auxiliar XOR_A XOR
   posición 12
19  signal XOR_C : STD_LOGIC := '0';                         --Señal auxiliar XOR_B XOR
   posición 10
20  signal LFSR :STD_LOGIC_VECTOR(15 downto 0) := "0011001100110101";
21 begin
22 process(clk,Rst)
23 begin
24     if clk'event and clk='1' then
25         end_random<='0';
26         if Rst='1' then
27             valor_DAC<= (others=>'0');
28             end_random<='0';
29             LFSR <= "0011001100110101";
30         elsif start_random = '1' then
   aleatorio por el registro de desplazamiento
31             XOR_A <- LFSR(13) XOR LFSR(15);
32             XOR_B<- XOR_A XOR LFSR(12);
33             XOR_C<= XOR_B XOR LFSR(10);
34             LFSR(15 downto 1)<= LFSR(14 downto 0);
35             LFSR(0)<= XOR_C;
   XOR en la ultima posición
36             if (sw = b) then
   valor de los switches con el valor de los pulsadores
37                 valor_DAC<= valor_ADC + "100000000000";
   al carácter bipolar de la salida
38                 end_random<='1';
39             else
39                 valor_DAC<= LFSR(15 downto 4);
   el DAC.
40                 end_random<='1';
41             end if;
42         end if;
43     end if;
44 end process;
45 end Behavioral;

```

--Secuencia aleatoria de números
--Generación del código

--Desplaza los 14 últimos bits
--Pone el bit obtenido por las

--Comprueba si concide el
--Valor del ADC mas 800 debido

--Fin del bloque

--Valor aleatorio generado a

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

El pseudo random genera un código aleatorio gracias un registro de desplazamiento. Comprueba si la contraseña metida por los pulsadores coincide con la de los interruptores y si coinciden pasa el valor del ADC al DAC. Sino pasa el código aleatorio.

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.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4
5 entity top_digital is
6   Port ( clk : in STD_LOGIC;--Reloj del sistema
7         rst : in STD_LOGIC;--Reset del sistema
8         Up0 : in STD_LOGIC;-- Pulsador 0
9         Down0 : in STD_LOGIC;--Pulsador1
10        Up1 : in STD_LOGIC;--Pulsador2
11        Down1 : in STD_LOGIC;--Pulsador3
12        sw : in STD_LOGIC_VECTOR (5 downto 0);--Switches
13        DOUT : in STD_LOGIC;
14        --Display
15        Disp0 : out STD_LOGIC;
16        Disp1 : out STD_LOGIC;
17        Disp2 : out STD_LOGIC;
18        Disp3 : out STD_LOGIC;
19
20        sclk: out STD_LOGIC;      --Señal sclk
21        CS0, CS1: out STD_LOGIC; --Señales CS
22
23        DIN: out STD_LOGIC;     --Señal DIN
24        Seg7 : out STD_LOGIC_VECTOR (6 downto 0));--Conjunto de LEDs que se activan en el display seleccionado
25
26
27
28 end top_digital;
29 --Declaracion de módulos
30 architecture Behavioral of top_digital is
31 component control_pulsadores is
32   Port (clk : in STD_LOGIC;
33         Rst : in STD_LOGIC;
34         Up0 : in STD_LOGIC;
35         Down0 : in STD_LOGIC;
36         Up1 : in STD_LOGIC;
37         Down1 : in STD_LOGIC;
38         b : out STD_LOGIC_VECTOR (5 downto 0));
39 end component;
40
41 component control_displays is
42
43 Port ( clk : in STD_LOGIC;
44         b : in STD_LOGIC_VECTOR (5 downto 0);
45         sw : in STD_LOGIC_VECTOR (5 downto 0);
46         Disp0 : out STD_LOGIC;
47         Disp1 : out STD_LOGIC;
48         Disp2 : out STD_LOGIC;
49         Disp3 : out STD_LOGIC;
50         Seg7 : out STD_LOGIC_VECTOR (6 downto 0));
51 end component;
52
53 component control_global is
54   Port ( clk : in STD_LOGIC;
55         rst : in STD_LOGIC;
56         end_ADC : in STD_LOGIC;
57         end_random : in STD_LOGIC;
58         end_DAC : in STD_LOGIC;
59         start_ADC : out STD_LOGIC;
60         start_random : out STD_LOGIC;
61         start_DAC : out STD_LOGIC;
62         global_st : out STD_LOGIC_VECTOR (1 downto 0));
63 end component;
64
65 component control_ADC is
66   Port ( clk : in STD_LOGIC;
67         Rst : in STD_LOGIC;
68         start_ADC : in STD_LOGIC;
69         DOUT : in STD_LOGIC;
70         end_ADC : out STD_LOGIC;
71         CS0 : out STD_LOGIC;
72         SCLK : out STD_LOGIC;
73         DIN : out STD_LOGIC;
74         valor_ADC : out STD_LOGIC_VECTOR (11 downto 0));
75 end component;
76
77 component pseudo_random is
78   Port ( clk : in STD_LOGIC;
79         rst : in STD_LOGIC;
80         start_random : in STD_LOGIC;
81         sw : in STD_LOGIC_VECTOR (5 downto 0);
82         b : in STD_LOGIC_VECTOR (5 downto 0);
83         valor_ADC : in STD_LOGIC_VECTOR (11 downto 0);
84         end_random : out STD_LOGIC;

```

```

85     valor_DAC : out STD_LOGIC_VECTOR (11 downto 0));
86 end component;
87
88 component control_DAC is
89   Port ( clk : in STD_LOGIC;
90         Rst : in STD_LOGIC;
91         start_DAC : in STD_LOGIC;
92         valor_DAC : in STD_LOGIC_VECTOR (11 downto 0);
93         end_DAC : out STD_LOGIC;
94         CS1 : out STD_LOGIC;
95         SCLK : out STD_LOGIC;
96         DIN : out STD_LOGIC);
97 end component;
98
99 --Señales auxiliares necesarias para la interconexión de módulos
100 signal b_s : STD_LOGIC_VECTOR (5 downto 0);
101 signal sw_s : STD_LOGIC_VECTOR (5 downto 0);
102 signal global_st: STD_LOGIC_VECTOR (1downto 0);
103 signal fin_ADC, fin_DAC, comienzo_ADC, comienzo_DAC, sclk_DAC_aux, sclk_ADC_aux, DIN_DAC_aux, DIN_ADC_aux: STD_LOGIC := '0';
104 signal comienzo_random, fin_random :STD_LOGIC;
105 signal v_ADC, v_DAC : STD_LOGIC_VECTOR (11 downto 0) := (others => '0');
106
107 begin
108
109 U1 : control_pulsadores port map (clk, rst, Up0, Down0, Up1, Down1, b_s);
110 U2 : control_displays port map (clk, b_s, sw_s, Disp0, Disp1, Disp2, Disp3,Seg7);
111 U3 : control_global port map (clk, rst, fin_ADC, fin_random, fin_DAC, comienzo_ADC, comienzo_random, comienzo_DAC,global_st);
112 U4 : control_ADC port map (clk, rst, comienzo_ADC, DOUT, fin_ADC, CS0, sclk_ADC_aux, DIN_ADC_aux, DIN_ADC);
113 U5 : pseudo_random port map (clk, rst,comienzo_random,sw_s,b_s,v_ADC,fin_Random,v_DAC);
114 U6 : control_DAC port map (clk, rst, comienzo_DAC, v_DAC, fin_DAC, CS1, sclk_DAC_aux, DIN_DAC_aux);
115
116 sclk <= sclk_ADC_aux or sclk_DAC_aux; --Sclk final
117 DIN <= DIN_ADC_aux or DIN_DAC_aux;    --Din final
118
119 end Behavioral;

```

5. MEJORAS PARA LA CONVOCATORIA ESTRAORDINARIA

MODIFICACIONES EN EL SUBSISTEMA ANALÓGICO

La modificación pedida en el subsistema analógica es la sustitución del filtro paso bajo de orden 2 del subsistema de entrada, por un filtro que realice la misma función pero siendo este de orden 4.

Para llevar a cabo esta modificación realizamos la asociación en cascada de dos filtros de orden 2 para conseguir el filtro de orden 4 pedido, cuyos calculos desarrollamos a continuación.

Cálculo de los componentes del nuevo filtro de orden 4

$$|H_{LP}(s)|_{s=j2\pi f_c} = \left| \frac{\omega_0^2}{(j2\pi f_c)^2 + j2\pi f_c(\omega_0/0,5) + \omega_0^2} \right|^2 = \frac{1}{\sqrt[4]{2}}$$

$$\frac{\omega_0^2}{\sqrt{(-4,56 \cdot 10^8 + \omega_0^2)^2 + 1,8284 \cdot 10^9 \omega_0^2}} = \sqrt{\frac{1}{\sqrt[4]{2}}}$$

$$\frac{\omega_0^2}{\sqrt{\omega_0^4 + 2,079 \cdot 10^{17} - 9,12 \cdot 10^8 \omega_0^2 + 1,8284 \cdot 10^9 \omega_0^2}} = \frac{1}{\sqrt[4]{2}}$$

$$\frac{\omega_0^2}{\sqrt{\omega_0^4 + 9,134 \cdot 10^8 \omega_0^2 + 2,079 \cdot 10^{17}}} = \frac{1}{\sqrt[4]{2}}$$

$$\sqrt[4]{2} \cdot \omega_0^4 = \omega_0^4 + 9,134 \cdot 10^8 \omega_0^2 + 2,079 \cdot 10^{17} \rightarrow 0 = -0,414 \omega_0^4 + 9,134 \cdot 10^8 \omega_0^2 + 2,079 \cdot 10^{17}$$

$$\text{Tomando } x = \omega_0^2 \rightarrow -0,414 x^2 + 9,134 \cdot 10^8 x + 2,079 \cdot 10^{17} = 0$$

$$x = \frac{-9,134 \cdot 10^8 \pm \sqrt{(9,134 \cdot 10^8)^2 + 4 \cdot 2,079 \cdot 10^{17} \cdot 0,414}}{-2 \cdot 0,414} = \begin{cases} -208001355,4 \\ 2414281579 \end{cases}$$

$$\omega_0 = \sqrt{x} = 49135,34 \text{ rad/s}$$

$$\omega_0 = \frac{1}{RC} \quad \text{Tomamos } C = 15 \text{ nF} \rightarrow R = 1,3 \text{ k}\Omega$$

