

PRÁCTICA 5

DISEÑO E IMPLEMENTACIÓN DE UNA UNIDAD DE ACOPLAMIENTO SERIE PARA UN CONVERTIDOR D/A

Palabras clave: Analizador Lógico, DAC (*Digital to Analog Converter*), VHDL, descripción estructural, FPGA, SPI (*Serial Peripheral Interface Bus*).

1. INTRODUCCIÓN

Después de realizar la práctica en la que se capturó una señal analógica con un circuito ADC acoplado a la FPGA a través del bus SPI, ahora se realizará el proceso inverso, es decir, se reconstruirá una señal analógica a partir de una digital. Esta conversión de señales digitales en analógicas resulta útil para trasladar información desde los sistemas de procesado, típicamente digitales, a etapas de actuación, típicamente analógicas. En esta práctica esta tarea se lleva a cabo con un dispositivo convertidor analógico-digital o DAC conectado a la FPGA a través de una interfaz SPI, para lo cual es necesario realizar un controlador en la FPGA que implemente el protocolo del bus SPI, de forma similar a como se hizo en la práctica anterior para el ADC.

El circuito convertidor que se utiliza es el MCP4911, tiene una resolución de 10 bits y está disponible con un encapsulado DIP8.

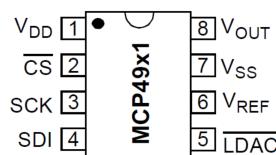


Figura 1. Distribución de terminales del DAC.

El dato digital se establece con los 10 interruptores disponibles en la placa DE0, el reset se realiza cuando se pulsa el botón 1 y la conversión (transmisión del dato en formato serie al DAC) se inicia cuando se pulsa el botón 3. Cuando finaliza una conversión se indica poniendo a nivel alto una señal de fin de conversión.

2. OBJETIVOS DE APRENDIZAJE

- 1.- Resolver problemas de sistemas electrónicos digitales complejos reutilizando módulos ya probados.
- 2.- Saber implementar circuitos secuenciales síncronos.
- 3.- Realizar descripciones estructurales y funcionales en VHDL de circuitos secuenciales sencillos.
- 4.- Realizar sistemas diseñados con dos niveles de jerarquía.
- 5.- Saber simular componentes y sistemas.
- 6.- Utilizar recursos periféricos conectados a una FPGA. Realización del mapa de entradas y salidas. Asignación de pines de la FPGA.
- 7.- Conocer el funcionamiento del bus SPI.
- 8.- Conocer el funcionamiento de los circuitos DAC.
- 9.- Saber realizar prototipos de sistemas electrónicos digitales.

3. TAREAS PREVIAS

Para una preparación adecuada de la práctica, además de la lectura detallada de este enunciado, es conveniente que el alumno realice un proyecto con los módulos fuente proporcionados para el controlador spi_dac antes de la sesión de laboratorio.

4. ESPECIFICACIONES DE LA UNIDAD DE CONTROL DE LA CONVERSIÓN A/D

La unidad de control que se debe implementar en la FPGA debe disponer de la siguiente interfaz de entrada-salida y debe implementar el protocolo que se especifica a través del cronograma de la Figura 2.

Señales de entrada:

din[9..0]: Dato de 10 bits a convertir establecido con los interruptores SW9-0.

clk: Señal de reloj principal de 50 MHz.

reset: Inicialización. Cuando está a nivel bajo se inicializa el sistema (botón 1 pulsado).

sc: Cuando hay un flanco descendente en esta señal (se pulsa el botón 2) se debe comenzar una conversión.

Señales de salida:

cs: Selección del DAC.

sck: Señal de sincronismo y temporización del DAC.

sdo: Dato serie de salida de la unidad de control al DAC.

eoc: Indicación de finalización de la conversión o, en este caso, de la transmisión serie.

El controlador del DAC recibe los datos de entrada de 10 bits (**din**) desde los interruptores y, cuando se pulsa el botón 2 (**sc**), se memoriza el dato y se generan las señales de control del DAC (**cs**, **sck**, **sdo**) de acuerdo con la Figura 2. En total el dato completo a transmitir es de 16 bits formado por 4 bits de configuración iniciales, los 10 bits del dato a convertir, y 2 bits de relleno. En este caso, el primer bit de configuración es 0 porque se desea escribir en el registro del DAC, y los demás están a '1' para que la salida se conecte a través de un *buffer* de corriente, la ganancia sea unitaria y la salida esté activa (ver la página 24 de las hojas de características del DAC).

Es necesario tener en cuenta que el reloj de 10 MHz para el funcionamiento interno del controlador y del propio DAC (señal **sck**) se debe obtener a partir del reloj base de la placa DE0, que es de 50 MHz. Para ello, en esta práctica se utiliza un divisor de reloj implementado con los recursos lógicos de la FPGA como un módulo dentro del esquema del controlador (**divisor_reloj** en Figura 3).

