Memoria

Sistemas Electrónicos para la Automatización

Docente: *Jorge Romero Sánchez*

Práctica 2: El Registro Universal Genérico

***Jorge Benavides Macías**: jorge2@uma.es*

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Contenido

Lista de figuras

Lista de códigos

1. Especificaciones

- Codificar y verificar el funcionamiento de un registro universal de n bits genérico y esquema FSM en VHDL.
- El Registro tiene un bus de control de tres bits que, según la siguiente tabla, permite cambiar la funcionalidad del mismo.

Control	Función
000	Carga el dato a la entrada
001	Contador Ascendente
010	Contador descendente
011	Desplaza a la izquierda
100	Desplaza a la derecha
101 hasta el 111	Conserva el dato

Cuadro 1: Bus de control del registro genérico

- **Como requerimiento especial importante no se podrá utilizar el paquete numeric_std de la biblioteca ieee.**

2. Estudio teórico

2.1. Desplazamientos

Los desplazamientos son operaciones sencillas, en esencia, es mover un conjunto de bits (un rango de un array) hacia la izquierda o derecha.

- **Desplazamiento izquierda**

  En el desplazamiento hacia la izquierda seleccionamos el rango de 0 a $n - 2$, lo "copiamos" en el rango $n - 1$ a 1 y en la posición 0 asignamos el valor 0.

- **Desplazamiento derecha**

  En el desplazamiento hacia la derecha seleccionamos el rango de $n - 1$ a 1, lo "copiamos" en el rango $n - 2$ a 0 y en la posición $n - 1$ asignamos el valor 0.

Este algoritmo de desplazamiento es distinto al que podemos realizar en otros lenguajes de programación ya que nosotros podemos definir dónde está el *0*, por convención hemos usado en la asignatura la *keyword downto* para fijar el 0 por el lado menos significativo.

En la Figura 1 se representan ambos desplazamientos siguiendo la lógica expuesta con anterioridad para un n igual a 3.

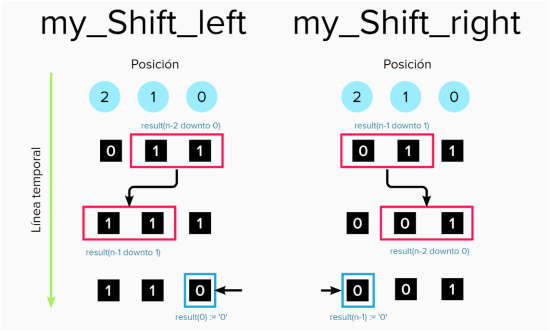

Figura 1: Infografía de los algoritmos de desplazamiento

2.2. Operaciones

- **Suma**

  La suma es una operación bit a bit, usamos dos operaciones lógicas, la *and* y la *xor* entre dos números genera la *suma o total* y el acarreo (*carry*). Este proceso se repite varias veces hasta que el acarreo sea 0. Para que este proceso tenga sentido y funcione el acarreo debe guardarse en una variable y en las sucesivas iteraciones se debe realizar la operación *and* y *xor* con dicho acarreo.

- **Resta**

  La resta es una operación similar a la realizada en la sección anterior, la diferencia radica en el uso de una puerta not en uno de los valores antes de realizar las operaciones *and* y *xor*.

En la Figura 2 se han representado la primera iteración de la suma y de la resta entre 1 y 1.

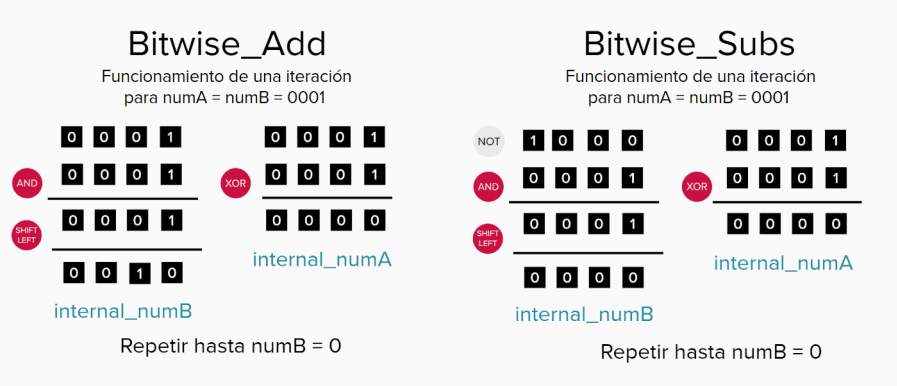

Figura 2: Infografía de los algoritmos de suma y resta

3. Diseño

3.1. Registro Universal

El registro universal es un dispositivo genérico que realiza una serie de operaciones genéricas disponibles a nivel de registro.

3.1.1. Entidad

La entidad del registro universal cuenta con 5 entradas y una salida, las 3 básicas de control *clk, cke, reset* más dos entradas de control y un número externo, la *q* es una salida de las distintas operaciones que se realizan dentro de la entidad. Para satisfacer uno de los requisitos y el nombre del dispositivo hemos definido un *n* como valor genérico.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity registro is
5    generic (n: integer := 8);
6    port(d: in std_logic_vector(n-1 downto 0);
7         control: in std_logic_vector (2 downto 0);
8         clk: in std_logic;
9         cke: in std_logic;
10        reset: in std_logic;
11        q: out std_logic_vector(n-1 downto 0));
12
13 end entity;
```

Código 1: Entidad de la FSM

3.1.2. Arquitectura

La arquitectura del registro cuenta con una serie de funciones que realizan las operaciones propuestas en los requisitos. El uso de de funciones en esta práctica tiene 3 objetivos:

- Extender el uso del lenguaje y aprender un poco más sobre VHDL.
- Simplificar la FSM implementada. En el Código 6 se puede apreciar que el núcleo de la FSM es muy simple y que se basa en llamar a la función que realiza la operación.
- Las operaciones de adición y substracción deben realizar un desplazamiento a la izquierda; en lugar de implementar dos veces el código, este se recoge en una función.

Función *my_Shift_left*
Esta función implementa el pseudoalgoritmo descrito en la Subsección 2.1, copiamos el rango[1] $[n-2:0]$ en $[n-1:1]$, añadimos un cero y devolvemos el vector.

