**Memoria**

*Sistemas Electrónicos para la Automatización*

**Docente:** *Jorge Romero Sánchez*

# *Práctica: El Registro Universal Genérico*

***Izan Amador Bustos**: izan.amador@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

Para implementar las funciones anteriormente requeridas, es necesario describir una serie de algoritmos que se correspondan con las especificaciones.

En esta sección se va a hacer un análisis intuitivo de los algoritmos con ejemplos que ilustren el funcionamiento de los mismos.

## 2.1. Desplazamientos

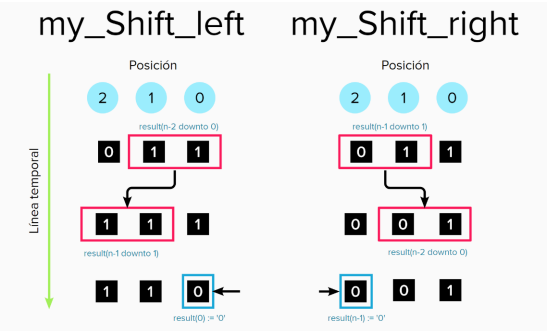


Figura 1: Infografía de los algoritmos de desplazamiento

Para ilustrar la idea de desplazamiento se plantea los ejemplos con números de tres bits.

**A la izquierda**
La idea es seleccionar un array desde la parte menos significativa del número, desplazarlo a la izquierda, e inyectar un cero en el bit menos significativo.

**A la derecha**
Se realiza el desplazamiento desde los bit más significativos, y se inyecta un cero en el bit más significativo.

### 2.2. Suma y Resta

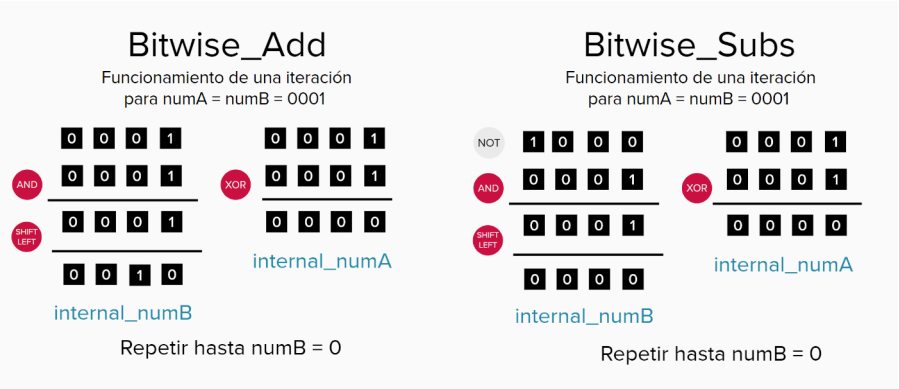


Figura 2: Infografía de los algoritmos de suma y resta

Para realizar las operaciones de suma y resta, la idea es usar los operadores *AND* y *XOR* bit a bit y las funciones de desplazamiento planteadas anteriormente.

**Para la suma**, de manera iterativa, es necesario por un lado, almacenar el resultado de la operación *XOR* entre las entradas (almacenadas en variables internas), en *A* y calcular el acarreo y desplazarlo a la izquierda para almacenarlo en el valor de *B*.

**Para la resta**, es necesario realizar un procedimiento similar llevando la cuenta en una variable llamada *borrow*, que se obtiene aplicando la operación de negación a uno de los operandos.

Ambos algoritmos se explicarán con más detalle en su implementación en el Código 4, y el Código 5.

## 3. Diseño

### 3.1. Registro Universal

#### 3.1.1. Entidad

```
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

Se definen los puertos de entrada en función de las especificaciones requeridas. Cabe destacar el caracter **genérico** del sistema.

### 3.1.2. Arquitectura

En la arquitectura se definen distintas funciones que implementan las funcionalidades necesarias para los diferentes estados de la FSM.

#### Función *my_Shift_left*

```
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

En el Código 2, para realizar la implementación de la función de **desplazamiento a la izquierda** se toma como entrada una variable *numA* de tipo *std_logic_vector* y se devuelve una variable *result* del mismo tipo.

El tamaño de *result*, se establece como $n - 1$ *downto* 0, y se le asigna a $n - 2$ *downto* 0, para realizar un desplazamiento a la izquierda de la posición de cada bit, perdiendo el primer bit. Además, el último bit se establece como un 0 y finalmente se devuelve el valor de *result*.