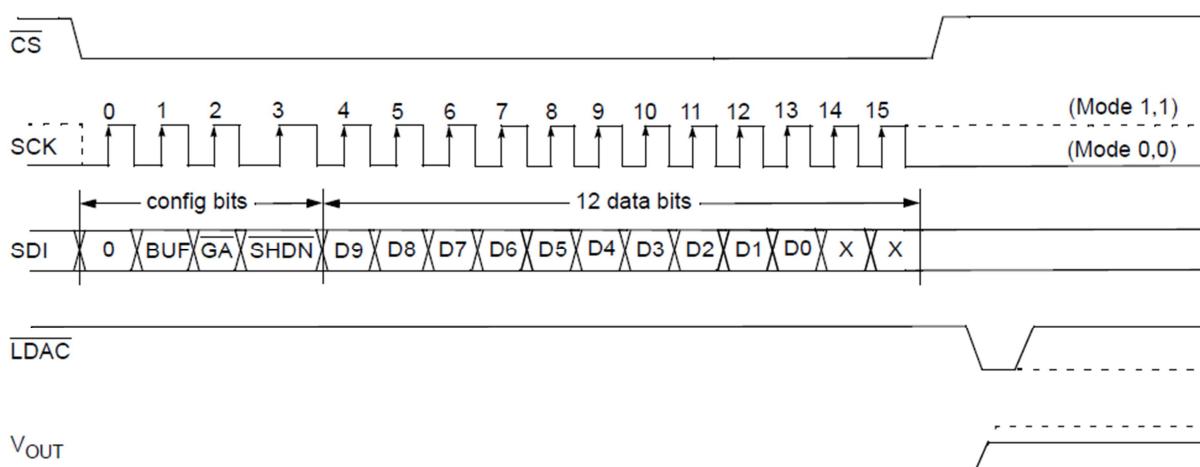


Figura 2. Cronograma de funcionamiento del DAC.

5. DESARROLLO DE LA PRÁCTICA

Tarea 1: Diseño e implementación de un módulo de control SPI para conexión a un convertidor D/A.

En este apartado se presenta una solución ESTRUCTURAL (frente a la solución ALGORITMICA de la práctica anterior) para la unidad de control SPI **spi_dac** que debe ser estudiada y probada, en primera instancia, mediante simulación.

Se debe crear un proyecto de Quartus con los módulos fuente proporcionados que se comentan a continuación, compilar el diseño y asignar los terminales de la FPGA según se indica en la Tabla 1. Finalmente se debe realizar la simulación RTL según los valores de prueba de la Figura 5.

En la Figura 3 se presenta el diagrama de bloques del diseño estructural proporcionado por el visor RTL del Quartus, es decir, que está formado por la interconexión de módulos más elementales, casi todos de propósito general, que se describen a continuación:

- 1.- **divisor_reloj.vhd**: a partir del reloj de 50 MHz de la placa DE0 genera el reloj interno de 10 MHz necesario, según las especificaciones, para el control del DAC.
- 2.- **impulso_ini.vhd**: sincroniza las entradas asíncronas. La entrada **sc** se sincroniza con el reloj interno del sistema a través de un detector de flanco que genera un pulso de un ciclo de reloj de duración tras la detección de un flanco en dicha entrada.
- 3.- **reg_des.vhd**: es un registro de desplazamiento con carga en paralelo que se encarga de memorizar y serializar el dato de entrada.
- 4.- **contador.vhd**: cuenta el número de ciclos de reloj durante la fase de transferencia. Como tiene que contar 16 ciclos es de 4 bits. Además, genera la señal de fin de conteo.
- 5.- **uc_spi_out.vhd**: es el cerebro del controlador. Se implementa mediante una máquina de estados que sincroniza a la salida la información generada por cada componente.

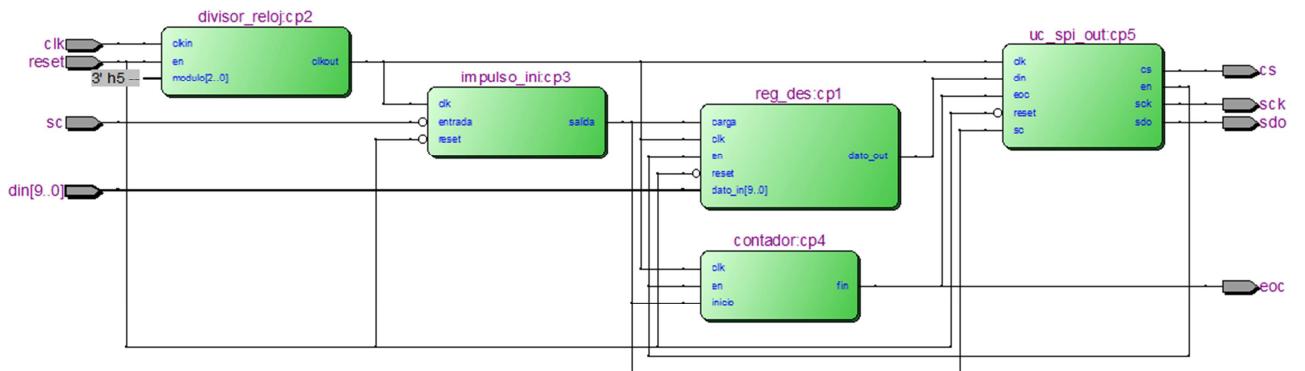


Figura 3. Esquema de la estructura del controlador del DAC.