```
1    function my_Shift_left(numA: std_logic_vector) return std_logic_vector is
2      variable result: std_logic_vector(n-1 downto 0):= (others=> '0');
3    begin
4      result:= numA;
5      result(n-1 downto 1) := result(n-2 downto 0);
6      result(0) := '0';
7      return result;
8    end function;
```

Código 2: Desplazamiento a la izquierda

[1] Recordamos que la posición menos significativa es el 0.

Función *my_Shift_right*

Esta función implementa el pseudoalgoritmo descrito en la Subsección 2.1, copiamos el rango $[n-1:1]$ en $[n-2:0]$, añadimos un cero y devolvemos el vector.

```
function my_Shift_right(numA: std_logic_vector) return std_logic_vector is
  variable result: std_logic_vector(n-1 downto 0):= (others=> '0');
begin
  result:= numA;
  result(n-2 downto 0) := result(n-1 downto 1);
  result(n-1) := '0';
  return result;
end function;

```

Código 3: Desplazamiento a la derecha

Función *Bitwise_Add*

Esta función implementa el pseudoalgoritmo descrito en la Subsección 2.1. El código es mucho más claro que la descripción de las operaciones que realiza, hay un bucle *for* cuyo objetivo es iterar las operaciones *and* y *xor* hasta que *carry* sea igual a 0, en el proceso se produce un desplazamiento que recurre a una función ya definida en el párrafo anterior.

```
function Bitwise_Add(numA, numB :std_logic_vector) return std_logic_vector is
  variable carry: std_logic_vector(n-1 downto 0):= (others=> '0');
  variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
  variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');

begin
  internal_numA := numA;
  internal_numB := numB;
  for i in n-1 downto 0 loop
    carry := internal_numA and internal_numB;
    internal_numA := internal_numA xor internal_numB;
    internal_numB := my_Shift_left(carry);
  end loop;
  return internal_numA;
end function;
```

Código 4: Operación suma bit a bit

Función *Bitwise_Subs*

Esta función implementa el pseudoalgoritmo descrito en la Subsección 2.1. El código es mucho más claro que la descripción de las operaciones que realiza, hay un bucle *for* cuyo objetivo es iterar las operaciones *not*, *and* y *xor* hasta que *borrow* sea igual a 0, en el proceso se produce un desplazamiento que recurre a una función ya definida en el párrafo anterior.

```
function Bitwise_Subs(numA, numB:std_logic_vector) return std_logic_vector is
  variable borrow: std_logic_vector(n-1 downto 0):= (others=> '0');
  variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
  variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');
begin
  internal_numA := numA;
  internal_numB := numB;

  for i in n-1 downto 0 loop
    borrow := ((not internal_numA) and internal_numB);
    internal_numA := internal_numA xor internal_numB;
    internal_numB := my_Shift_left(borrow);
  end loop;
```

```
14      return internal_numA;
15    end function;
```

Código 5: Operación resta bit a bit

Proceso Secuencial y Combinacional
La arquitectura cuenta con dos procesos, uno secuencial con una lista de sensibilidades que incluye el *reset*, *clk* y la señal de control "*control*", cuando el *reset* es igual a 1 la variable auxiliar *d_aux* es 0, si el reset es 0 comprueba que hay un flanco de subida del reloj y que el *cke* está activo entonces selecciona el estado según la señal de *control*; el otro proceso es combinacional y es simplemente una asignación de la variable auxiliar *d_aux* a la salida *q*. Esta FSM es distinta de otras hechas previamente ya que no tiene un tipo *estado* definido, ni un número finito de estados, depende tamaño de la señal de control que en este caso es 3 de bits.

El último estado se podría haber evitado de alguna manera porque la asignación que estamos realizando carece de sentido, en un lenguaje de alto nivel como *C++* sería equivalente a escribir *a = a;* que no tiene ningún tipo de sentido porque no tiene efecto en el valor original de la variable, para este caso lo hemos dejado por claridad del código.

```
1   begin
2     Secuencial: process (clk, reset, control)
3     begin
4       if reset = '1' then
5         d_aux <= (others => '0');
6       elsif rising_edge( clk ) and cke = '1' then
7         if control = "000" then           -- Carga
8           d_aux <= d;
9         elsif control = "001" then        -- Cuenta ascendente
10          d_aux <= Bitwise_Add(d_aux,number_1);
11        elsif control = "010" then        -- Cuenta descendente
12          d_aux <= Bitwise_Subs(d_aux,number_1);
13        elsif control = "011" then        -- Desplazamiento izquierda
14          d_aux <= my_Shift_left(d_aux);
15        elsif control = "100" then        -- Desplazamiento derecha
16          d_aux <= my_Shift_right(d_aux);
17        elsif control = "101" or control = "110" or control = "111" then -- Mantiene
18          d_aux <= d_aux;
19        end if;
20      end if;
21    end process;
22
23    q <= d_aux;
24  end behavioral;
```

Código 6: Arquitectura de la FSM

4. Simulación

4.1. Test Bench

La arquitectura del banco de pruebas está basada en el fichero de estímulos proporcionado en el campus. El fichero en cuestión ha sido modificado para probar el registro; instanciamos el componente "registro", algunas constantes para gestionar la frecuencia del *clk*, la señal de *control* y de tamaño del registro universal genérico.

Hemos definido más señales internas para simulación que se han asociado en el proceso *DUT (Device Under Test)*. Un proceso llamado *clock_gen* para generar el reloj, el *cke_interno* a 1, un proceso de *reset* y uno de *control*, en el proceso de *control* se prueban todos los estados posibles durante un tiempo determinado.

El resto de estímulos está hecho en fichero .txt cuyos valores son las posibles entradas de la *d*.