#### Función *my_Shift_right*

```
1   function my_Shift_right(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-2 downto 0) := result(n-1 downto 1);
6     result(n-1) := '0';
7     return result;
8   end function;
9
```

Código 3: Desplazamiento a la derecha

El Código 3, de la función de desplazamiento hacia la derecha, presenta la misma estructura.

Sin embargo se le asigna a $n - 2$ *downto* 0 el resultado de $n - 1$ *downto* 0 para desplazar el conjunto hacia la derecha, y se le añade un cero en el bit más significativo.

### Función *Bitwise_Add*

```
1  function Bitwise_Add(numA, numB :std_logic_vector) return std_logic_vector is
2    variable carry: std_logic_vector(n-1 downto 0):= (others=> '0');
3    variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
4    variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');
5
6  begin
7    internal_numA := numA;
8    internal_numB := numB;
9    for i in n-1 downto 0 loop
10     carry := internal_numA and internal_numB;
11     internal_numA := internal_numA xor internal_numB;
12     internal_numB := my_Shift_left(carry);
13   end loop;
14   return internal_numA;
15 end function;
```

Código 4: Operación suma bit a bit

La función toma dos argumentos de entrada *numA* y *numB*, de tipo *std_logic_vector* y devuelve un resultado del mismo tipo.

Las entradas se almacenan como variables internas, para realizar las operaciones el bucle for.

Se calcula un acarreo como la operación *AND* entre ambas variables internas. En la primera variable se almacena el valor de la operación *XOR* entre ambas variables y en la segunda variable se almacena el acarreo desplazado, haciendo uso de la función definida anteriormente.

### Función *Bitwise_Subs*

```
1  function Bitwise_Subs(numA, numB:std_logic_vector) return std_logic_vector is
2    variable borrow: std_logic_vector(n-1 downto 0):= (others=> '0');
3    variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
4    variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');
5  begin
6    internal_numA := numA;
7    internal_numB := numB;
8
9    for i in n-1 downto 0 loop
10     borrow := ((not internal_numA) and internal_numB);
11     internal_numA := internal_numA xor internal_numB;
12     internal_numB := my_Shift_left(borrow);
13   end loop;
14   return internal_numA;
15 end function;
```

Código 5: Operación resta bit a bit

La función sigue la misma estructura que la de adición. La diferencia principal radica en que el cálculo del acarreo se sustituye por el de la variable *borrow* que representa el *préstamo* de la resta, y se calcula negando el primer operando y realizando la operación *AND* con el segundo.

El resultado de desplazar la variable se almacena en *internal_numB*.

El resto de la función sigue un desarrollo similar al de la función de suma.

#### Proceso Secuencial y Combinacional

```
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

Se establecen dos procesos para implementar una FSM:

- **Secuencial**: se define una máquina de estados con la codificación especificada en el enunciado y el uso de las funciones definidas para la práctica para los diferentes casos. En el caso de mantener el valor, realizamos una auto-asignación que podría ser omitida.

  La máquina cuenta con un reset que reinicia su funcionamiento, actúa por flanco de subida de reloj y necesita la activación de la señal de *cke* para poder funcionar.

- **Combinacional**: el proceso se encuentra definido de manera implícita. Se trata de la asignación de *la línea 23*.

# 4. Simulación

## 4.1. Test Bench

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