Cálculo de polos:

$$\left(s^2 + s \left(\frac{49135,34}{0,5} \right) + (49135,34)^2 \right)^2 = 0$$

$$\begin{aligned} s &= -49132,2 - 2,87811j \\ s &= -49132,2 + 2,87811j \\ s &= -49132,5 - 2,87695j \\ s &= -49132,6 + 2,87695j \end{aligned} \quad \left. \begin{array}{l} \text{Polo cuádruple} \\ \text{en módulo en} \end{array} \right\} f = 7820,63$$

Cálculo de los polos de cada filtro individual:

$$s^2 + s \left(\frac{49135,34}{0,5} \right) + (49135,34)^2 = 0$$

$$s = -49135,3 \quad \text{polo doble} \quad \rightarrow \quad f = 7820 \text{ Hz}$$

~~•~~Cálculo de la frecuencia de corte del filtro individual

La frecuencia de corte será la que satisface la siguiente ecuación:

$$\left| \frac{\omega_0^2}{(j2\pi f_c)^2 + j2\pi f_c \left(\frac{\omega_0}{0,5} \right) + \omega_0^2} \right| = \frac{1}{\sqrt{2}}$$

Usando el valor de ω_0 calculado anteriormente. Resolviendo una

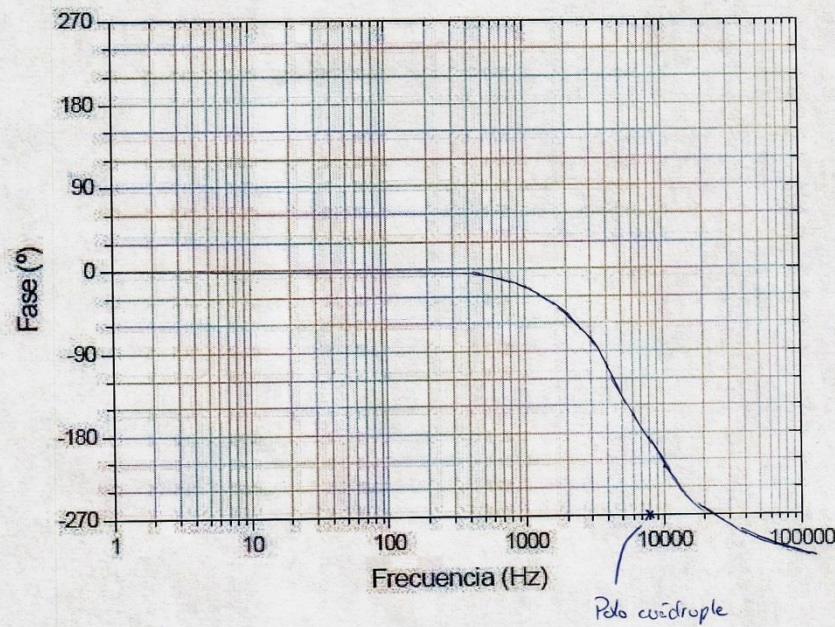
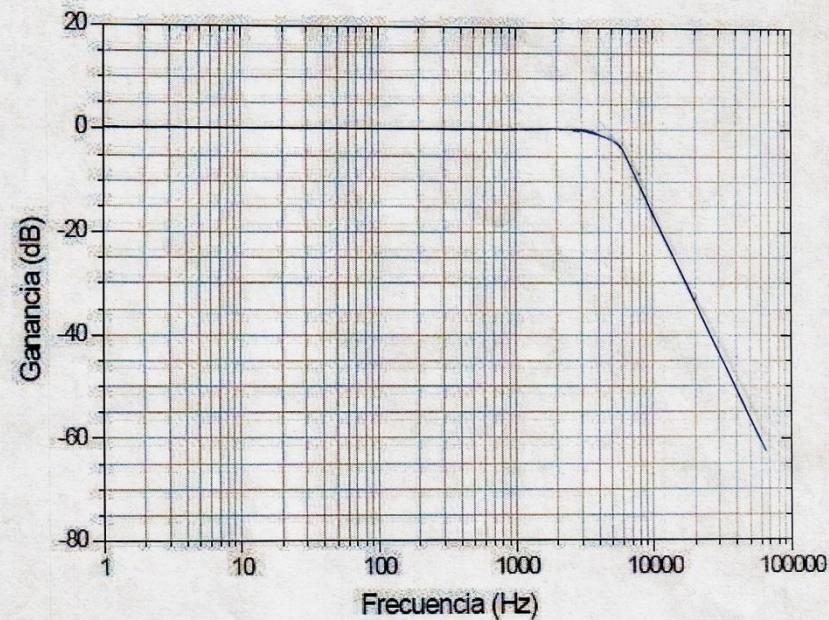
frecuencia de corte $f_c = 5033 \text{ Hz}$

Comportamiento del filtro global

Circuitos Electrónicos

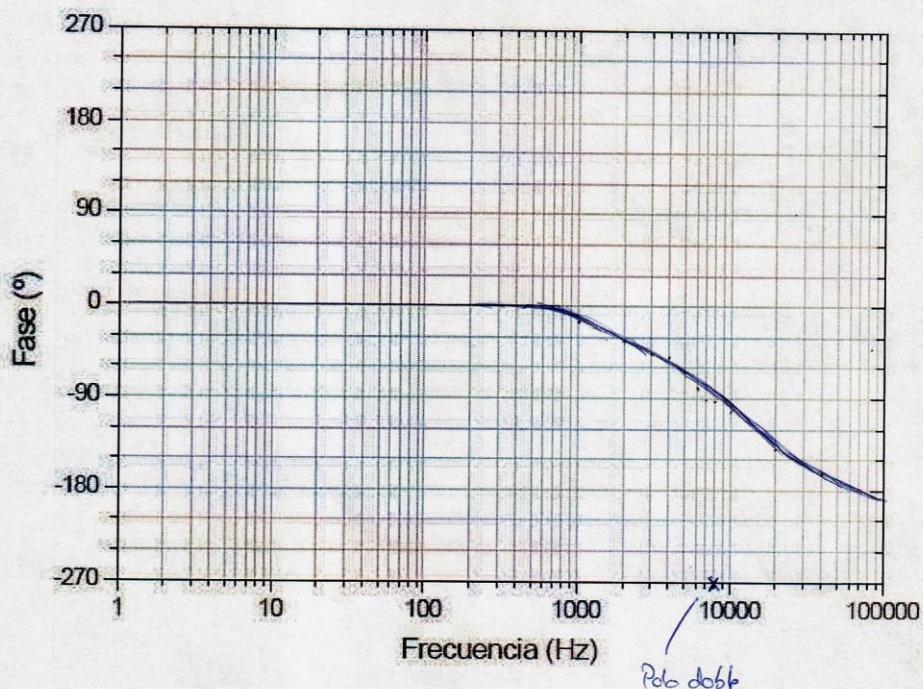
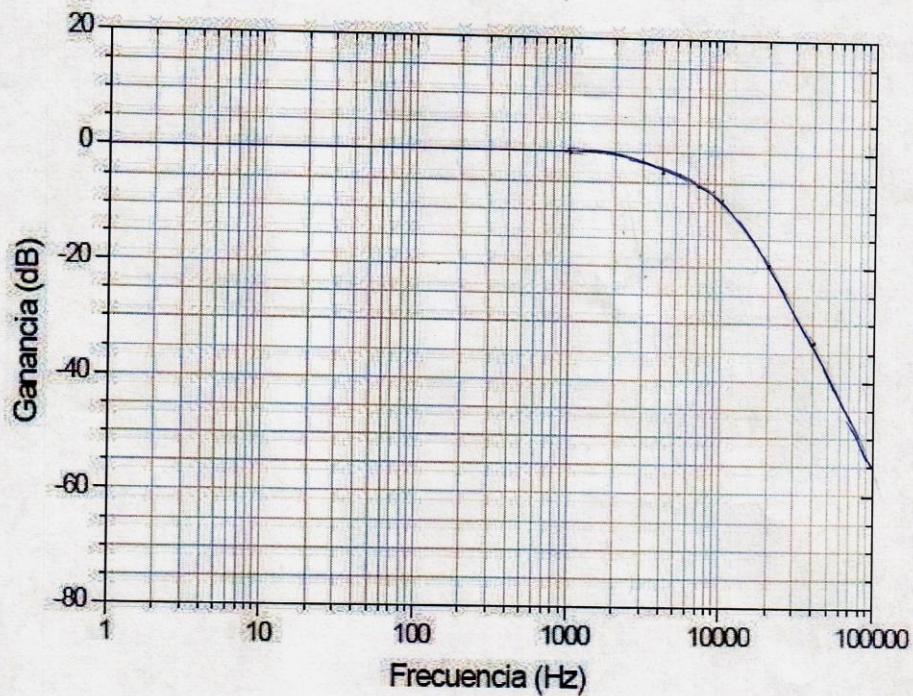
Curso 2016-2017

Medidas del comportamiento del filtro



Comportamiento de los filtros individuales

Medidas del comportamiento del filtro *individual*



MODIFICACIONES EN EL SUBSISTEMA DIGITAL

Para la implementación de la mejora hemos añadido una nueva entrada a Top_Digital relacionada con el switch-6 de la BASYS2 ("E2" en asociaciones .ucf), dicha entrada tiene el identificador de sw6. La entrada sw6 será un 1 en el modo control volumen de la mejora y un 0 para el modo encriptación.