```
architecture Comportamiento of Test_Bench is
-- Se instancia el componente del registro universal
  component registro
    generic (n : integer := 8);
    port(d       : in  std_logic_vector(n-1 downto 0);
         control : in  std_logic_vector (2 downto 0);
         clk     : in  std_logic;
         cke     : in  std_logic;
         reset   : in  std_logic;
         q       : out std_logic_vector(n-1 downto 0));
  end Component registro;

  -- Algunas constantes de tiempo
  constant semiperiodo : time := 10 ns;
  constant tiempo_control : time := 50 ns;
  constant n : integer := 3; -- Tamaño de la entrada

  signal d_interno, q_interno : std_logic_vector(n-1 downto 0) := (others => 'U');
  signal control_interno : std_logic_vector(2 downto 0) := (others => 'U');
  signal reset_interno, cke_interno : std_logic := 'U';
  signal clk_interno : std_logic := '0';

begin

  DUT : registro
    generic map (n)
    port map(
      d       => d_interno,
      q       => q_interno,
      control => control_interno,
      reset   => reset_interno,
      clk     => clk_interno,
      cke     => cke_interno);

-- Taken from The Student Guide to VHDL, Peter J.Asheden
  clock_gen : process (clk_interno) is
  begin
    if clk_interno = '0' then
      clk_interno <= '1' after semiperiodo, '0' after 2*semiperiodo;
    end if;
  end process clock_gen;

  cke_interno <= '1';

  reset : process
  begin
    reset_interno <= '0';
    wait for 5 ns;
    reset_interno <= '1';
    wait for 5 ns;
    reset_interno <= '0';
    wait;
  end process reset;

  control : process
  begin -- Selección de distintos estados del registro genérico universal
    control_interno <= "000"; -- Carga
    wait for tiempo_control;
    control_interno <= "001"; -- Contador Ascendente
    wait for tiempo_control;
    control_interno <= "010"; -- Contador Descendente
    wait for tiempo_control;
    control_interno <= "011"; -- Desplazamiento a la izquierda
    wait for tiempo_control;
    control_interno <= "100"; -- Desplazamiento a la derecha
    wait for tiempo_control - 20 ns;
    control_interno <= "101"; -- Matener el valor
    wait for tiempo_control;
    control_interno <= "110"; -- Matener el valor
    wait for tiempo_control;
    control_interno <= "111"; -- Matener el valor
```

```
72      wait for tiempo_control;
73      wait;
74    end process control;
75  [...]
```

Código 7: Test bench de la FSM

4.2. Fichero de estímulos

El fichero de estímulos es es una secuencia de números aleatorios cuyo objetivo es generar un archivo *csv* para realizar una verificación en Analog Discovery.