```
11    end Component registro;
12
13    -- Algunas constantes de tiempo
14    constant semiperiodo : time := 10 ns;
15    constant tiempo_control : time := 50 ns;
16    constant n : integer := 3; -- Tamaño de la entrada
17
18
19    signal d_interno, q_interno : std_logic_vector(n-1 downto 0) := (others => 'U');
20    signal control_interno : std_logic_vector(2 downto 0) := (others => 'U');
21    signal reset_interno, cke_interno : std_logic := 'U';
22    signal clk_interno : std_logic := '0';
23
24  begin
25
26    DUT : registro
27      generic map (n)
28      port map(
29        d       => d_interno,
30        q       => q_interno,
31        control => control_interno,
32        reset   => reset_interno,
33        clk     => clk_interno,
34        cke     => cke_interno);
35
36  -- Taken from The Student Guide to VHDL, Peter J.Asheden
37    clock_gen : process (clk_interno) is
38    begin
39      if clk_interno = '0' then
40        clk_interno <= '1' after semiperiodo,
41                       '0' after 2*semiperiodo;
42      end if;
43    end process clock_gen;
44
45    cke_interno <= '1';
46
47    reset : process
48    begin
49      reset_interno <= '0';
50      wait for 5 ns;
51      reset_interno <= '1';
52      wait for 5 ns;
53      reset_interno <= '0';
54      wait;
55    end process reset;
56
57   -- Selección de distintos estados del registro genérico universal
58    control : process
59    begin
60      control_interno <= "000"; -- Carga
61      wait for tiempo_control;
62      control_interno <= "001"; -- Contador Ascendente
63      wait for tiempo_control;
64      control_interno <= "010"; -- Contador Descendente
65      wait for tiempo_control;
66      control_interno <= "011"; -- Desplazamiento a la izquierda
67      wait for tiempo_control;
68      control_interno <= "100"; -- Desplazamiento a la derecha
69      wait for tiempo_control - 20 ns;
70      control_interno <= "101"; -- Matener el valor
71      wait for tiempo_control;
72      control_interno <= "110"; -- Matener el valor
73      wait for tiempo_control;
74      control_interno <= "111"; -- Matener el valor
75      wait for tiempo_control;
76      wait;
77    end process control;
78
79  [...]
```

Código 7: Test bench de la FSM

En el banco de pruebas, se define el componente del registro con un tamaño genérico y los puertos de entrada y salida para los datos *d*, *control*, *cke*, *reset* y *q*.

Posteriormente, se declaran las señales internas para conectarlos a los puertos de entrada y salida, y se ins-

tancia el componente.

Se definen distintos procesos: para la generación de una señal de reloj, establecer el *cke* interno a 1, realizar el reset del sistema.

Además se establece un **proceso llamado control** encargado de establecer los distintos estados del registro universal.

El banco de pruebas completo se detalla en el Código 11.

## 4.2. Fichero de estímulos