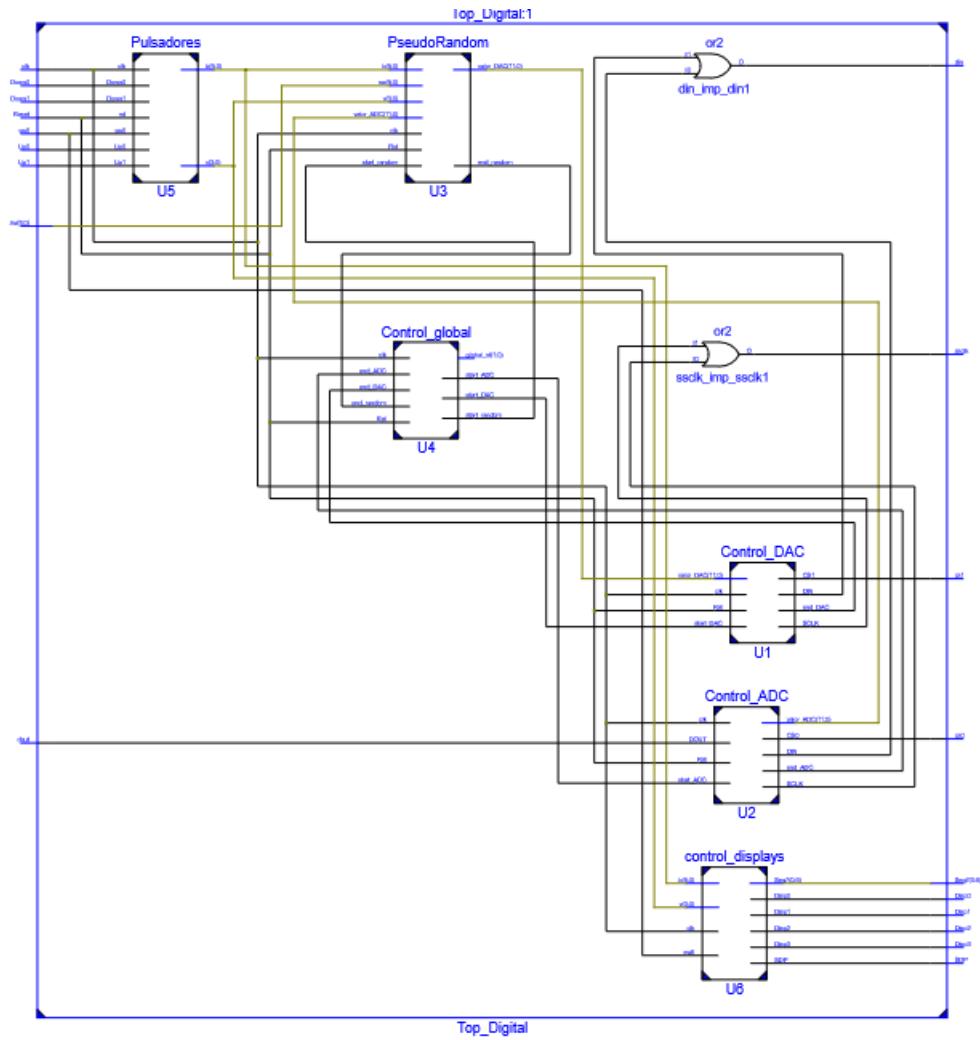
La entrada sw6 se conecta directamente a los módulos control displays y Pulsadores para informar del estado del programa (Volumen o Encriptación).

El bloque Pulsadores almacena el nivel de volumen (Del 1 al 9) en una nueva señal llamada v (3 down to 0) la cual sale de Reg_b. Reg_b se encarga además de controlar el funcionamiento de los pulsadores. Si estamos en modo volumen (sw6 =1) el incremento +10 y decremento -10 están deshabilitados.

La señal v se dirige al bloque Visualización, para mostrar su valor por los leds, y al bloque Pseudorandom para alterar el nivel de la señal.

Hemos añadido una salida a Top_Digital SDP ("N13" en asociaciones .ucf). La salida SDP sale del bloque control displays y gestiona el punto decimal mostrado en los leds. Esta señal ilumina el punto decimal del led más a la izquierda si sw6 es 1 o el led más a la derecha si estamos en el modo de funcionamiento clásico del proyecto.

Finalmente en el bloque encriptación se encarga de atenuar o amplificar el nivel de señal mediante el valor de la señal v.



5. CÓDIGO COMPLETO

A continuación se adjunta el código completo de nuestro proyecto.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Top_Digital is
5    Port(
6      clk : in STD_LOGIC;          -- Señal de reloj
7      Reset : in std_logic;        -- Reset de sistema (SW7)
8      Up0 : in STD_LOGIC;          -- Pulsador 0
9      Down0 : in STD_LOGIC;         -- Pulsador 1
10     Up1 : in STD_LOGIC;          -- Pulsador 2
11     Down1 : in STD_LOGIC;         -- Pulsador 3
12     dout : in STD_LOGIC;          -- Señal que le entra al ADC
13     sw : in STD_LOGIC_VECTOR (5 downto 0); --Swithches
14     sw6: in STD_LOGIC;           --Seleccion del modo de trabajo (SW6)
15     din : out STD_LOGIC;          -- Señal de salida procesada en el DAC y otro por el ADC
16     ssclk : out STD_LOGIC;         --Señal de los pulsos de 1us
17     cs0 : out STD_LOGIC;           --Habilita el funcionamiento del ADC
18     cs1 : out STD_LOGIC;           --Habilita el funcionamiento del DAC
19     Disp0 : out STD_LOGIC;          --Enable display 1
20     Disp1 : out STD_LOGIC;          --Enable display 2
21     Disp2 : out STD_LOGIC;          --Enable display 3
22     Disp3 : out STD_LOGIC;          --Enable display 4
23     SDP : out STD_LOGIC;           --Enable Punto Decimal
24     Seg7 : out STD_LOGIC_VECTOR(0 to 6)); --Segmentos del display
25
26 end Top_Digital;
27
28 architecture Behavioral of Top_Digital is
29
30   signal s_start_ADC:STD_LOGIC:= '0'; --Señal de comienzo del ADC
31   signal s_start_DAC:STD_LOGIC:= '0'; --Señal de comienzo del DAC
32   signal s_start_random:STD_LOGIC;    --Señal de comienzo de encriptacion
33
34   signal s_end_DAC:STD_LOGIC:= '0';    --Señal de finalizacion del ADC
35   signal s_end_ADC:STD_LOGIC:='0';     --Señal de finalizacion del DAC
36   signal s_end_random:STD_LOGIC;       --Señal de finalizacion de encriptacion
37
38   signal s_valor_ADC:STD_LOGIC_VECTOR (11 downto 0):= (others=>'0'); -- señal que saca
ADC con los bits recibidos
39   signal s_valor_DAC:STD_LOGIC_VECTOR (11 downto 0):= (others=>'0'); -- señal que saca
el DAC con los bits recibidos
40
41   signal s_b:STD_LOGIC_VECTOR(5 downto 0); --Señal auxiliar de pulsadores con b que es
la señal asociado al codigo
42   signal s_v:STD_LOGIC_VECTOR(3 downto 0); --Señal auxiliar de pulsadores con v que es
la señal asociado al volumen
43
44   signal s_global_st : STD_LOGIC_VECTOR(1 downto 0); --Estado global del sistema (Señal
de control)
45
46   signal din1 :STD_LOGIC := '0';    --Señal din para el ADC (se asocian a la DIN en un OR)
47   signal din2 :STD_LOGIC := '0';    --Señal din para el DAC (se asocian a la DIN en un OR)
48
49   signal ssclk1 :STD_LOGIC:= '0'; -- señal ssclk para el ADC (se asocian a la SSCLK en
un OR)
50   signal ssclk2 :STD_LOGIC:= '0'; -- señal ssclk para el ADC (se asocian a la SSCLK en
un OR)
```

```

52
53 Component Control_DAC is
54     Port ( clk : IN std_logic;
55             Rst : IN std_logic;
56             start_DAC : IN std_logic;
57             end_DAC : OUT std_logic;
58             CS1 : OUT std_logic;
59             SCLK : OUT std_logic;
60             DIN : OUT std_logic;
61             valor_DAC : IN std_logic_vector (11 downto 0) );
62 end component;
63
64 Component Control_ADC is
65     Port ( clk : in STD_LOGIC;
66             Rst : in STD_LOGIC;
67             start_ADC : in STD_LOGIC;
68             end_ADC : out STD_LOGIC;
69             CS0 : out STD_LOGIC;
70             SCLK : out STD_LOGIC;
71             DIN : out STD_LOGIC;
72             DOUT : in STD_LOGIC;
73             valor_ADC : out STD_LOGIC_VECTOR (11 downto 0));
74 END COMPONENT;
75
76 component PseudoRandom is
77     Port ( clk : in STD_LOGIC;
78             Rst : in STD_LOGIC;
79             start_random : in STD_LOGIC;
80             end_random : out STD_LOGIC;
81             sw : in STD_LOGIC_VECTOR (5 downto 0);
82             b : in STD_LOGIC_VECTOR (5 downto 0);
83             v : in STD_LOGIC_VECTOR (3 downto 0);
84             valor_ADC : in STD_LOGIC_VECTOR (11 downto 0);
85             valor_DAC : out STD_LOGIC_VECTOR (11 downto 0));
86
87 END COMPONENT;
88
89 component Control_global is
90     Port ( clk : in STD_LOGIC;
91             Rst : in STD_LOGIC;
92             start_ADC : out STD_LOGIC;
93             start_random : out STD_LOGIC;
94             start_DAC : out STD_LOGIC;
95             end_ADC : in STD_LOGIC;
96             end_random : in STD_LOGIC;
97             end_DAC : in STD_LOGIC;
98             global_st : out STD_LOGIC_VECTOR (1 downto 0));
99 end component;
100
101 component Pulsadores is
102     Port ( clk : in STD_LOGIC;
103             rst : in STD_LOGIC;
104             Up0 : in STD_LOGIC;
105             Down0 : in STD_LOGIC;
106             Up1 : in STD_LOGIC;
107             Down1 : in STD_LOGIC;
108             sw6 : in STD_LOGIC;
109             b : out STD_LOGIC_VECTOR(5 downto 0);

```

Top_Digital.vhd

```

110           v : out STD_LOGIC_VECTOR(3 downto 0));
111    end component;
112
113  component control_displays is
114    Port ( clk : in STD_LOGIC;
115            b : in STD_LOGIC_VECTOR (5 downto 0);
116            v : in STD_LOGIC_VECTOR(3 downto 0);
117            Disp0 : out STD_LOGIC;
118            Disp1 : out STD_LOGIC;
119            Disp2 : out STD_LOGIC;
120            Disp3 : out STD_LOGIC;
121            sw6 : in STD_LOGIC;
122            SDP : out STD_LOGIC;
123            Seg7 : out STD_LOGIC_VECTOR (0 to 6));
124  end component;
125
126
127 begin
128
129 U1: Control_DAC
130 port map(
131     clk => clk,
132     Rst => Reset,
133     start_DAC => s_start_DAC,
134     end_DAC => s_end_DAC,
135     CS1 => cs1,
136     SCLK => ssclk2,
137     DIN => din2,
138     valor_DAC =>s_valor_DAC
139 );
140
141 U2: Control_ADC
142 port map(
143     clk => clk,
144     Rst => Reset,
145     start_ADC => s_start_ADC,
146     end_ADC => s_end_ADC,
147     CS0 => cs0,
148     SCLK => ssclk1,
149     DOUT => dout,
150     DIN => din1,
151     valor_ADC =>s_valor_ADC
152 );
153
154 U3: PseudoRandom
155 port map(
156     clk => clk,
157     Rst => Reset,
158     start_random => s_start_random,
159     end_random =>s_end_random,
160     sw => sw,
161     b =>s_b,
162     v =>s_v,
163     valor_ADC => s_valor_ADC,
164     valor_DAC => s_valor_DAC
165 );
166
167 U4: Control_global

```