```
#Fichero de Estímulos de Registro Universal.
#Device Name: Discovery2NI
#Nombre: Izan Amador, Jorge Benavides
#Fecha: 11 de Enero de 2022.
#
# Delay Time (ns) Dato de entrada (D) [2:0]
  0  ns      000
  20 ns      001
  20 ns      011
  20 ns      100
  20 ns      011
  20 ns      101
  20 ns      110
  20 ns      111
```

4.3. Cronograma de simulación

En la Figura 3 se presenta el resultado de la simulación, en magenta se representan las entradas y en rojo las salidas. La simulación ha sido un éxito. Se representan los 7 estados perfectamente.

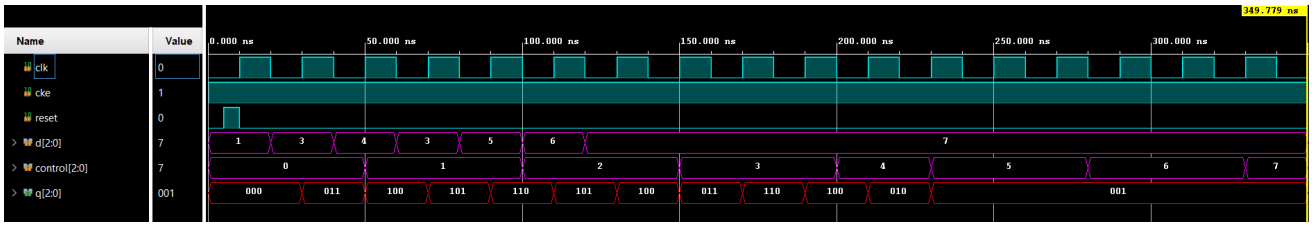

Figura 3: Cronograma de simulación

4.4. Fichero CSV generado

El CSV generado a partir del fichero de estímulos.*txt* será de utilidad para la verificación con Analago Discovery.

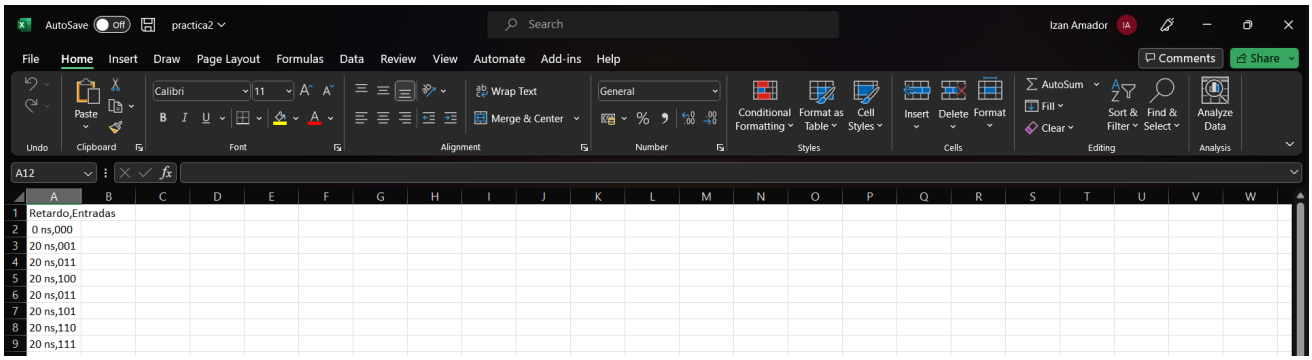

Figura 4: CSV generado del fichero de estímulos

5. Síntesis

La síntesis del dispositivo (ver Figura 5) ha sido un éxito no tiene ningún Latch. Se puede diferenciar perfectamente la zona secuencial de la combinacional por los elementos de memoria.