```
#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
```

Se crea un fichero de estímulos para la simulación con diferentes valores para el dato de entrada que se alteran cada 20 nanosegundos.

## 4.3. Cronograma de simulación

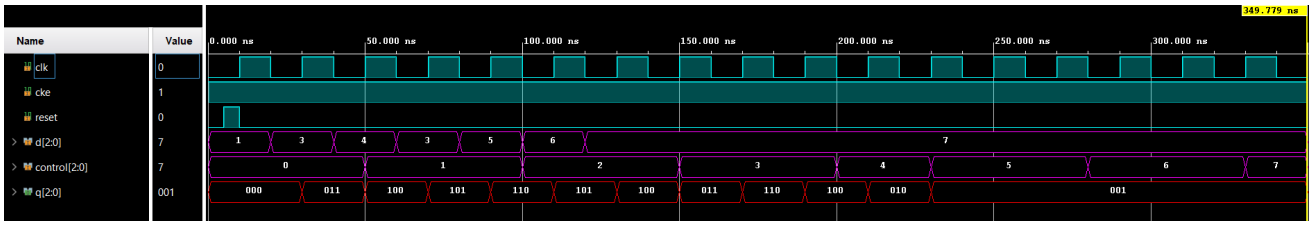


Figura 3: Cronograma de simulación

En la simulación podemos apreciar que las señales de *clk*, *cke* y *reset* funcionan sin problema. Se observa la correcta carga de la entrada desde el fichero en *d* y la asignación en de la señal *control* en el proceso.

En la salida *q*, se pueden apreciar las distintas funciones de la FSM en los flancos de subida: la carga, el contador ascendente y descendente, los desplazamientos y la conservación del dato.

### 4.4. Fichero CSV generado

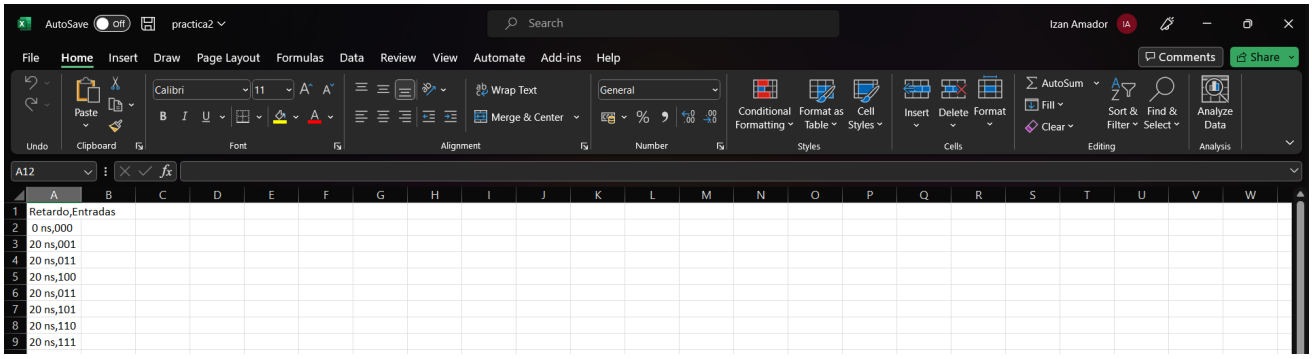