```
168      port map(
169          clk => clk,
170          Rst => Reset,
171          start_ADC => s_start_ADC,
172          start_random => s_start_random,
173          start_DAC => s_start_DAC,
174          end_ADC => s_end_ADC,
175          end_random => s_end_random,
176          end_DAC => s_end_DAC,
177          global_st => s_global_st
178      );
179
180 U5: Pulsadores
181     port map(
182         clk => clk,
183         rst => Reset,
184         Up0 => Up0,
185         Down0 => Down0,
186         Up1 => Up1,
187         Down1 => Down1,
188         sw6 =>sw6,
189         v => s_v,
190         b =>s_b
191     );
192
193 U6: control_displays
194     port map(
195         clk => clk,
196         b => s_b,
197         v => s_v,
198         Disp0 => Disp0,
199         Disp1 => Disp1,
200         Disp2 =>Disp2,
201         Disp3 =>Disp3,
202         sw6 =>sw6,
203         SDP => SDP,
204         Seg7 => Seg7
205     );
206
207     SSCLK<= ssclk1 or ssclk2;
208     DIN <= din1 or din2;
209
210
211 end Behavioral;
212
213
214
215
216
```

```
1 -----
2 --CONTROL DAC
3 --El bloque de control del DAC es el encargado de generar
4 --todas las señales digitales que necesita el DAC para realizar la conversión
5 -----
6 library IEEE;
7 use IEEE.STD_LOGIC_1164.ALL;
8 use IEEE.STD_LOGIC_ARITH.ALL;
9 use IEEE.STD_LOGIC_UNSIGNED.ALL;
10
11
12 entity Control_DAC is
13     Port ( clk : in STD_LOGIC;    --Entrada de reloj
14            Rst : in STD_LOGIC;   --Reset del sistema
15            start_DAC : in STD_LOGIC; --Señal que da comienzo al bloque
16            end_DAC : out STD_LOGIC;  --Señal de Finalizado el bloque DAC
17            valor_DAC : in STD_LOGIC_VECTOR (11 downto 0); --Señal procedente del
18            encriptador.
19            CS1 : out STD_LOGIC;    --Señal de control del DAC
20            SCLK : out STD_LOGIC;  --Señal de control
21            DIN : out STD_LOGIC); --envio de los datos del DAC
22 end Control_DAC;
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
```

```

52      end if;
53  end if;
54
55 end process;
56
57
58 --SEGUNDO CONTADOR
59 process (clk)
60 begin
61 if clk'event and clk='1' then
62     if( fin_de_cuenta = '1' and s_sclk ='0')then
63         s_sclk<='1';
64     end if;
65     if (fin_de_cuenta = '1' and s_sclk ='1')then
66         s_sclk<='0';
67     end if;
68 end if;
69 end process;
70 sclk<=s_sclk;
71
72 PROCESS (Rst, clk)
73 BEGIN
74 IF (Rst = '1') THEN
75     end_DAC<='0';
76     en_cnt <= '0';
77     bit_DIN <= "0000";
78     dato_DAC <= "000000000000000000000000";
79
80 ELSIF clk'event and clk = '1' THEN
81
82 CASE n_state IS
83 --ESTADO DE REPOSO
84 WHEN st_00 =>
85     end_DAC<='0';
86     en_cnt <= '0';
87     dato_DAC <= "000" & valor_DAC & '0';
88     IF (start_DAC = '1') THEN --Permiso para que el DAC empiece
89         n_state <= st_01;
90     END IF;
91
92
93 --ESTADO DE ENVIO DE LOS DOS BYTES
94 WHEN st_01=>
95     end_DAC<='0';
96     en_cnt <= '1';
97     IF (fin_de_cuenta = '1' and s_sclk = '1') THEN --se van desplizando los bits al
15 que es el que se va sacando por DIN
98         dato_DAC <= dato_DAC(14 downto 0) & '0';
99         bit_DIN <= bit_DIN + "01";
100        IF bit_DIN >= "1111" THEN -- condicion de cambio de estado
101            n_state <= st_11;
102            bit_DIN <= (others => '0'); --bit-DIN a 0
103        END IF;
104    END IF;
105 --FIN DEL ESTADO DE ENVIO DE LOS DOS BYTES
106
107 --ESTADO DE FINALIZACION
108 WHEN st_11 =>

```

```
109      n_state <= st_00;
110      end_DAC<='1';
111      WHEN others =>
112          end_DAC<='0';
113          n_state <= st_00;
114      END CASE;
115      END IF;
116  END PROCESS;
117  DIN <= dato_DAC(15) when (n_state= st_01) else '0'; --Saca cada unos de los nuevos
  "bits15" y su desplazamiento. Se ejecuta a la vez
118  CS1 <= '0' when (n_state = st_01) else '1';
119  end Behavioral;
120
121
```

```
1 -----
2 --- CONTROL ADC
3 --El bloque de control del ADC es el encargado de enviar y recibir todas las señales
4 --que deben llegar al ADC.
5 --Este bloque debe realizar 2 cosas: En primer lugar, enviar el byte de petición por
6 --medio de las señales CS0, SCLK y DIN.
7 --En segundo, recibir la respuesta a través de la señal DOUT y entregar el valor
8 --digital que se reciba a la siguiente etapa
9 -----
10 -----
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13 use IEEE.STD_LOGIC_ARITH.ALL;
14 use IEEE.STD_LOGIC_UNSIGNED.ALL;
15
16 entity Control_ADC is
17     Port ( clk : in STD_LOGIC;          --Reloj del sistema
18             Rst : in STD_LOGIC;          --Reset del sistema
19             start_ADC : in STD_LOGIC;    --Enable del ADC
20             end_ADC : out STD_LOGIC;    -- Final del bloque
21             CS0 : out STD_LOGIC;        -- Salida que indica el uso del ADC y su control
22             SCLK : out STD_LOGIC;       -- Señal interna de un contador de lus
23             DIN : out STD_LOGIC;        -- Salida que saca de 1 en 1 los bits que se van
24             procesando a los distintos modulos
25             DOUT : in STD_LOGIC;        -- Entrada de los bits al ADC
26             valor_ADC : out STD_LOGIC_VECTOR (11 downto 0); -- Salida del ADC con los
27             datos
28 end Control_ADC;
29
30
31 architecture Behavioral of Control_ADC is
32
33 type STATE_TYPE is (st_00,st_01,st_11,st_10); --Estados
34
35 --Señales para el primer contador
36 signal contador_primero : STD_LOGIC_VECTOR(5 downto 0) := "000000"; --Primer contador
37 de un lus
38 signal fin_de_cuenta : STD_LOGIC; --Cuenta cuando el contador llega al valor deseado
39 signal s_sclk : STD_LOGIC := '0'; --señal interna de sclk internconecta los contadores
40
41 signal n_state : STATE_TYPE := st_00;
42 signal bit_DIN : STD_LOGIC_VECTOR(3 DOWNTO 0);
43 signal byte_peticion_ADC : STD_LOGIC_VECTOR(7 DOWNTO 0) := (others => '0');
44 signal en_cnt : STD_LOGIC; --señal de habilitacion de los contadores
45 signal dato_ADC : STD_LOGIC_VECTOR(15 DOWNTO 0); -- dato que se manda 2 bytes al
46 enciptador con los "0-000"
47
48 begin
49 --PRIMER CONTADOR
50 process(clk)
51 begin
52     if clk'event and clk='1' then      --contador que activa un pulso cada lus
53         fin_de_cuenta <= '0';
54         if (en_cnt='1') then
55             contador_primero <= contador_primero + '1';
56         end if;
57         if contador_primero >= "110010" then      --50 equivale al numero en binario ya
```

Control_ADC.vhd

```

que el periodo del reloj es de 20ns y necesitamos 50 pulsos de reloj para 1us
50      fin_de_cuenta <= '1';
51      contador_primero <= (others => '0');
52  end if;
53  if rst='1' then
54      fin_de_cuenta <= '0';
55      contador_primero <= (others => '0');
56  end if;
57 end if;
58
59
60 end process;
61
62 --SEGUNDO CONTADOR
63 process (clk)
64 begin
65
66  if clk'event and clk='1' then
67      if( fin_de_cuenta = '1' and s_sclk ='0')then    --segun lo entendido en el
enunciado fin de cuenta hace que cambien la señal s_sclk segun lo que hay anteriormente
68      s_sclk<='1';                                --por lo tanto activa un pulso de
ssclk durante un 1us y luego otro 1us a '0'
69  end if;
70  if (fin_de_cuenta = '1' and s_sclk ='1')then
71      s_sclk<='0';
72  end if;
73  end if;
74 end process;
75 sclk <= s_sclk;
76
77
78 --PROCESO PRINCIPAL DEL ADC
79 PROCESS (Rst, clk)
80 BEGIN
81  IF (Rst = '1') THEN          -- declaracion de los valores por defecto
82      valor_ADC<="00000000000000";
83      end_ADC<='0';
84      n_state<=st_00;
85      en_cnt <= '0';
86      bit_DIN <= "0000";
87      dato_ADC <= (others=>'0');
88
89  ELSIF clk'event and clk = '1' THEN
90      CASE n_state IS
91      --ESTADO DE REPOSO
92      WHEN st_00 =>
93          end_ADC <= '0';
94          byte_peticion_ADC <= "10010111"; --97 en HEXADECIMAL
95          IF start_ADC = '1' THEN --condicion para el cambio de estado
96              n_state <= st_01; --siguiente estado
97          END IF;
98
99  --ENVIO DEL BYTE DE PETICION
100     WHEN st_01 => --envio del bit del byte de peticion mas signficativo
101         en_cnt <= '1';
102         end_ADC <= '0';
103         IF (fin_de_cuenta = '1' and s_sclk = '1') THEN
104             byte_peticion_ADC <= byte_peticion_ADC(6 downto 0) & '0'; --Concatenacion

```