5.1. Esquemático RTL Registro Universal

El esquemático corresponde al registro universal con un tamaño de 3 bits.

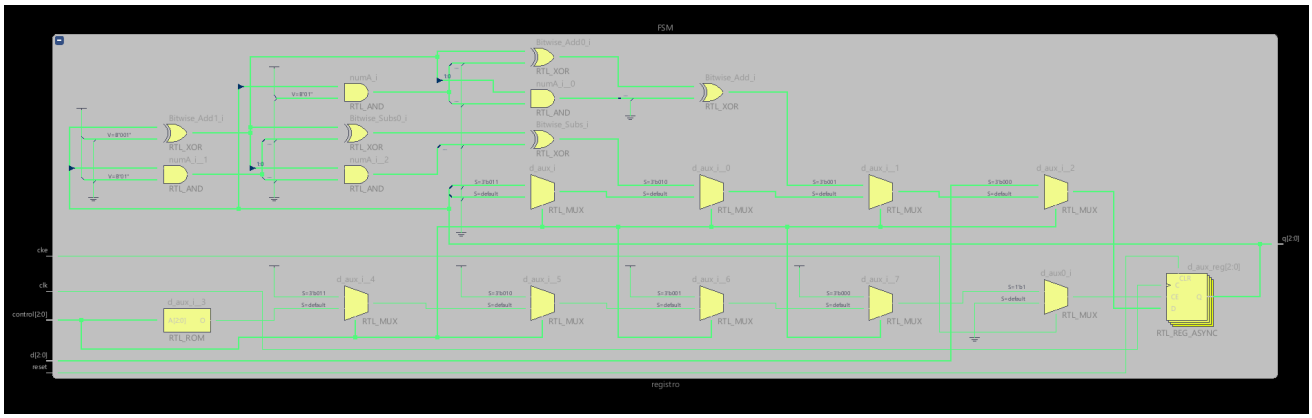

Figura 5: RTL del registro universal.

5.2. Esquemático RTL Top

El esquemático del top es interesante porque cuenta con dos sincronizadores (componente hecho en prácticas anteriores), uno para el *reset* y otro para el *cke*, además de la FSM.

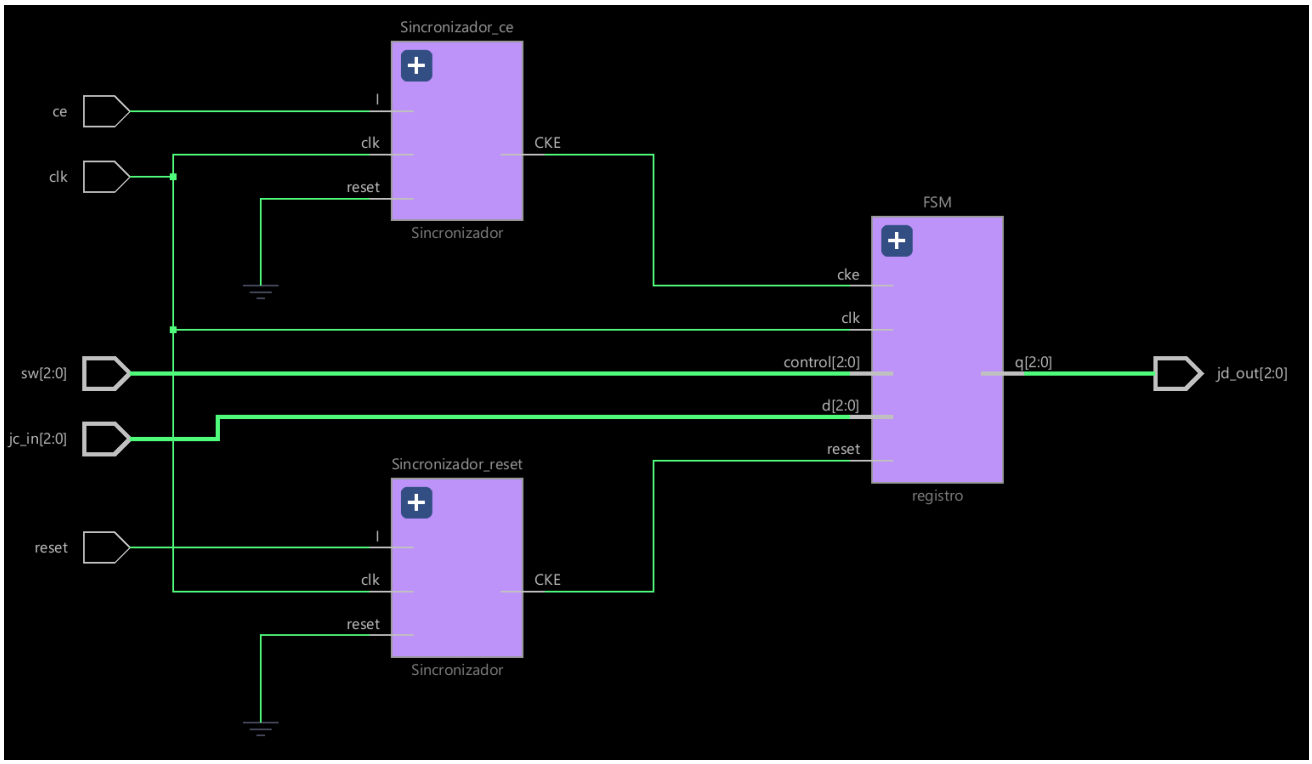

Figura 6: RTL del Top

6. Verificación con Analog Discovery

Para realizar la verificación hemos modificado el archivo .xdc para activar los pmod JC y JD, además de los switches disponibles.

En las siguientes imágenes se representa cada uno de los estados implementados en la FSM verificado con Analog Discovery.

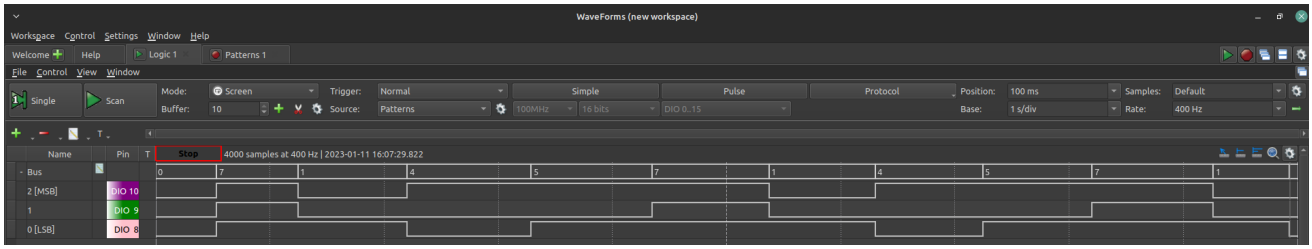

Figura 7: Verificación Analog Discovery - Estado: Cargar dato.

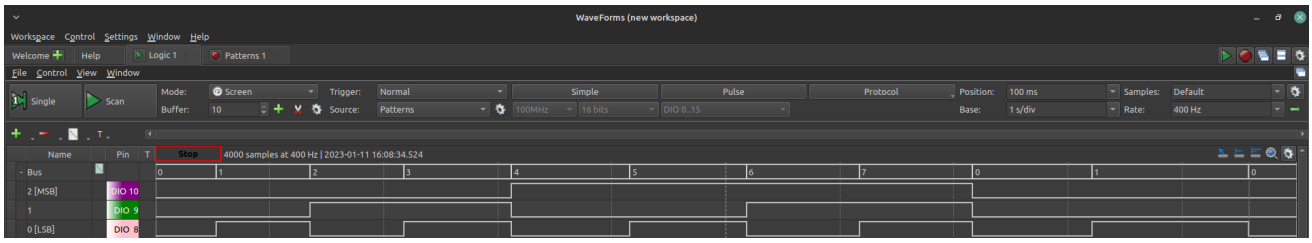

Figura 8: Verificación Analog Discovery - Estado: Contador Ascendente.

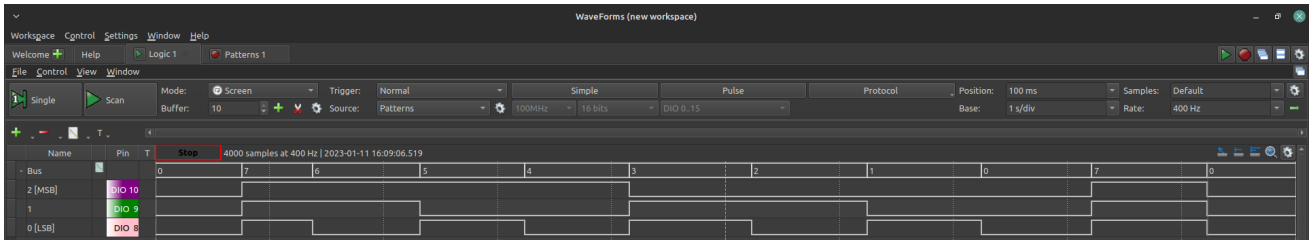