A continuación, en la Figura 4 se muestra el resultado de la simulación. Cuando se pulsa el botón 2 se pone a cero la señal **sc** y comienza el proceso de transmisión seleccionando el periférico. Después se deja pasar un tiempo de 100 ns y se envían los bits: primero los correspondientes a los cuatro bits de control en este caso “0111”, y después los 10 del dato. Finalmente se envían los dos bits extra para completar la transmisión de 16 bits. Al acabar la transmisión se pone a ‘1’ la señal de selección **cs**. El reloj **sck** del bus solamente se activa durante la transmisión.

Proyecto 2

Práctica 5: Diseño e implementación de una unidad de acoplamiento serie para un convertidor D/A.

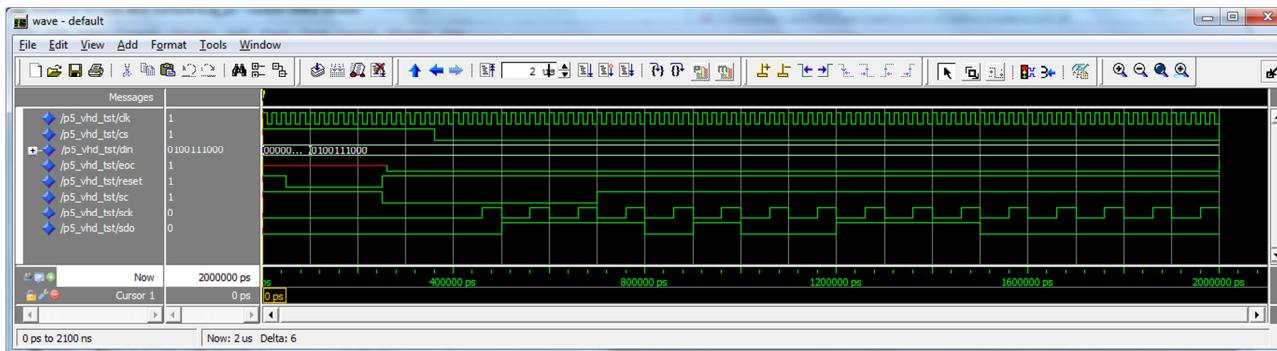


Figura 4. Simulación del controlador del DAC.

```

46  dato_entrada: PROCESS
47  BEGIN
48    din<="0000000000";  WAIT FOR 100 ns;
49    din<="0100111000";  WAIT;
50  END PROCESS dato_entrada;
51
52  inicio: PROCESS
53  BEGIN
54    reset<='1';  WAIT FOR 50 ns;
55    reset<='0';  WAIT FOR 200 ns;
56    reset<='1';  WAIT;
57  END PROCESS inicio;
58
59  ini_conversion: PROCESS
60  BEGIN
61    sc<='1';  WAIT FOR 250 ns;
62    sc<='0';  WAIT FOR 450 ns;
63    sc<='1';  WAIT;
64  END PROCESS ini_conversion;
65
66  reloj: PROCESS
67  BEGIN
68    clk<='0';
69    FOR i IN 0 TO 120 LOOP
70      clk <= '1'; wait for 10 ns;
71      clk <= '0'; wait for 10 ns;
72    END LOOP;
73    WAIT;
74  END PROCESS reloj;
75  END spi_dac_arch;

```

Figura 5. Vectores de prueba para la simulación del controlador del DAC.

Tabla 1. Listado de la asignación de terminales de la FPGA obtenido con la herramienta *Pin Planner* de Altera.

Node Name	Direction	Location	I/O Bank	VREF Group
cs	Output	PIN_AA16	4	B4_N1
eoc	Output	PIN_J1	1	B1_N1
sck	Output	PIN_AB16	4	B4_N1
sdo	Output	PIN_AA15	4	B4_N1
clk	Input	PIN_G21	6	B6_N1
din[0]	Input	PIN_J6	1	B1_N0
din[1]	Input	PIN_H5	1	B1_N0
din[2]	Input	PIN_H6	1	B1_N0
din[3]	Input	PIN_G4	1	B1_N0
din[4]	Input	PIN_G5	1	B1_N0
din[5]	Input	PIN_J7	1	B1_N1
din[6]	Input	PIN_H7	1	B1_N0
din[7]	Input	PIN_E3	1	B1_N0
din[8]	Input	PIN_E4	1	B1_N0
din[9]	Input	PIN_D2	1	B1_N0
reset	Input	PIN_G3	1	B1_N0
sc	Input	PIN_F1	1	B1_N0

En el ANEXO 1 se incluye el modelado de cada uno de los subcircuitos que componen el controlador del DAC:

Tarea 2: Generación de una señal analógica a partir de un dato digital establecido con los interruptores externos conectados a la FPGA.