```
de bits
105      bit_DIN <= bit_DIN + "01"; --aumentamos la cuenta de bit_DIN
106      IF bit_DIN >= "0111" THEN --en verdad el primero ya esta enviado luego
107          quedan 7 mas por enviar. Condicion de cambio de estado
108          n_state <= st_10; --pasamos al siguiente estado
109          bit_DIN <= (others => '0'); --ponemos Bit-din a todo ceros aqui ya que
110          se vuelve a usar en el siguiente estado
111      END IF;
112      END IF;

113  --RECEPCION DE LOS BYTES DE RESPUESTA
114      WHEN st_10 =>           --modulo de recepcion de los bits
115          en_cnt<='1';
116          end_ADC <= '0';
117          IF (fin_de_cuenta = '1' and s_sclk = '0') THEN
118              dato_ADC <= dato_ADC(14 downto 0) & DOUT;
119          END IF;
120          IF (fin_de_cuenta = '1' and s_sclk = '1') THEN
121              bit_DIN <= bit_DIN + "01";
122              IF bit_DIN = "1111" THEN
123                  n_state <= st_11;
124                  bit_DIN <= (others => '0');
125              END IF;
126          END IF;

127  --ENVIO DE LA SEÑAL DEL ADC
128
129      WHEN st_11 =>
130          en_cnt <= '0';
131          n_state <= st_00;
132          end_ADC <= '1'; --Finalizacion del proceso del ADC
133          valor_ADC <= dato_ADC(14 downto 3); --se envian 12 bits al ADC se quitan los ceros
134
135
136      WHEN others =>
137          end_ADC <= '0';
138          n_state <= st_00;
139      END CASE;
140  END IF;
141 end process;
142 DIN <= '0' when ((n_state =st_11) or (n_state= st_00)) else byte_peticion_adc(7) ;
143 --Saca cada unos de nuestros nuevos "bits7" se ejecuta en paralelo
144 CS0 <= '0' when ((n_state = st_01) or (n_state = st_10)) else '1';
145 end Behavioral;
146
```

```
1 -----
2 -----
3 --PSEUDORANDOM
4 --El bloque de encriptación de la señal es el encargado de comparar el código
5 predefinido de cifrado (fijado en los interruptores)
6 --con el código introducido por el usuario (generado mediante los pulsadores). En caso
7 de que sean iguales,
8 --la señal de entrada se transmitirá a la salida.
9 --En caso de que no lo sean, se entrega al DAC la salida de un generador
10 pseudo-aleatorio de 16 bits.
11 -----
12 -----
13
14 entity PseudoRandom is
15     Port ( clk : in STD_LOGIC;    -- Reloj del sistema
16             Rst : in STD_LOGIC;    -- Reset del sistema
17             start_random : in STD_LOGIC;    --Enable del encriptador
18             end_random : out STD_LOGIC;    --Finalizacion del bloque
19             sw : in STD_LOGIC_VECTOR (5 downto 0);    --Codigo predefinido
20             b : in STD_LOGIC_VECTOR (5 downto 0);    --Codigo de los pulsadores
21             v : in STD_LOGIC_VECTOR (3 downto 0);    --Valor del volumen
22             valor_ADC : in STD_LOGIC_VECTOR (11 downto 0);    --Valor que llega del ADC
23             valor_DAC : out STD_LOGIC_VECTOR (11 downto 0));    --Valor mandado al DAC
24
25 end PseudoRandom;
26
27 architecture Behavioral of PseudoRandom is
28 --Señales auxiliares del bloque PseudoRandom
29 signal primeraXOR : STD_LOGIC := '0';    -- Salida de la primera puerta XOR combinada
30 con la posicion 13 y 15
31 signal segundaXOR : STD_LOGIC := '0';    -- Salida de la segunda puerta XOR combinada
32 con la posicion 12
33 signal terceraXOR : STD_LOGIC := '0';    -- Salida de la tercera puerta XOR combinada
34 con la posicion 10
35 signal Q :STD_LOGIC_VECTOR(15 downto 0) := "0011001100110101";
36 signal v_ADC : STD_LOGIC_VECTOR (11 downto 0);
37
38 begin
39
40 process(clk,Rst)
41 begin
42     if clk'event and clk='1' then
43         end_random<='0';
44         if Rst='1' then          -- declaracion de los valores por defecto cuando se
45 activa el reset
46             valor_DAC<= (others=>'0');
47             end_random<='0';
48             Q <= "1111000010100101";--1111000010100101 --Secuencia aleatoria
49         elsif start_random = '1' then -- Si el enable esta activado
50             primeraXOR <= Q(13) XOR Q(15);      --combinacion de la primera XOR
51             segundaXOR<= primeraXOR XOR Q(12);  --combinacion de la segunda XOR
52             terceraXOR<= segundaXOR XOR Q(10);  --combinacion de la tercera XOR
```

```

50
51      Q(15 downto 1)<= Q(14 downto 0); -- desplazamos los numeros del 0 al 14 a la
52      posicion del 1 al 15
53      Q(0)<= terceraXOR;           -- y cambiamos el valor en la posicion 0 con
54      el nuevo valor
55
56      if (sw = b) then --Si la clave coincide con la clave predefinida la señal
57      pasa bien
58
59      CASE v IS --Maquina que controla el volumen segun lo pulsado
60
61      when "0001" => --1
62          if (valor_ADC(11) = '0') then
63              v_ADC<= "0000" & valor_ADC(11 downto 4);
64          else
65              v_ADC<= "1111" & valor_ADC(11 downto 4);
66          end if;
67
68      when "0010" => --2
69          if (valor_ADC(11) = '0') then
70              v_ADC<= "000" & valor_ADC(11 downto 3);
71          else
72              v_ADC<= "111" & valor_ADC(11 downto 3);
73          end if;
74
75      when "0011" => --3
76          if (valor_ADC(11) = '0') then
77              v_ADC<= "00" & valor_ADC(11 downto 2);
78          else
79              v_ADC<= "11" & valor_ADC(11 downto 2);
80          end if;
81
82      when "0100" => --4
83          if (valor_ADC(11) = '0') then
84              v_ADC<= "0" & valor_ADC(11 downto 1);
85          else
86              v_ADC<= "1" & valor_ADC(11 downto 1);
87          end if;
88
89      when "0101" => --5
90
91          v_ADC<= valor_ADC;
92
93      when "0110" => --6
94
95          v_ADC<= valor_ADC(10 downto 0) & "0";
96
97      when "0111" => --7
98
99          v_ADC<= valor_ADC(9 downto 0) & "00";
100
101     when "1000" => --8
102
103         v_ADC<= valor_ADC(8 downto 0) & "000";
104

```

```

105      when "1001" => --9
106
107      v_ADC<= valor_ADC(7 downto 0) & "0000";
108
109
110      when others =>
111
112      v_ADC<=valor_ADC;
113
114  END CASE;
115
116      valor_DAC<= v_ADC + "100000000000"; --X800 hexadecimal a binario --sacamos
el valor del ADC mas 800 ya que es de caracter bipolar
117      end_random<='1'; --Fin del bloque
118      else --Si la clave esta mal
119      valor_DAC<= Q(15 downto 4); --La salida es el valor aleatorio generado
al DAC.
120      end_random<='1';
121      end if;
122
123      end if;
124
125
126  end if;
127 end process;
128
129
130
131 end Behavioral;
132
133

```

```
1 -----  
2 --CONTROL GLOBAL  
3 --El bloque de control global es el encargado de gestionar cuándo tiene que comenzar  
4 --el procesamiento de cada uno de los bloques que dependen de él. Por simplicidad  
5 --utilizaremos un protocolo de comunicación asíncrona basado en una señal de inicio  
6 --y otra señal de fin. De esta forma, cada bloque comenzará su procesamiento cuando  
7 --haya terminado el bloque inmediatamente anterior. Puesto que en las especificaciones  
8 --se indica que la frecuencia de muestreo deben ser 8 kHz,  
9 --un contador interno generará cada 125 µs la señal de inicio del primero de los  
bloques.  
10 -----  
11 library IEEE;  
12 use IEEE.STD_LOGIC_1164.ALL;  
13 use IEEE.STD_LOGIC_ARITH.ALL;  
14 use IEEE.STD_LOGIC_UNSIGNED.ALL;  
15  
16  
17 entity Control_global is  
18     Port ( clk : in STD_LOGIC;      -- Reloj del sistema  
19             Rst : in STD_LOGIC;      -- Reset del sistema  
20             start_ADC : out STD_LOGIC;-- Enable del adc  
21             start_random : out STD_LOGIC;-- Enable del random  
22             start_DAC : out STD_LOGIC; --Enable del DAC  
23             end_ADC : in STD_LOGIC; -- Fin del adc  
24             end_random : in STD_LOGIC; -- Fin del random  
25             end_DAC : in STD_LOGIC; -- Fin del dac  
26             global_st : out STD_LOGIC_VECTOR (1 downto 0)); --Señal de control de estado  
27 end Control_global;  
28  
29 architecture Behavioral of Control_global is  
30  
31  
32 signal st : STD_LOGIC_VECTOR(1 downto 0);  
33  
34 signal contador : STD_LOGIC_VECTOR(12 downto 0) := "00000000000000"; --señal auxiliar  
para contar los 125us  
35 signal cnt_125us : STD_LOGIC; --señal auxiliar que activa un pulso cuando la cuenta  
llega a los 125us  
36 begin  
37  
38 --CONTADOR DE 125US  
39 process (clk)  
40 begin  
41     if clk'event and clk='1' then  
42         cnt_125us <= '0';  
43         contador <= contador + '1';  
44         if contador >= "1100001101010" then --6250 --valor máximo de la cuenta.  
45             cnt_125us<='1';  
46             contador <= (others => '0');  
47         end if;  
48         if Rst = '1' then  
49             cnt_125us <= '0';  
50             contador <= (others => '0');  
51         end if;  
52     end if;  
53 end process;  
54  
55
```