Figura 4: Archivo CSV generado

Se observa como el fichero de valores separados mediante comas se corresponde con los estímulos.

## 5. Síntesis

### 5.1. Esquemático RTL Registro Universal

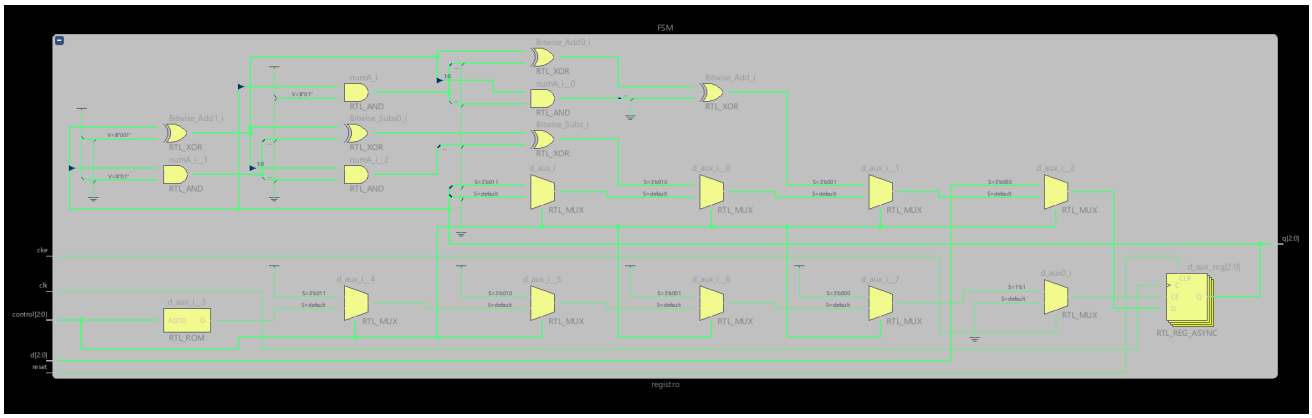


Figura 5: RTL del registro universal.

En la Figura 6, se aprecia un registro universal de 3 bits, con su correspondiente parte secuencial y combinacional.

## 5.2. Esquemático RTL Top

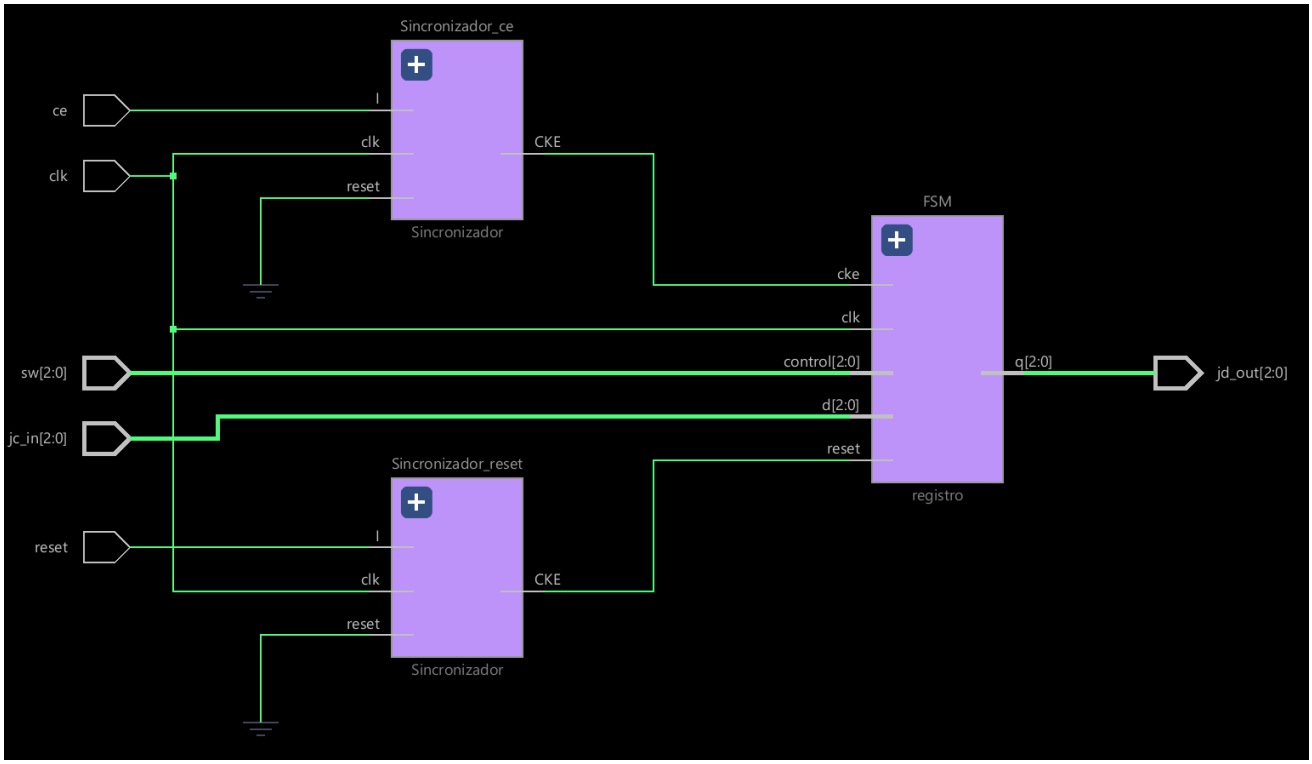


Figura 6: RTL del Top

En el esquemático del top se observa el correcto conexionado de los módulos sincronizadores para los botones.

# 6. Verificación con Analog Discovery

Para la verificación en Analog Discovery se modificó el fichero .xdc para usar los PMODs JC y JD, además de los botones para el reset y cke, y los switches para la codificación del estado.