Después de preparar el diseño y estudiar su funcionamiento se debe probar montando el hardware, para lo cual se utilizará la placa DE0 y la placa de prototipos donde se instala el DAC.

Se comienza por conectar en la placa de prototipos el DAC y realizar las interconexiones con la placa DE0. Proporcionar al dispositivo la señal de alimentación (VDD) correspondiente a los 3.3 V de la placa DE0 y conectar la tensión de referencia (VREF) a este mismo valor. Elegir los puertos de conexión adecuados en la FPGA para la entidad **spi_dac**.

La señal de entrada digital que se desea convertir se debe conectar, bit a bit, a los interruptores de la placa DE0. La salida **eoc** se debe conectar a uno de los LEDs. Y la entrada de inicio de conversión a un botón. Para el cableado utilizar la siguiente tabla de asignación:

Tabla 2. Interconexiones entre la FPGA y el DAC.

Patilla DAC	Denominación de la patilla en el MCP4911	Conector Expansión DE0	Patilla Conector DE0 (J4)	Pin FPGA
1 y 6	VDD y VREF Alimentación y referencia	GPIO1	29	
2	CS	GPIO0	4	PIN_AA16
3	SCK	GPIO0	2	PIN_AB16
4	SDI	GPIO0	5	PIN_AA15
5 y 7	Referencia 0 V y LDAC	GPIO1	30	--
8	Salida analógica	Conectar al polímetro o al osciloscopio y medir la tensión Vdc.		

Una vez conectado todo el sistema, se debe probar de la siguiente manera. Configurar los interruptores con la combinación binaria que se utilizó en la simulación “0100111000” y pulsar el botón 3 para realizar la conversión.

Comprobar que se enciende el LED0 indicando que se realizó la conversión. Tomar la medida de la tensión de salida, anotarla en la Tabla 3, y comprobar que se corresponde con el valor esperado de acuerdo a la ecuación 4-1 de las hojas de características del DAC.

Para una mayor exactitud, no tomar el valor nominal de 3.3 V para la tensión de referencia, si no medirlo también con la instrumentación. Completar la Tabla 4 de resultados con otros valores digitales de entrada.

Tabla 3. Valor de la tensión de referencia (tensión en la patilla 8 del DAC).

VREF [mV]	Vsalida [mV]

Tabla 4. Tabla de resultados.

Muestra nº	Combinación binaria de entrada	Tensión de salida Vdc	Valor teórico de la tensión de salida	Error
1	“0000000000”			
2	“0000000001”			
3	“1111111111”			
4	“1111111110”			

Tarea 3: Utilización del Osciloscopio para monitorizar el puerto SPI.

Una vez probado el funcionamiento se debe conectar el osciloscopio para capturar una trama del bus. En la siguiente figura se muestra una trama correspondiente a la conversión del dato “0100111000”, que es el mismo que se utilizó en la simulación. Comprobar que el resultado es el mismo y probar con otros valores. Se puede observar también la configuración de la señal de disparo (TRIGGER) que activa el muestreo, en este caso la señal **cs**.

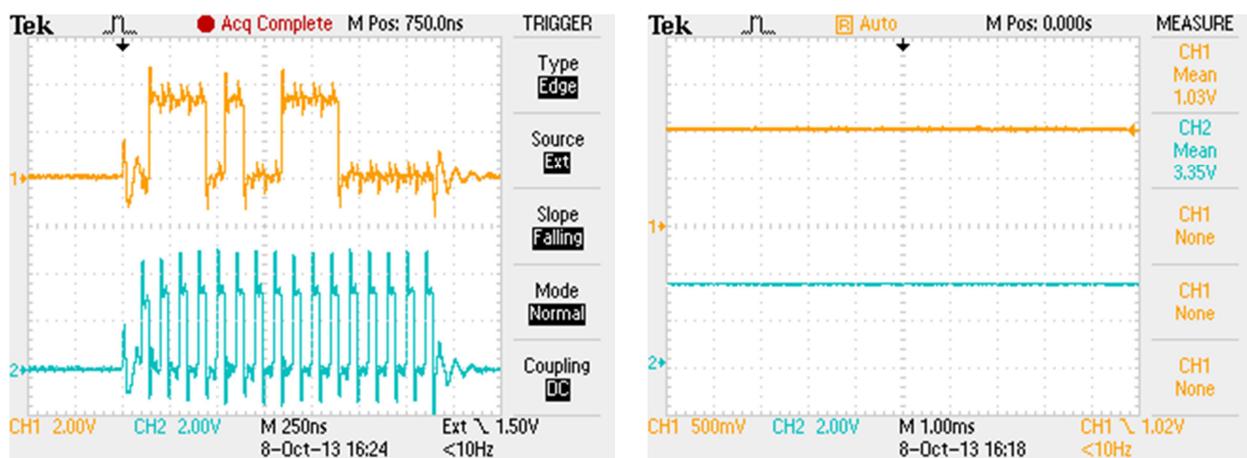


Figura 6. Oscilogramas con las señales SDI y SCK del bus SPI del DAC (izquierda), y (derecha) tensión eléctrica de salida del DAC (CH1) y tensión de referencia (CH2).