```

56  --INICIO DEL PROCESO PRINCIPAL
57  process (clk,Rst)
58  begin
59      if (Rst = '1') THEN --Valores por defecto.
60          start_ADC <= '0';
61          start_random <= '0';
62          start_DAC <= '0';
63          st <= "00";
64
65      ELSIF clk'event and clk = '1' THEN
66          start_ADC <= '0';
67          start_random <= '0';
68          start_DAC <= '0';
69      CASE st IS
70      --ESTADO INICIAL
71          WHEN "00" => --estado de reposo
72              start_ADC <= '0';
73              start_random <= '0';
74              start_DAC <= '0';
75
76              IF cnt_125us ='1' THEN --pasa al siguiente estado cuando se activa la cuenta
77                  start_ADC <= '1';
78                  st <= "01";
79              END IF;
80
81      --ESTADO ADC
82          WHEN "01" =>
83
84              IF ( end_ADC ='1') then --pasa al siguiente estado si ha acabado el ADC
85                  start_random <='1';
86                  st <= "11";
87              END IF;
88
89
90      --ESTADO DAC
91          WHEN "11" =>
92              if (end_random ='1') then --Si finaliza random
93                  start_DAC<='1';           --Empieza el DAC
94                  st <= "10";
95              end if;
96
97          WHEN "10" =>
98              IF ( end_DAC ='1') then
99                  st <= "00";
100             END IF;
101          WHEN others =>
102              st <= "00";
103          END CASE;
104      END IF;
105  end process;
106  global_st <=st;
107
108
109 end Behavioral;
110
111

```

```
1 ---CONTROL DE PULSADORES
2 -- El bloque de control de pulsadores se encarga
3 -- de seleccionar el valor de la clave de
4 -- del programa y su volumen.
5 -----
6
7 library IEEE;
8 use IEEE.STD_LOGIC_1164.ALL;
9
10
11 entity Pulsadores is
12     Port ( clk : in STD_LOGIC;                      -- Reloj del sistema
13            rst : in STD_LOGIC;                      -- Reset del sistema
14            Up0 : in STD_LOGIC;                       -- Pulsador 0 o +1
15            Down0 : in STD_LOGIC;                     -- Pulsador 1 o -1
16            Up1 : in STD_LOGIC;                       -- Pulsador 2 o +10
17            Down1 : in STD_LOGIC;                     -- Pulsador 3 o -10
18            sw6 : in STD_LOGIC;
19            b : out STD_LOGIC_VECTOR(5 downto 0);    -- Clave introducida
20            v : out STD_LOGIC_VECTOR(3 downto 0));   -- Volumen introducido
21 end Pulsadores;
22
23 architecture Behavioral of Pulsadores is
24     --Señales auxiliares
25     signal s_rst_cnt:STD_LOGIC;
26     signal s_en_cnt:STD_LOGIC ;
27     signal s_cnt_100ms:STD_LOGIC;
28     signal s_rst_b:STD_LOGIC;
29     signal s_en_b:STD_LOGIC ;
30     signal s_sel:STD_LOGIC_VECTOR(1 downto 0);
31
32 component MooreFSM is
33     Port ( clk : in STD_LOGIC;
34            Up0 : in STD_LOGIC;
35            Down0 : in STD_LOGIC;
36            Up1 : in STD_LOGIC;
37            Down1 : in STD_LOGIC;
38            cnt_100ms : in STD_LOGIC;
39            Rst : in STD_LOGIC;
40
41            rst_cnt : out STD_LOGIC;
42            en_cnt : out STD_LOGIC;
43            rst_b : out STD_LOGIC;
44            en_b : out STD_LOGIC;
45            sel : out STD_LOGIC_VECTOR (1 downto 0));
46 end component;
47
48 component contador_100ms is
49     Port ( clk : in STD_LOGIC;
50            rst_cnt : in STD_LOGIC;
51            en_cnt : in STD_LOGIC;
52            cnt_100ms : out STD_LOGIC);
53 end component;
54
55 component Reg_b is
56     Port ( clk : in STD_LOGIC;
57            rst_b : in STD_LOGIC;
58            en_b : in STD_LOGIC;
```

```
59      sw6 : in STD_LOGIC;
60      sel : in STD_LOGIC_VECTOR(1 downto 0);
61
62      b : out STD_LOGIC_VECTOR(5 downto 0);
63      v : out STD_LOGIC_VECTOR(3 downto 0));
64
65 end component;
66
67 begin
68
69 U1: MooreFSM          --Port Map
70 port map(
71     clk =>clk,
72     Up0 => Up0,
73     Down0 =>Down0,
74     Up1 =>Up1,
75     Down1 => Down1,
76     cnt_100ms => s_cnt_100ms,
77     Rst => rst,
78     rst_cnt=> s_rst_cnt,
79     rst_b=>s_rst_b,
80     en_cnt=> s_en_cnt,
81     en_b => s_en_b,
82     sel => s_sel ) ;
83
84 U2: contador_100ms
85 port map(
86     clk => clk,
87     rst_cnt => s_rst_cnt,
88     en_cnt => s_en_cnt,
89     cnt_100ms => s_cnt_100ms);
90
91 U3: Reg_b
92 port map(
93     clk => clk,
94     rst_b =>s_rst_b,
95     en_b =>s_en_b,
96     sel => s_sel,
97     v => v,
98     sw6 => sw6,
99     b => b);
100
101 end Behavioral;
102
103
104
```

```
1 -----  
2 --AUTÓMATA DE MOORE  
3 --Máquina de estados (Finite State Machine, FSM) de Moore que controla el  
4 --funcionamiento del módulo. Recibe las señales de los pulsadores (Up0, Down0, Up1 y  
Down1) y el  
5 --interruptor (Rst), gestiona la cuenta de 100 ms. (mediante las señales rst_cnt,  
en_cnt, y cnt_100ms), y  
6 --controla el valor del registro Reg_b (mediante las señales rst_b, en_b y sel).  
7 -----  
8 library IEEE;  
9 use IEEE.STD_LOGIC_1164.ALL;  
10  
11  
12 entity MooreFSM is  
13     Port ( clk : in STD_LOGIC;                                     --Reloj del sistema  
14             Up0 : in STD_LOGIC;                                     -- Pulsador 0 o +1  
15             Down0 : in STD_LOGIC;                                    -- Pulsador 1 o -1  
16             Up1 : in STD_LOGIC;                                     -- Pulsador 2 o +10  
17             Down1 : in STD_LOGIC;                                    -- Pulsador 3 o decremento -10  
18             cnt_100ms : in STD_LOGIC;                                -- Fin de cuenta  
19             Rst : in STD_LOGIC;                                     -- Reset del sistema  
20             sw6: in STD_LOGIC;                                     -- Reset de contador  
21             rst_cnt : out STD_LOGIC;                                -- Enable de la cuenta de 100ms  
22             en_cnt : out STD_LOGIC;                                -- Reset de b  
23             rst_b : out STD_LOGIC;                                 -- Enable de b  
24             en_b : out STD_LOGIC;                                 -- Estado del sistema  
25             sel : out STD_LOGIC_VECTOR (1 downto 0));  
26 end MooreFSM;  
27  
28 architecture Behavioral of MooreFSM is  
29  
30 type STATE_TYPE is(st_Reset,st_Main,st_Up0a,st_Up0b,          -- Estados del Moore  
31 st_Upla,st_Uplb,st_Down0a,st_Down0b,st_Down1a,st_Down1b);  
32  
33 signal n_state : STATE_TYPE := st_Main; --estado Main(inicial por defecto)  
34  
35  
36  
37  
38 begin  
39 process(CLK)  
40 begin  
41     if(clk'event and clk='1') then  
42         --Valores por defecto a las salidas  
43         rst_cnt <= '1';    --en cada ejecución debemos poner si ha ocurrido que se ha  
activado el contador 100ms a '0'  
44         en_cnt <= '0';  
45         rst_b <= '0';  
46         en_b <= '0';  
47         sel <= "00";  
48  
49         case n_state is           --Estado de reset que resetea la b  
50             when st_Reset =>  
51                 rst_b <= '1';  
52                 n_state <= st_Main;  
53             when st_Main =>        --Estado Main del Moore desde el que se accede a  
el resto de estados  
54                 if(Rst = '1') then
```

```
55          n_state <= st_Reset;
56      elsif(Up0 = '1') then -- estado que se da si se pulsa el pulsador
57          asociado a +1 ( en nuestro caso el primero)
58          n_state <= st_Up0a;
59      elsif(Down0 = '1') then -- estado que se da si se pulsa el pulsador
60          asociado a -1 ( en nuestro caso el segundo)
61          n_state <= st_Down0a;
62      elsif(Up1 = '1') then -- estado que se da si se pulsa el pulsador
63          asociado a +10 ( en nuestro caso el tercero)
64          n_state <= st_Upla;
65      elsif(Down1 = '1') then -- estado que se da si se pulsa el pulsador
66          asociado a -10 ( en nuestro caso el cuarto)
67          n_state <= st_Down1a;
68      end if;

69      -----INCREMENTO +1-----
70
71      when st_Up0a =>           --estado al que se llega despues de Up0
72          rst_cnt <= '0';
73          en_cnt <= '1';
74          if(cnt_100ms = '1') then --espera hasta que se cumpla la condicion de
que llegue a los 100ms
75          n_state <= st_Up0b;    --llega a 100ms y pasa al siguiente estado
76          else
77          n_state <= st_Up0a;   --si no ha pasado de estado sigue en este hasta
que llegue a los 100ms
78          end if;
79      when st_Up0b =>
80          en_b <= '1';
81          sel <= "00";
82          n_state <= st_Main;

83      -----INCREMENTO +10-----
84
85      when st_Upla =>
86          rst_cnt <= '0';
87          en_cnt <= '1';
88          if(cnt_100ms = '1') then
89          n_state <= st_Uplb;
90          else
91          n_state <= st_Upla;
92          end if;
93      when st_Uplb =>
94          en_b <= '1';
95          sel <= "10";
96          n_state <= st_Main;

97      -----DECREMENTO -1-----
98
99      when st_Down0a =>
100         rst_cnt <= '0';
101         en_cnt <= '1';
102         if(cnt_100ms = '1') then
103             n_state <= st_Down0b;
104         else
105             n_state <= st_Down0a;
106         end if;
107     when st_Down0b =>
```