Figura 9: Verificación Analog Discovery - Estado: Contador Descendente.

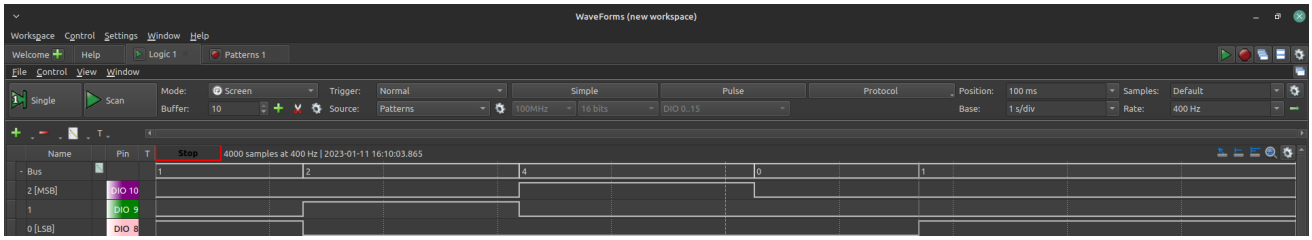

Figura 10: Verificación Analog Discovery - Estado: Desplazamiento a la izquierda.

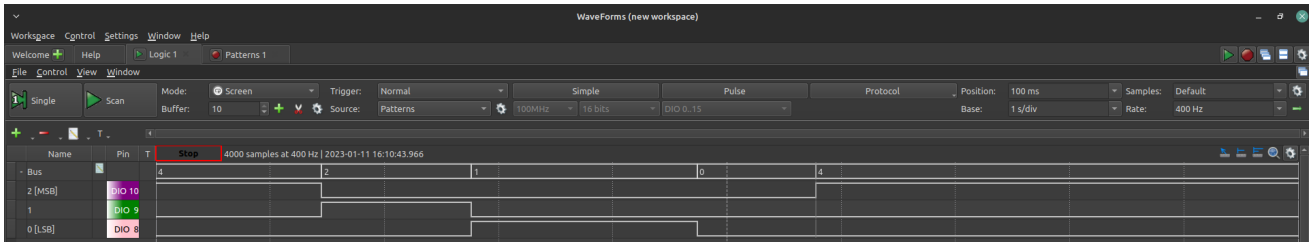

Figura 11: Verificación Analog Discovery - Estado: Desplazamiento a la derecha.

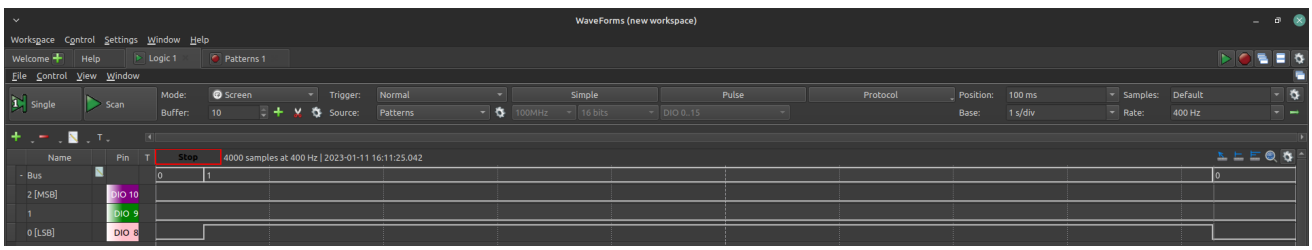

Figura 12: Verificación Analog Discovery - Estado: Conservar dato.

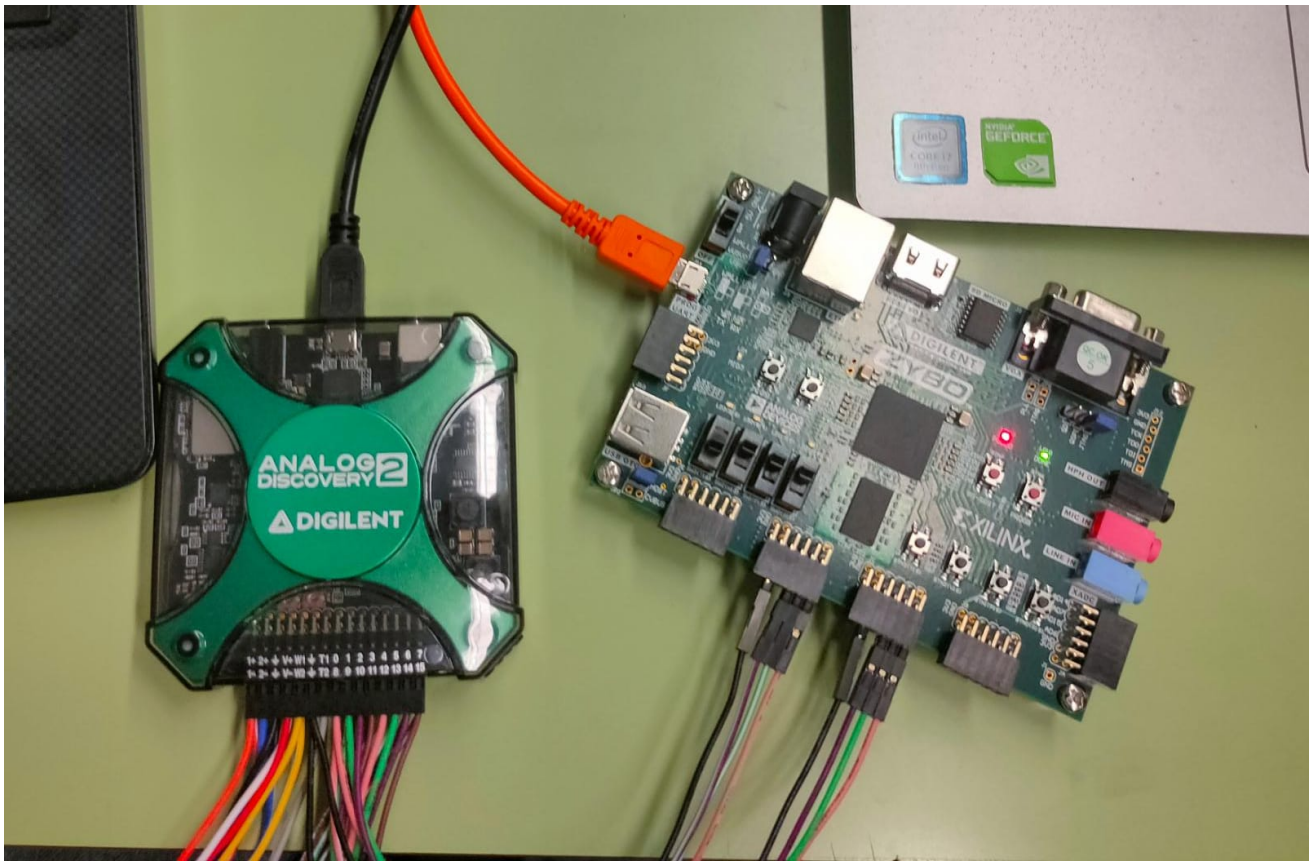

Figura 13: Montaje para verificación del registro universal genérico con Analog Discovery

7. Anexos

7.1. Top

7.1.1. Entidad

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity top is
5    generic(n : integer := 3;
6            filter_size  : integer := 32);
7    port(jc_in : in std_logic_vector (n-1 downto 0); -- D
8         sw    : in std_logic_vector (2 downto 0);   -- Control
9         ce    : in std_logic;
10        clk   : in std_logic;
11        reset : in std_logic;
12        jd_out: out std_logic_vector(n-1 downto 0)); -- q
13 end top;
```