6. EVALUACIÓN

- 1.- Realización del proyecto en Quartus completo (Tarea 1).
- 3.- Simulación (Tarea 1).
- 3.- Montaje (Tarea 2).
- 4.- Tabla de valores (Tarea 2).
- 5.- Captura de valores en osciloscopio (Tarea 3).

calificación

ANEXO I:

MODELADO DE LOS COMPONENTES DEL CONTROLADOR DEL DAC.

```

1  -----DIVISOR DE FRECUENCIA-----
2  --Entidad que genera una señal clkout de un bit de frecuencia f_out,
3  --a partir de una señal de entrada clkin de frecuencia f_in,
4  --de modo que:
5  --si modulo es cero o uno, entonces f_out=f_in,
6  --y si modulo es mayor que 1, entonces f_out=f_in/modulo.
7  --En donde modulo es el dato de entrada que representa el valor
8  --por el cual se divide la frecuencia f_in.
9
10 LIBRARY IEEE;
11 USE IEEE.std_logic_1164.all;
12 USE IEEE.numeric_std.all;
13
14 ENTITY divisor_reloj IS
15  --el modulo del divisor debe ser menor que (2^n)-1
16  --si n es 16, entonces el modulo maximo sera 65535
17  GENERIC(n: INTEGER:=3);
18  PORT( clkin: IN std_logic;
19        clkout: OUT std_logic;
20        en: IN std_logic;
21        modulo: IN std_logic_vector(n-1 downto 0)
22        );
23 END divisor_reloj;
24
25 ARCHITECTURE descripcion OF divisor_reloj IS
26  --contador es la variable que se incrementara
27  SIGNAL contador: INTEGER RANGE 0 TO (2**n)-1;
28
29  --se utiliza una señal que representa la mitad de la division
30  SIGNAL mitad_modulo: std_logic_vector((n-1) DOWNTO 0);
31
32  --señales de tipo entero con el valor del divisor y su mitad
33  SIGNAL divisor, mitad_divisor:INTEGER RANGE 0 TO (2**n)-1;
34
35 BEGIN
36  --se rota a la derecha de forma que quede dividido a la mitad
37  mitad_modulo((n-2) DOWNTO 0)<=modulo((n-1) DOWNTO 1);
38
39  --se pone a cero el bit más significativo
40  mitad_modulo(n-1)<='0';
41  --se pasan a datos de tipo entero para compararse con contador
42  divisor<=TO_INTEGER(UNSIGNED(modulo));
43  mitad_divisor<=TO_INTEGER(UNSIGNED(mitad_modulo));
44
45 PROCESS(clkin,divisor,mitad_divisor,en)
46 BEGIN
47  IF en='0' THEN
48    clkout<='0';
49    contador<=1;
50  ELSE
51    IF (divisor <= 1) THEN
52      clkout<=clkin;
53      contador<=1;
54    ELSE
55      IF(clkin='1' AND clkin'EVENT) THEN
56        IF(contador<=mitad_divisor) THEN
57          clkout<='1';
58          contador<=contador+1;
59        ELSE IF(contador>mitad_divisor AND contador<divisor) THEN
60          clkout<='0';
61          contador<=contador+1;
62        ELSE
63          contador<=1;
64          clkout<='0';
65        END IF;
66      END IF;
67    END IF;
68  END IF;
69  END PROCESS;
70 END descripcion;
71

```

Figura 7. Módulo **divisor_reloj.vhd**. Este módulo divide la frecuencia del reloj de entrada por un factor dado por la otra entrada **modulo**. En este caso se divide por 5, ya que se necesita un reloj de 10 MHz para la sincronización de la comunicación SPI con el DAC, y la frecuencia de partida es la de 50 MHz que proporciona el reloj de la placa DE0.

Proyecto 2

Práctica 5: Diseño e implementación de una unidad de acoplamiento serie para un convertidor D/A.