```
107          en_b <= '1';
108          sel <= "01";
109          n_state <= st_Main;
110
111      -----DECREMENTO -10-----
112
113      when st_Down1a =>
114          rst_cnt <= '0';
115          en_cnt <= '1';
116          if(cnt_100ms = '1') then
117              n_state <= st_Down1b;
118          else
119              n_state <= st_Down1a;
120          end if;
121      when st_Down1b =>
122          en_b <= '1';
123          sel <= "11";
124          n_state <= st_Main;
125
126      end case;
127  end if;
128 end process;
129
130
131 end Behavioral;
132
133
```

```
1 -----  
2 --Contador que genera un pulso cada 100ms y esta habilitado por una señal de enable  
3 -----  
4 library IEEE;  
5 use IEEE.STD_LOGIC_ARITH.ALL;  
6 use IEEE.STD_LOGIC_UNSIGNED.ALL;  
7 use IEEE.STD_LOGIC_1164.ALL;  
8  
9  
10 entity Contador_100ms is  
11     Port ( clk : in STD_LOGIC;           --Reloj del sistema  
12             rst_cnt : in STD_LOGIC;        -- Reset del contador  
13             en_cnt : in STD_LOGIC;         -- Enable del contador  
14             cnt_100ms : out STD_LOGIC);    -- Fin de cuenta  
15 end Contador_100ms;  
16  
17 architecture Behavioral of Contador_100ms is  
18  
19 signal contador : STD_LOGIC_VECTOR(22 downto 0):="000000000000000000000000"; --  
declaracion de la señal.  
20 --22 down to 0 ya que son 5M y nos hacen falta 2^23  
21 --numeros binarios para poder representar 5M que son numeros de pulsos de reloj  
22 --necesarios para contar 100ms  
23 begin  
24  
25 process (clk)  
26 begin  
27     if clk'event and clk='1' then  
28         cnt_100ms <= '0';  
29         if en_cnt='1' then  
30             contador <= contador + '1';  
31         end if;  
32         if contador >= "10011000100101101000000" then  
33             -- valor máximo de la cuenta en  
34             cnt_100ms <= '1';  
35             contador <= (others => '0');  
36         end if;  
37         if rst_cnt='1' then  
38             -- en caso de tener reset ponemos el  
39             contador a '0' y ponemos el contador de 100 a cero tambien.  
40             cnt_100ms <= '0';  
41             contador <= (others => '0');  
42         end if;  
43     end if;  
44 end process;  
45  
46 end Behavioral;
```

```

1  -----
2  --REG_b
3  --Este registro contiene en binario el valor del código introducido por el usuario, b.
4  En su
5  --funcionalidad se incorpora también que el registro sea capaz de variar su valor (en
6  función de lo que
7  --indiquen las señales que provienen de la máquina de estados de Moore), y que no se
8  excedan los
9  --límites máximos y mínimos permitidos (0-63)
10 -----
11 library IEEE;
12 use IEEE.STD_LOGIC_ARITH.ALL;
13 use IEEE.STD_LOGIC_UNSIGNED.ALL;
14 use IEEE.STD_LOGIC_1164.ALL;
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
entity Reg_b is
    Port ( clk : in STD_LOGIC;                      --Reloj del sistema
           rst_b : in STD_LOGIC;                     -- Rst de b
           en_b : in STD_LOGIC;                      -- Enable de b
           sel : in STD_LOGIC_VECTOR(1 downto 0);    -- Señal de control de suma o resta
           sw6: in STD_LOGIC;                        -- Modo volumen o contraseña
           v : out STD_LOGIC_VECTOR(3 downto 0);     -- Valor almacenado en Volumen
           b : out STD_LOGIC_VECTOR(5 downto 0));    -- Valor de la contraseña almacenado
end Reg_b;
architecture Behavioral of Reg_b is
signal s_Reg_b : STD_LOGIC_VECTOR(5 downto 0); -- señal auxiliar para poder usar
correctamente el b.
signal s_Reg_v : STD_LOGIC_VECTOR(3 downto 0); -- señal auxiliar para poder usar
correctamente el volumen.
begin
process (rst_b, clk,s_Reg_b,sel) --Proceso que actualiza el valor de b y de v segun lo
que se ha pulsado
begin
    if (rst_b = '1') then s_Reg_b <= "000000"; s_Reg_v <= "0101";                    --
Asignación del valor por defecto con el Reset
    elsif clk'event and clk='1' then
        if en_b = '1' then
            if sw6 = '0' then
                if ((sel = "00") and (s_Reg_b < "111111")) then s_Reg_b <= s_Reg_b + "01";
                -- Incremento +1 de la señal
                end if;
                -- Ponemos
                "111111" ya que es la maximo numero(63) no podemos sumar mas de este numero
                if ((sel = "01") and (s_Reg_b > "000000")) then s_Reg_b <= s_Reg_b - "01";
                -- Decremento -1 de la señal
                end if;
                -- Ponemos
                "000000" ya que es el minimo valor es 0
                if ((sel = "10") and (s_Reg_b < "110110")) then s_Reg_b <= s_Reg_b + "01010";
                -- Incremento +10 de la señal
                end if;
                --
Ponemos "110110" que es el numero 54 y no podremos sumar 10 a ese numero
                if ((sel = "11") and (s_Reg_b > "001001")) then s_Reg_b <= s_Reg_b - "01010";
                -- Decremento -10 de la señal

```

```
42      end if;                                         -- Ponemos
        "001001" que es 9 como valor minimo para no restar y sobrepasar el 0.
43      elsif sw6 ='1' then
44          if ((sel = "00") and (s_Reg_v < "1001")) then s_Reg_v <= s_Reg_v + "01";
          -- Incremento +1 de la señal
45      end if;                                         -- Ponemos
        "111111" ya que es la maximo numero(63) no podemos sumar mas de este numero
46      if ((sel = "01") and (s_Reg_v > "0001")) then s_Reg_v <= s_Reg_v - "01";
          -- Decremento -1 de la señal
47      end if;
48      end if;
49      end if;
50  end if;
51  b<=s_Reg_b;
52  v<=s_Reg_v;
53 end process;
54
55
56 end Behavioral;
57
58
```

```

1  -----
2  -- Bloque Control Display
3  -- Recibe la señal b que es el estado en el que estan los displays segun lo pulsado en
4  Pulsadores
5  -- para posteriormente traducirlo con un conversor BCD y sacarlo por los displays.
6  -- Tambien recibe la señal v con el valor del volumen
7  -----
8  library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
10
11 entity control_displays is
12     Port ( clk : in STD_LOGIC;                                --Reloj del sistema
13            v : in STD_LOGIC_VECTOR(3 downto 0);                -- Volumen del sistema
14            b : in STD_LOGIC_VECTOR (5 downto 0);                --Clave pulsada
15            sw6: in STD_LOGIC;                                  -- Modo. Volumen o encriptacion
16            Disp0 : out STD_LOGIC;                             --Enable display 0
17            Disp1 : out STD_LOGIC;                             --Enable display 1
18            Disp2 : out STD_LOGIC;                             --Enable display 2
19            Disp3 : out STD_LOGIC;                             --Enable display 3
20            SDP : out STD_LOGIC;                            --Salida Punto Decimal
21            Seg7 : out STD_LOGIC_VECTOR (0 to 6));           --salida que se encarga de encender
22   los segmentos de cada display
23 end control_displays;
24
25 architecture Behavioral of control_displays is
26   -- variables auxiliares
27   signal s_Digito0 :STD_LOGIC_VECTOR (3 downto 0);
28   signal s_Digito1 :STD_LOGIC_VECTOR (3 downto 0);
29   signal s_Digito2 :STD_LOGIC_VECTOR (3 downto 0);
30   signal s_Digito3 :STD_LOGIC_VECTOR (3 downto 0);
31   signal s_cnt_5ms :STD_LOGIC;
32
33 Component visualizacion is
34     Port ( clk : in STD_LOGIC;
35            cnt_5ms : in STD_LOGIC;
36            Dígito0 : in STD_LOGIC_VECTOR(3 downto 0);
37            Dígito1 : in STD_LOGIC_VECTOR(3 downto 0);
38            Dígito2 : in STD_LOGIC_VECTOR(3 downto 0);
39            Dígito3 : in STD_LOGIC_VECTOR(3 downto 0);
40            sw6: in STD_LOGIC;
41            Disp0 : out STD_LOGIC;
42            Disp1 : out STD_LOGIC;
43            Disp2 : out STD_LOGIC;
44            Disp3 : out STD_LOGIC;
45            SDP : out STD_LOGIC;
46            Seg7 : out STD_LOGIC_VECTOR(0 to 6));
47
48 end component;
49
50 Component conversorBCD is
51     Port ( clk : in STD_LOGIC;
52            b : in STD_LOGIC_VECTOR(5 downto 0);
53            v : in STD_LOGIC_VECTOR(3 downto 0);
54            sw6: in STD_LOGIC;
55            D0 : out STD_LOGIC_VECTOR(3 downto 0);
56            D1 : out STD_LOGIC_VECTOR(3 downto 0);

```

```
57           D2 : out STD_LOGIC_VECTOR(3 downto 0);
58           D3 : out STD_LOGIC_VECTOR(3 downto 0));
59 end component;
60
61 Component contador_5ms is
62     Port ( clk : in STD_LOGIC;
63             cnt_5ms : out STD_LOGIC);
64 end component;
65
66
67
68 begin
69 U1: visualizacion
70 port map(
71     clk=>clk,
72     cnt_5ms=>s_cnt_5ms,
73     Dígito0=>s_Dígito0,
74     Dígito1=>s_Dígito1,
75     Dígito2=>s_Dígito2,
76     Dígito3=>s_Dígito3,
77     Disp0=>Disp0,
78     Disp1=>Disp1,
79     Disp2=>Disp2,
80     Disp3=>Disp3,
81     SDP => SDP,
82     sw6 => sw6,
83     Seg7=>Seg7);
84
85 U2:conversorBCD
86 port map(
87     clk=>clk,
88     b=>b,
89     v => v,
90     sw6 => sw6,
91     D0=>s_Dígito0,
92     D1=>s_Dígito1,
93     D2=>s_Dígito2,
94     D3=>s_Dígito3);
95
96 U3: contador_5ms
97 port map(
98     clk=>clk,
99     cnt_5ms=>s_cnt_5ms);
100
101 end Behavioral;
102
103
```