Código 8: Entidad del top

7.1.2. Arquitectura

```
architecture Behavioral of top is
  component Sincronizador
    generic (m : integer := 1);  -- tamano del filtro
    Port(
      I     : in  std_logic;
      CKE   : out std_logic;
      reset : in  std_logic;
      clk   : in  std_logic);
  end component Sincronizador;

  component registro
    generic (n: integer := 8);
    port(d: in std_logic_vector(n-1 downto 0);
         control: in std_logic_vector (2 downto 0);
         clk: in std_logic;
         cke: in std_logic;
         reset: in std_logic;
         q: out std_logic_vector(n-1 downto 0));
  end component registro;

  signal reset_interno : std_logic := 'U';
  signal ce_interno    : std_logic := 'U';

begin
  Sincronizador_reset : Sincronizador generic map(filter_size)
    port map(
      I     => reset,
      CKE   => reset_interno,
      reset => '0',
      clk   => clk
      );

  Sincronizador_ce : Sincronizador generic map(filter_size)
    port map(
      I     => ce,
      CKE   => ce_interno,
      reset => '0',
      clk   => clk
      );

  FSM : registro generic map(n)
    port map(
      d => jc_in,
      control => sw,
      clk => clk,
      cke => ce_interno,
      reset => reset_interno,
      q => jd_out);

end Behavioral;
```

Código 9: Arquitectura del top

7.2. Registro genérico universal

```
----------------------------------------------------------------------------------
-- Company: Universidad de Málaga
-- Engineer: Izan Amador, Jorge L. Benavides
--
-- Create Date: 11.1.2022
-- Design Name: registro
-- Module Name: registro
-- Project Name: practica2
-- Target Devices: Zybo
-- Tool Versions: Vivado 2022.1
-- Description: Universal register with several operations without numeric library.
--
-- Dependencies:
```

```
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  -- Sincronizers implemented in buttons
19  ----------------------------------------------------------------------------
20  library ieee;
21  use ieee.std_logic_1164.all;
22
23  entity registro is
24    generic (n: integer := 8);
25    port(d: in std_logic_vector(n-1 downto 0);
26         control: in std_logic_vector (2 downto 0);
27         clk: in std_logic;
28         cke: in std_logic;
29         reset: in std_logic;
30         q: out std_logic_vector(n-1 downto 0));
31
32  end entity;
33
34  architecture behavioral of registro is
35    signal d_aux: std_logic_vector(n-1 downto 0):= (others => '0');
36    signal number_1: std_logic_vector(n-1 downto 0) := (0 => '1', others => '0');
37  ---------------------------------FUNCIONES-----------------------------------
38    function my_Shift_left(numA: std_logic_vector) return std_logic_vector is
39      variable result: std_logic_vector(n-1 downto 0):= (others=> '0');
40    begin
41      result:= numA;
42      result(n-1 downto 1) := result(n-2 downto 0);
43      result(0) := '0';
44      return result;
45    end function;
46
47    function my_Shift_right(numA: std_logic_vector) return std_logic_vector is
48      variable result: std_logic_vector(n-1 downto 0):= (others=> '0');
49    begin
50      result:= numA;
51      result(n-2 downto 0) := result(n-1 downto 1);
52      result(n-1) := '0';
53      return result;
54    end function;
55
56    function Bitwise_Add(numA, numB :std_logic_vector) return std_logic_vector is
57      variable carry: std_logic_vector(n-1 downto 0):= (others=> '0');
58      variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
59      variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');
60
61    begin
62      internal_numA := numA;
63      internal_numB := numB;
64      for i in n-1 downto 0 loop
65        carry := internal_numA and internal_numB;
66        internal_numA := internal_numA xor internal_numB;
67        internal_numB := my_Shift_left(carry);
68      end loop;
69      return internal_numA;
70    end function;
71
72    function Bitwise_Subs(numA, numB:std_logic_vector) return std_logic_vector is
73      variable borrow: std_logic_vector(n-1 downto 0):= (others=> '0');
74      variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
75      variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');
76    begin
77      internal_numA := numA;
78      internal_numB := numB;
79
80      for i in n-1 downto 0 loop
81        borrow := ((not internal_numA) and internal_numB);
82        internal_numA := internal_numA xor internal_numB;
83        internal_numB := my_Shift_left(borrow);
84      end loop;
85      return internal_numA;
86    end function;
87  ----------------------------------------------------------------------------
88  begin
89    Secuencial: process (clk, reset, control)
90    begin
91      if reset = '1' then
92        d_aux <= (others => '0');
```

```
    elsif rising_edge( clk ) and cke = '1' then
      if control = "000" then           -- Carga
        d_aux <= d;
      elsif control = "001" then        -- Cuenta ascendente
        d_aux <= Bitwise_Add(d_aux,number_1);
      elsif control = "010" then        -- Cuenta descendente
        d_aux <= Bitwise_Subs(d_aux,number_1);
      elsif control = "011" then        -- Desplazamiento izquierda
        d_aux <= my_Shift_left(d_aux);
      elsif control = "100" then        -- Desplazamiento derecha
        d_aux <= my_Shift_right(d_aux);
      elsif control = "101" or control = "110" or control = "111" then -- Mantiene
        d_aux <= d_aux;
      end if;
    end if;
  end process;

  q <= d_aux;
end behavioral;
```