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY impulso_ini IS
5    PORT (clk,reset,entrada: IN std_logic;
6          salida: OUT std_logic);
7  END;
8
9  ARCHITECTURE comportamiento OF impulso_ini IS
10   TYPE tipo IS (e0,e1,e2);
11   SIGNAL estado: tipo;
12 BEGIN
13   PROCESS (clk,reset)
14   BEGIN
15     IF reset='1' THEN estado<=e0;
16     ELSE
17       IF clk='1' AND clk'event THEN
18         CASE estado IS
19           WHEN e0 => IF entrada='0' THEN estado<=e0; ELSE estado<=e1; END IF;
20           WHEN e1 => estado<=e2;
21           WHEN e2=> IF entrada='1' THEN estado<=e2; ELSE estado<=e0; END IF;
22         END CASE;
23       END IF;
24     END IF;
25   END PROCESS;
26   PROCESS (estado)
27   BEGIN
28     CASE estado IS
29       WHEN e0 => salida<='0';
30       WHEN e1=> salida<='1';
31       WHEN e2 => salida<='0';
32     END CASE;
33   END PROCESS;
34 END comportamiento;

```

Figura 8. Módulo detector de pulsos **impulso_ini.vhd**. Este módulo genera un pulso de un ciclo de reloj de duración cuando hay un flanco de subida en la señal de entrada, a la que llega la señal externa que da la orden de hacer una conversión digital-analógica. Está pensado para que se pueda conectar como señal de orden de conversión una onda cuadrada (reloj) de la frecuencia de reconstrucción, de forma que cuando se pone a '1' sólo se haga una conversión y se espere a que vuelva a '0'.

```

1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3
4  ENTITY reg_des IS
5    PORT(dato_in:IN std_logic_vector(9 DOWNTO 0);
6          reset,carga,clk,en: IN std_logic;
7          dato_out: OUT std_logic);
8  END reg_des;
9
10 ARCHITECTURE circuito OF reg_des IS
11   SIGNAL Q:std_logic_vector (15 DOWNTO 0);
12
13 BEGIN
14   PROCESS (reset,CLK)
15   BEGIN
16     IF reset='1' THEN Q<="0000000000000000";
17     ELSE
18       IF (CLK'event AND CLK='0') THEN
19         IF carga='1' THEN Q(11 DOWNTO 2)<=dato_in; Q(15 DOWNTO 12)<="0111";
20         ELSIF en='1' THEN Q<=Q(14 DOWNTO 0) & '0';
21       END IF;
22     END IF;
23   END PROCESS;
24   dato_out<=Q(15);
25 END circuito;

```

Figura 9. Módulo **reg_des.vhd**. Se trata de un registro de desplazamiento con carga en paralelo. Se cargan los 10 bits del dato y se concatenan con los 4 de configuración y los 2 de relleno.

```

1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3  USE IEEE.numeric_std.all;
4
5  ENTITY contador IS
6  PORT(clk,inicio,en: IN std_logic;
7   fin: OUT std_logic);
8  END contador;
9
10 ARCHITECTURE algoritmo OF contador IS
11  SIGNAL Q:unsigned(4 downto 0);
12 BEGIN
13  PROCESS (inicio,clk)
14  BEGIN
15    IF inicio='1' THEN Q<="00000";
16    ELSIF clk'event AND clk='0' THEN
17      IF en='1' THEN Q<=Q + 1;
18      END IF;
19    END IF;
20  END PROCESS;
21  fin<=Q(4) AND NOT Q(3) AND NOT Q(2) AND NOT Q(1) AND NOT Q(0);
22 END algoritmo;

```

Figura 10. Módulo **contador.vhd**. Es un contador en binario natural de 4 bits que cuenta los 16 ciclos de una transmisión SPI al DAC. Además, genera la señal de **fin** de conversión al finalizar la cuenta, que se inicia cuando se desactiva la señal **inicio**.

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY uc_spi_out IS
5  PORT (clk,reset,din,sc,eoc: IN std_logic;
6   cs,en,sck,sdo: OUT std_logic);
7  END;
8
9  ARCHITECTURE comportamiento OF uc_spi_out IS
10  TYPE tipo IS (e0,e1,e2);
11  SIGNAL est: tipo;
12 BEGIN
13  PROCESS (clk,reset)
14  BEGIN
15    IF reset='1' THEN est<=e0;
16    ELSE
17      IF clk='1' AND clk'event THEN
18        CASE est IS
19          WHEN e0 => IF sc='1' THEN est<=e1; ELSE est<=e0; END IF;
20          WHEN e1 => est<=e2;
21          WHEN e2 => IF eoc='1' THEN est<=e0; ELSE est<=e2; END IF;
22        END CASE;
23      END IF;
24    END IF;
25  END PROCESS;
26
27  PROCESS(est,eoc,clk,din)
28  BEGIN
29    CASE est IS
30      WHEN e0=> cs<='1'; en<='0'; sdo<='0'; sck<='0';
31      WHEN e1=> cs<='0'; en<='0'; sdo<=din; sck<='0';
32      WHEN e2=> IF eoc='0' THEN cs<='0'; sck<=clk; ELSE cs<='1'; sck<='0'; END IF;
33      en<='1'; sdo<=din;
34    END CASE;
35  END PROCESS;
36 END comportamiento;

```

Figura 11. Módulo **uc_spi_out.vhd** con la unidad de control del controlador del DAC. Se trata de una máquina de estados que se implementa con dos procesos, uno síncrono que calcula el estado, y otro, combinacional, que calcula las salidas.