```

1
2 --Bloque de Visualizacion cuya finalidad es la recibir la salida del BCD de los
3 digitos que se han interpretado a partir de la señal b
4 -- y sacar su correspondiente representacion a los displays mediante seg7 y
5 activandolos mediante Disp
6 -----
7 library IEEE;
8 use IEEE.STD_LOGIC_1164.ALL;
9 use IEEE.STD_LOGIC_ARITH.ALL;
10 use IEEE.STD_LOGIC_UNSIGNED.ALL;
11
12 entity visualizacion is
13     Port ( clk : in STD_LOGIC;                                -- Reloj de sistema
14             cnt_5ms : in STD_LOGIC;                            -- Cuenta de 5ms
15             Dígito0 : in STD_LOGIC_VECTOR(3 downto 0);-- Valor del BCD para su
16             Dígito1 : in STD_LOGIC_VECTOR(3 downto 0);-- Valor del BCD para su
17             Dígito2 : in STD_LOGIC_VECTOR(3 downto 0);-- Valor del BCD para su
18             Dígito3 : in STD_LOGIC_VECTOR(3 downto 0);-- Valor del BCD para su
19             sw6: in STD_LOGIC;                                --Modo volumen o encriptacion
20             Disp0 : out STD_LOGIC;                           -- Enable display
21             Disp1 : out STD_LOGIC;                           -- Enable display
22             Disp2 : out STD_LOGIC;                           -- Enable display
23             SDP : out STD_LOGIC;                            --Salida Punto Decimal
24             Disp3 : out STD_LOGIC;                           -- Enable display
25             Seg7 : out STD_LOGIC_VECTOR(0 to 6));          -- Segmentos del display
26
27             --NOTA: los numeros van de G,F,E,D,C,B,A POR QUE LO TENEMOS EN 0 TO 6
28             ADEMÁS SON ACTIVAS A NIVEL BAJO
29 end visualizacion;
30
31 architecture Behavioral of visualizacion is
32
33 signal sel : STD_LOGIC_VECTOR (1 downto 0) := "00";
34 signal dígitoBCD : STD_LOGIC_VECTOR (3 downto 0):= "0000";
35
36 begin
37
38 begin
39     if clk'event and clk='1' then --proceso que cambia la señal de valor cada 5 ms
40         if(cnt_5ms = '1') then      --pasa de 00-01-10-11-00....
41             sel<=sel+"01";
42         else
43             sel<= sel;
44         end if;
45     end if;
46 end process;
47
48 process (sel, dígito0, dígito1, dígito2, dígito3) --este proceso cambia de estado cada
49             5ms valor imperceptible por el ojo humano luego vemos
50 begin
51             --todo el rato los displays
52             encendidos aunque la realidad es que se apagan.

```

```
50
51      -- Valores por defecto
52      digitoBCD <= digito0;
53      Disp0 <= '1'; Disp1 <= '1'; Disp2 <= '1'; Disp3 <= '1'; SDP <= '1';
54
55      CASE sel IS
56          when "00" =>
57              digitoBCD <= digito0;
58              Disp0 <= '0';
59              Disp1 <= '1';
60              Disp2 <= '1';
61              Disp3 <= '1';
62          if sw6= '0' THEN
63              SDP <= '0';
64          end if;
65          when "01" =>
66              digitoBCD <= digito1;
67              Disp0 <= '1';
68              Disp1 <= '0';
69              Disp2 <= '1';
70              Disp3 <= '1';
71          SDP <= '1';
72          when "10" =>
73              digitoBCD <= digito2;
74              Disp0 <= '1';
75              Disp1 <= '1';
76              Disp2 <= '1';
77              Disp3 <= '1';
78          SDP <= '1';
79          when "11" =>
80              digitoBCD <= digito3;
81              Disp0 <= '1';
82              Disp1 <= '1';
83              Disp2 <= '1';
84              Disp3 <= '0';
85          if sw6= '1' THEN
86              SDP <= '0';
87          end if;
88          when others =>
89              Disp0 <= '1';
90              Disp1 <= '1';
91              Disp2 <= '1';
92              Disp3 <= '1';
93          SDP <= '1';
94          digitoBCD <= digito0;
95
96      END CASE;
97  end process;
98
99  with DigitoBCD select Seg7 <=
100    "1000000" when "0000", ---los numero van de G,F,E,D,C,B,A PORQUE LO TENEMOS EN 0 TO 6
ADEMAS SON ACTIVAS A NIVEL BAJO
101    "1111001" when "0001",
102    "0100100" when "0010",
103    "0110000" when "0011",
104    "0011001" when "0100",
105    "0010010" when "0101",
106    "0000010" when "0110",
```

```
107      "1111000" when "0111",
108      "0000000" when "1000",
109      "0011000" when "1001",
110      "1111111" when others;
111
112
113
114  end Behavioral;
115
116
```

```

1  -- Bloque que se encarga de recibir la señal b que viene de pulsadores en funcion de
2  -- lo pulsado
3  -- que es interpretada mediante unas divisiones para obtener asi de un numero en
4  -- binario su correspondiente representacion
5  -- en unidades (D0) y decenas (D1) el resto de Digitos no los usamos de momento en
6  -- nuestra implementacion.
7  -----
8  library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
10 use IEEE.STD_LOGIC_ARITH.ALL;
11 use IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

conversorBCD.vhd

```
50      if b >= conv_std_logic_vector(30, 6) then
51          D1 <= "0011";
52          resto := b - conv_std_logic_vector(30, 6); -- resto = data_in - 30
53      end if;
54
55      if b >= conv_std_logic_vector(40, 6) then
56          D1 <= "0100";
57          resto := b - conv_std_logic_vector(40, 6); -- resto = data_in - 40
58      end if;
59
60      if b >= conv_std_logic_vector(50, 6) then
61          D1 <= "0101";
62          resto := b - conv_std_logic_vector(50, 6); -- resto = data_in - 50
63      end if;
64
65      if b >= conv_std_logic_vector(60, 6) then
66          D1 <= "0110";
67          resto := b - conv_std_logic_vector(60, 6); -- resto = data_in - 60
68      end if;
69
70      D0 <= resto(3 downto 0); -- Asignacion de salida
71  end if;
72 end process;
73
74 end Behavioral;
75
76
```

```
1  -- Contador que genera un pulso cada 5ms
2
3  -----
4  library IEEE;
5  use IEEE.STD_LOGIC_ARITH.ALL;
6  use IEEE.STD_LOGIC_UNSIGNED.ALL;
7  use IEEE.STD_LOGIC_1164.ALL;
8
9
10 entity contador_5ms is
11     Port ( clk : in STD_LOGIC;
12             cnt_5ms : out STD_LOGIC);
13 end contador_5ms;
14
15 architecture Behavioral of contador_5ms is
16
17 signal contador : STD_LOGIC_VECTOR(17 downto 0) := "00000000000000000000";
18
19 begin
20
21 process (clk)
22 begin
23     if clk'event and clk='1' then
24         cnt_5ms <= '0';
25         contador <= contador + '1';
26         if contador >= "111101000010010000" then          -- Valor máximo de la cuenta.
27             cnt_5ms <= '1';
28             contador <= (others => '0');
29         end if;
30     end if;
31
32 end process;
33
34
35 end Behavioral;
36
37
```

```

1 # Reloj principal del sistema
2 NET "CLK" LOC = "M6";                      # Señal de reloj del sistema
3 # Conexiones de los PULSADORES
4 NET "Up0" LOC = "G12";                      # Pulsador 0 --Con este sumo 1
5 NET "Down0" LOC = "C11";                     # Pulsador 1 --Con este resto 1
6 NET "Up1" LOC = "M4";                        # Pulsador 2 --con este sumo 10
7 NET "Down1" LOC = "A7";                      # Pulsador 3 --con este resto 10
8 #NET "Reset" LOC = "G1";                     #RESET
9 ## Conexiones de los INTERRUPTORES
10 #NET "SW0" LOC = "P11";                     # Interruptor 0 (SW0)
11 # Conexiones de los DISPLAYS
12 NET "SDP" LOC = "N13";                      # señal = DP
13 NET "SEG7<6>" LOC = "L14";                 # señal = CA
14 NET "SEG7<5>" LOC = "H12";                 # Señal = CB
15 NET "SEG7<4>" LOC = "N14";                 # Señal = CC
16 NET "SEG7<3>" LOC = "N11";                 # Señal = CD
17 NET "SEG7<2>" LOC = "P12";                 # Señal = CE
18 NET "SEG7<1>" LOC = "L13";                 # Señal = CF
19 NET "SEG7<0>" LOC = "M12";                 # Señal = CG
20 # Señales de activación de los displays
21 NET "DISP0" LOC = "F12";                    # Activación del display 0 = AN0, activa a nivel bajo
22 NET "DISP1" LOC = "J12";                    # Activación del display 1 = AN1, activa a nivel bajo
23 NET "DISP2" LOC = "M13";                    # Activación del display 2 = AN2, activa a nivel bajo
24 NET "DISP3" LOC = "K14";                    # Activación del display 3 = AN3, activa a nivel bajo
25 ## Salidas digitales externas
26 NET "DIN" LOC = "A13";                     # Salida Digital0
27 NET "SSCLK" LOC = "C12";                   # Salida Digital1
28 NET "CS0" LOC = "C13";                     # Salida Digital2
29 NET "CS1" LOC = "D12";                     # Salida Digital3
30 #NET "OUT_DIG4" LOC = "N5";                # Salida Digital4
31 #NET "OUT_DIG5" LOC = "N4";                # Salida Digital5
32 #NET "OUT_DIG6" LOC = "P4";                # Salida Digital6
33 #NET "OUT_DIG7" LOC = "G1";                # Salida Digital7
34 #SWITCHES
35 NET "SW<0>" LOC = "P11";                 #señal del swicth 1 para la encriptacion la sw
36 NET "SW<1>" LOC = "L3";                  #señal del swicth 2 para la encriptacion la sw
37 NET "SW<2>" LOC = "K3";                  #señal del swicth 3 para la encriptacion la sw
38 NET "SW<3>" LOC = "B4";                  #señal del swicth 4 para la encriptacion la sw
39 NET "SW<4>" LOC = "G3";                  #señal del swicth 5 para la encriptacion la sw
40 NET "SW<5>" LOC = "F3";                  #señal del swicth 6 para la encriptacion la sw
41 NET "SW6" LOC = "E2";                     #señal del swicth 6 para la encriptacion la sw
42 NET "Reset" LOC = "N3";                   #RESET que se asocia con el sw7
43 #ENTRADA DIGITAL
44 NET "DOUT" LOC = "C6";
45
46

```