Código 10: Entidad del top

7.3. Test Bench

```
----------------------------------------------------------------------------------
-- Company: Universidad de Málaga
-- Engineer: Izan Amador, Jorge L. Benavides
--
-- Create Date: 23.11.2022 17:47:41
-- Design Name: Debouncer
-- Module Name: Test_Bench - Behavioral
-- Project Name: Sincronizador
-- Target Devices: Zybo
-- Tool Versions: Vivado 2022.1
-- Description: Debouncer for a button.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use STD.textIO.ALL;                     -- Se va a hacer uso de ficheros.

entity Test_Bench is
--  Port ( );
end Test_Bench;

architecture Comportamiento of Test_Bench is

  component registro
    generic (n : integer := 8);
    port(d       : in  std_logic_vector(n-1 downto 0);
         control : in  std_logic_vector (2 downto 0);
         clk     : in  std_logic;
         cke     : in  std_logic;
         reset   : in  std_logic;
         q       : out std_logic_vector(n-1 downto 0));
  end Component registro;


  constant semiperiodo    : time    := 10 ns;
  constant tiempo_control : time    := 50 ns;
  constant n              : integer := 3;


  signal d_interno, q_interno       : std_logic_vector(n-1 downto 0) := (others => 'U');
  signal control_interno            : std_logic_vector(2 downto 0)   := (others => 'U');
```

```
49    signal reset_interno, cke_interno : std_logic                    := 'U';
50    signal clk_interno                : std_logic                    := '0';
51
52
53  begin
54
55    DUT : registro
56      generic map (n)
57      port map(
58        d       => d_interno,
59        q       => q_interno,
60        control => control_interno,
61        reset   => reset_interno,
62        clk     => clk_interno,
63        cke     => cke_interno);
64
65  -- Taken from The Student Guide to VHDL, Peter J.Asheden
66    clock_gen : process (clk_interno) is
67    begin
68      if clk_interno = '0' then
69        clk_interno <= '1' after semiperiodo,
70                       '0' after 2*semiperiodo;
71      end if;
72    end process clock_gen;
73
74    cke_interno <= '1';
75
76    reset : process
77    begin
78      reset_interno <= '0';
79      wait for 5 ns;
80      reset_interno <= '1';
81      wait for 5 ns;
82      reset_interno <= '0';
83      wait;
84    end process reset;
85
86    control : process
87    begin
88      control_interno <= "000";
89      wait for tiempo_control;
90      control_interno <= "001";
91      wait for tiempo_control;
92      control_interno <= "010";
93      wait for tiempo_control;
94      control_interno <= "011";
95      wait for tiempo_control;
96      control_interno <= "100";
97      wait for tiempo_control - 20 ns;
98      control_interno <= "101";
99      wait for tiempo_control;
100     control_interno <= "110";
101     wait for tiempo_control;
102     control_interno <= "111";
103     wait for tiempo_control;
104     wait;
105   end process control;
106
107   Estimulos_Desde_Fichero : process
108
109     file Input_File  : text;
110     file Output_File : text;
111
112     variable Input_Data   : BIT_VECTOR(n-1 downto 0) := (OTHERS => '0');
113     variable Delay        : time                     := 0 ms;
114     variable Input_Line   : line                     := NULL;
115     variable Output_Line  : line                     := NULL;
116     variable Std_Out_Line : line                     := NULL;
117     variable Correcto     : Boolean                  := True;
118     constant Coma         : character                := ',';
119
120
121   begin
122 -- sumador_Estimulos.txt contiene los estÃmulos y los tiempos de retardo para el semisumador.
123     file_open(Input_File, "C:\Users\izana\Documents\GitHub\SEA\Estimulos\practica2_Estimulos.txt", read_mode);
124
125 -- sumador_Estimulos.csv contiene los estÃmulos y los tiempos de retardo para el Analog Discovery 2.
126     file_open(Output_File, "C:\Users\izana\Documents\GitHub\SEA\CSV\practica2.csv", write_mode);
127
```

```
128 -- Titles: Son para el formato EXCEL *.CSV (Comma Separated Values):
129     write(Std_Out_Line, string'("Retardo"), right, 7);
130     write(Std_Out_Line, Coma, right, 1);
131     write(Std_Out_Line, string'("Entradas"), right, 8);
132 
133     Output_Line := Std_Out_Line;
134 
135     writeline(output, Std_Out_Line);
136     writeline(Output_File, Output_Line);
137 
138     while (not endfile(Input_File)) loop
139 
140       readline(Input_File, Input_Line);
141 
142       read(Input_Line, Delay, Correcto);  -- Comprobación de que se trata de un texto que representa
143       -- el retardo, si no es así leemos la siguiente línea.
144       if Correcto then
145 
146         read(Input_Line, Input_Data);  -- El siguiente campo es el vector de pruebas.
147         d_interno <= TO_STDLOGICVECTOR(Input_Data);
148         -- De forma simultánea lo volcaremos en consola en csv.
149         write(Std_Out_Line, Delay, right, 5);  -- Longitud del retardo, ej. "20 ms".
150         write(Std_Out_Line, Coma, right, 1);
151         write(Std_Out_Line, Input_Data, right, 2);  --Longitud de los datos de entrada.
152 
153         Output_Line := Std_Out_Line;
154 
155         writeline(output, Std_Out_Line);
156         writeline(Output_File, Output_Line);
157 
158         wait for Delay;
159       end if;
160     end loop;
161 
162     file_close(Input_File);             -- Cerramos el fichero de entrada.
163     file_close(Output_File);            -- Cerramos el fichero de salida.
164     wait;
165   end process Estimulos_Desde_Fichero;
166 
167 
168 end Comportamiento;
```

Código 11: Test Bench Completo