Proyecto 2

Práctica 5: Diseño e implementación de una unidad de acoplamiento serie para un convertidor D/A.

```

1  --Práctica 7: Sistemas Electrónicos Digitales
2  --Universidad de Vigo, EE Industrial, curso 2013-2014
3  --Camilo Quintáns Graña
4
5  LIBRARY ieee;
6  USE ieee.std_logic_1164.all;
7  LIBRARY work;
8
9  ENTITY spi_dac IS
10   PORT(reset,sc,clk:  IN STD_LOGIC;
11        din :  IN STD_LOGIC_VECTOR(9 DOWNTO 0);
12        sdo,eoc,sck,cs:  OUT STD_LOGIC);
13  END spi_dac;
14
15  ARCHITECTURE circuito OF spi_dac IS
16
17  COMPONENT reg_des
18   PORT(reset,carga,clk,en:  IN STD_LOGIC;
19        dato_in :  IN STD_LOGIC_VECTOR(9 DOWNTO 0);
20        dato_out : OUT STD_LOGIC);
21  END COMPONENT;
22
23  COMPONENT divisor_reloj
24  GENERIC (n: INTEGER);
25  PORT(clkin,en:  IN STD_LOGIC;
26        --para el modulo se pone n-1
27        modulo :  IN STD_LOGIC_VECTOR(2 DOWNTO 0);
28        clkout : OUT STD_LOGIC);
29  END COMPONENT;
30
31  COMPONENT impulso_ini
32  PORT(clk,reset,entrada : IN STD_LOGIC;
33        salida : OUT STD_LOGIC);
34  END COMPONENT;
35
36  COMPONENT contador
37  PORT(clk,inicio,en: IN STD_LOGIC;
38        fin : OUT STD_LOGIC);
39  END COMPONENT;
40
41  COMPONENT uc_spi_out
42  PORT(clk,reset,din,sc,eoc : IN STD_LOGIC;
43        cs,en,sck,sdo : OUT STD_LOGIC);
44  END COMPONENT;
45
46  SIGNAL  clk_spi :  STD_LOGIC;
47  SIGNAL  fin_de_conversion :  STD_LOGIC;
48  SIGNAL  habilitacion :  STD_LOGIC;
49  SIGNAL  impulso_inicio_conversion :  STD_LOGIC;
50  SIGNAL  not_sc :  STD_LOGIC;
51  SIGNAL  not_reset :  STD_LOGIC;
52  SIGNAL  valor_division :  STD_LOGIC_VECTOR(2 DOWNTO 0);
53  SIGNAL  dato_serie:  STD_LOGIC;
54
55  BEGIN
56    cp1:reg_des
57    PORT MAP(reset => not_reset,
58              carga => impulso_inicio_conversion,
59              clk => clk_spi,
60              en => habilitacion,
61              dato_in => din,
62              dato_out => dato_serie);
63
64    cp2: divisor_reloj
65    GENERIC MAP(n => 3)
66    PORT MAP(clkin => clk,
67              en => reset, --habilitar_reloj,
68              modulo => valor_division,
69              clkout => clk_spi);
70
71    cp3: impulso_ini
72    PORT MAP(clk => clk_spi,
73              reset => not_reset,
74              entrada => not_sc,
75              salida => impulso_inicio_conversion);
76
77    not_sc <= NOT(sc);
78    not_reset <= NOT(reset);
79
80    cp4: contador
81    PORT MAP(clk => clk_spi,
82              inicio => impulso_inicio_conversion,
83              en => habilitacion,
84              fin => fin_de_conversion);
85
86    cp5: uc_spi_out
87    PORT MAP(clk => clk_spi,
88              reset => not_reset,
89              din => dato_serie,
90              sc => impulso_inicio_conversion,
91              eoc => fin_de_conversion,
92              cs => cs,
93              en => habilitacion,
94              sck => sck,
95              sdo => sdo);
96
97    valor_division<="101";
98    EOC <= fin_de_conversion;
99
100
101  END circuito;

```

Figura 72. Descripción estructural **spi_dac.vhd** del controlador del DAC.

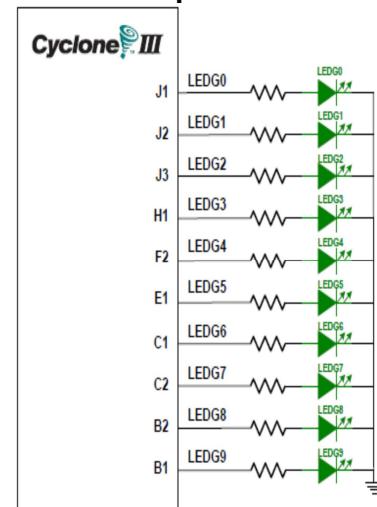
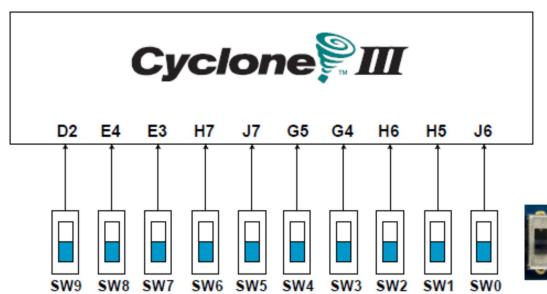
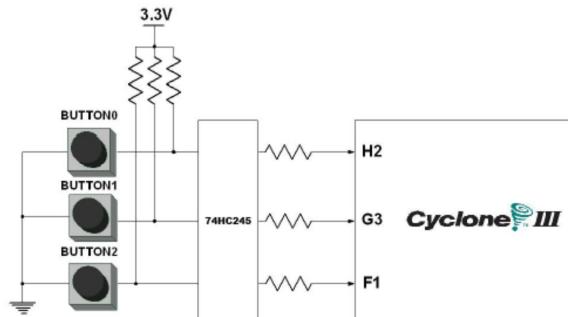
ANEXO II:
PLACA DE0. CONEXIONES ENTRE LOS PERIFÉRICOS Y LOS TERMINALES DE LA FPGA.

Interruptores

Signal Name	FPGA Pin No.
SW[0]	PIN_J6
SW[1]	PIN_H5
SW[2]	PIN_H6
SW[3]	PIN_G4
SW[4]	PIN_G5
SW[5]	PIN_J7
SW[6]	PIN_H7
SW[7]	PIN_E3
SW[8]	PIN_E4
SW[9]	PIN_D2

Diodos LED

Signal Name	FPGA Pin No.
LEDG[0]	PIN_J1
LEDG[1]	PIN_J2
LEDG[2]	PIN_J3
LEDG[3]	PIN_H1
LEDG[4]	PIN_F2
LEDG[5]	PIN_E1
LEDG[6]	PIN_C1
LEDG[7]	PIN_C2
LEDG[8]	PIN_B2
LEDG[9]	PIN_B1

Esquema LEDs**Esquema de los interruptores****Esquema de los pulsadores****Puertos de expansión de la placa DE0****(GPIO 0)
J4**

[AB12] GPIO0_CLKIN0	1	2	GPIO0_D0 [AB16]
[AA12] GPIO0_CLKIN1	3	4	GPIO0_D1 [AA16]
[AA15] GPIO0_D2	5	6	GPIO0_D3 [AB15]
[AA14] GPIO0_D4	7	8	GPIO0_D5 [AB14]
[AB13] GPIO0_D6	9	10	GPIO0_D7 [AA13]
5V	11	12	GND
[AB10] GPIO0_D8	13	14	GPIO0_D9 [AA10]
[AB8] GPIO0_D10	15	16	GPIO0_D11 [AA8]
[AB5] GPIO0_D12	17	18	GPIO0_D13 [AA5]
[AB3] GPIO0_CLKOUT0	19	20	GPIO0_D14 [AB4]
[AA3] GPIO0_CLKOUT1	21	22	GPIO0_D15 [AA4]
[V14] GPIO0_D16	23	24	GPIO0_D17 [U14]
[Y13] GPIO0_D18	25	26	GPIO0_D19 [W13]
[U13] GPIO0_D20	27	28	GPIO0_D21 [V12]
3.3V	29	30	GND
[R10] GPIO0_D22	31	32	GPIO0_D23 [V11]
[Y10] GPIO0_D24	33	34	GPIO0_D25 [W10]
[T8] GPIO0_D26	35	36	GPIO0_D27 [V8]
[W7] GPIO0_D28	37	38	GPIO0_D29 [W6]
[V5] GPIO0_D30	39	40	GPIO0_D31 [U7]

**(GPIO 1)
J5**

[AB11] GPIO1_CLKIN0	1	2	GPIO1_D0 [AA20]
[AA11] GPIO1_CLKIN1	3	4	GPIO1_D1 [AB20]
[AA19] GPIO1_D2	5	6	GPIO1_D3 [AB19]
[AB18] GPIO1_D4	7	8	GPIO1_D5 [AA18]
[AA17] GPIO1_D6	9	10	GPIO1_D7 [AB17]
5V	11	12	GND
[Y17] GPIO1_D8	13	14	GPIO1_D9 [W17]
[U15] GPIO1_D10	15	16	GPIO1_D11 [T15]
[W15] GPIO1_D12	17	18	GPIO1_D13 [V15]
[R16] GPIO1_CLKOUT0	19	20	GPIO1_D14 [AB9]
[T16] GPIO1_CLKOUT1	21	22	GPIO1_D15 [AA9]
[AA7] GPIO1_D16	23	24	GPIO1_D17 [AB7]
[T14] GPIO1_D18	25	26	GPIO1_D19 [R14]
[U12] GPIO1_D20	27	28	GPIO1_D21 [T12]
3.3V	29	30	GND
[R11] GPIO1_D22	31	32	GPIO1_D23 [R12]
[U10] GPIO1_D24	33	34	GPIO1_D25 [T10]
[U9] GPIO1_D26	35	36	GPIO1_D27 [T9]
[Y7] GPIO1_D28	37	38	GPIO1_D29 [U8]
[V6] GPIO1_D30	39	40	GPIO1_D31 [V7]