A continuación se adjuntan las imágenes de la verificación de las diferentes funciones del sistema:

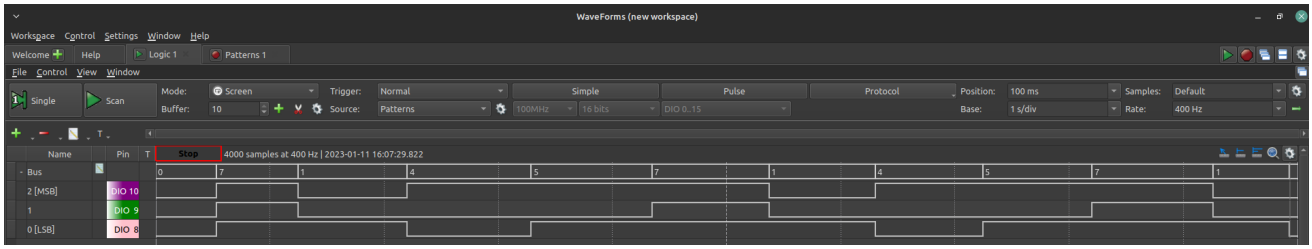


Figura 7: Verificación Analog Discovery - Estado: Cargar dato.

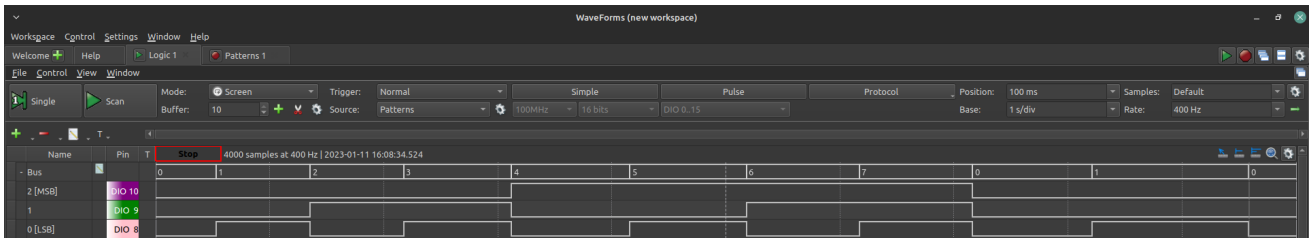


Figura 8: Verificación Analog Discovery - Estado: Contador Ascendente.

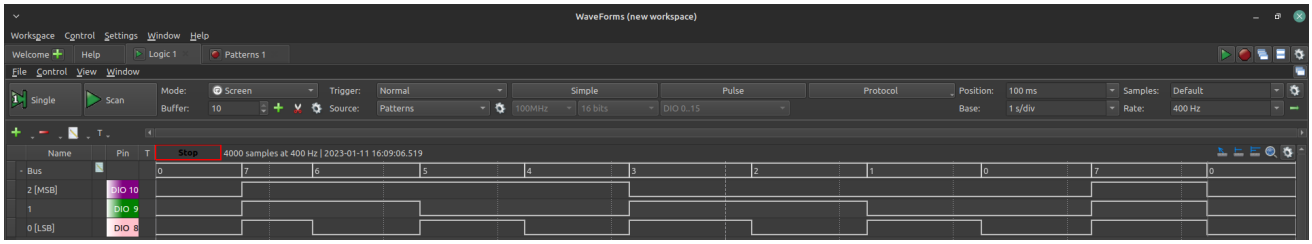


Figura 9: Verificación Analog Discovery - Estado: Contador Descendente.

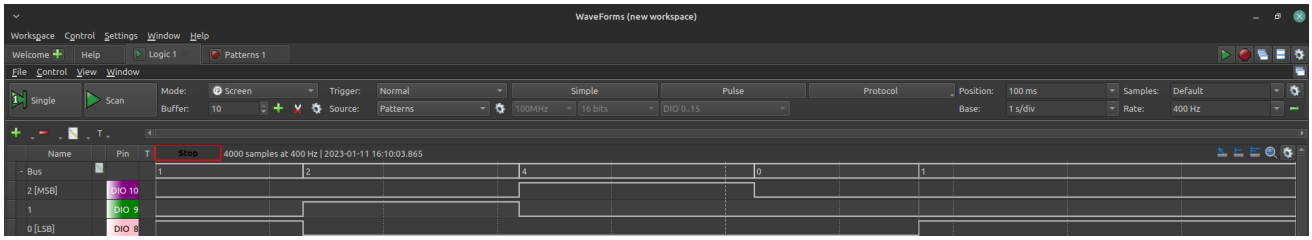


Figura 10: Verificación Analog Discovery - Estado: Desplazamiento a la izquierda.

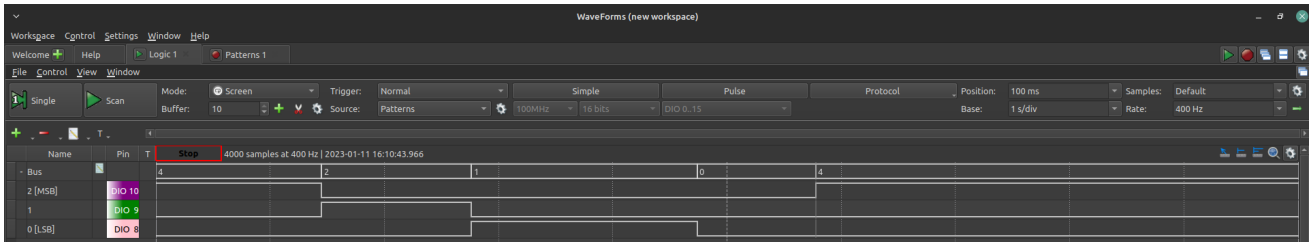


Figura 11: Verificación Analog Discovery - Estado: Desplazamiento a la derecha.

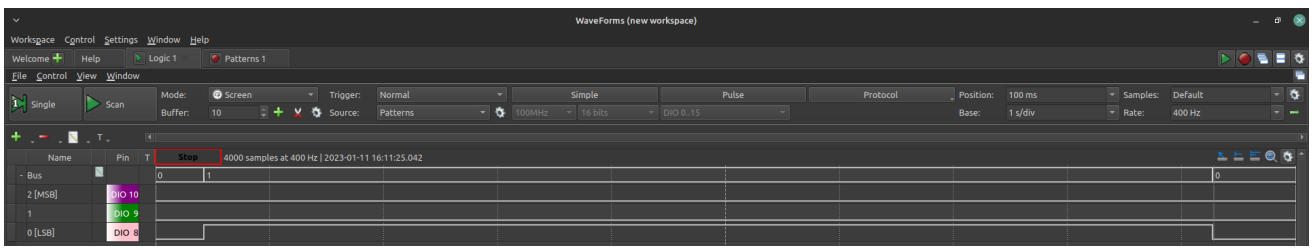


Figura 12: Verificación Analog Discovery - Estado: Conservar dato.

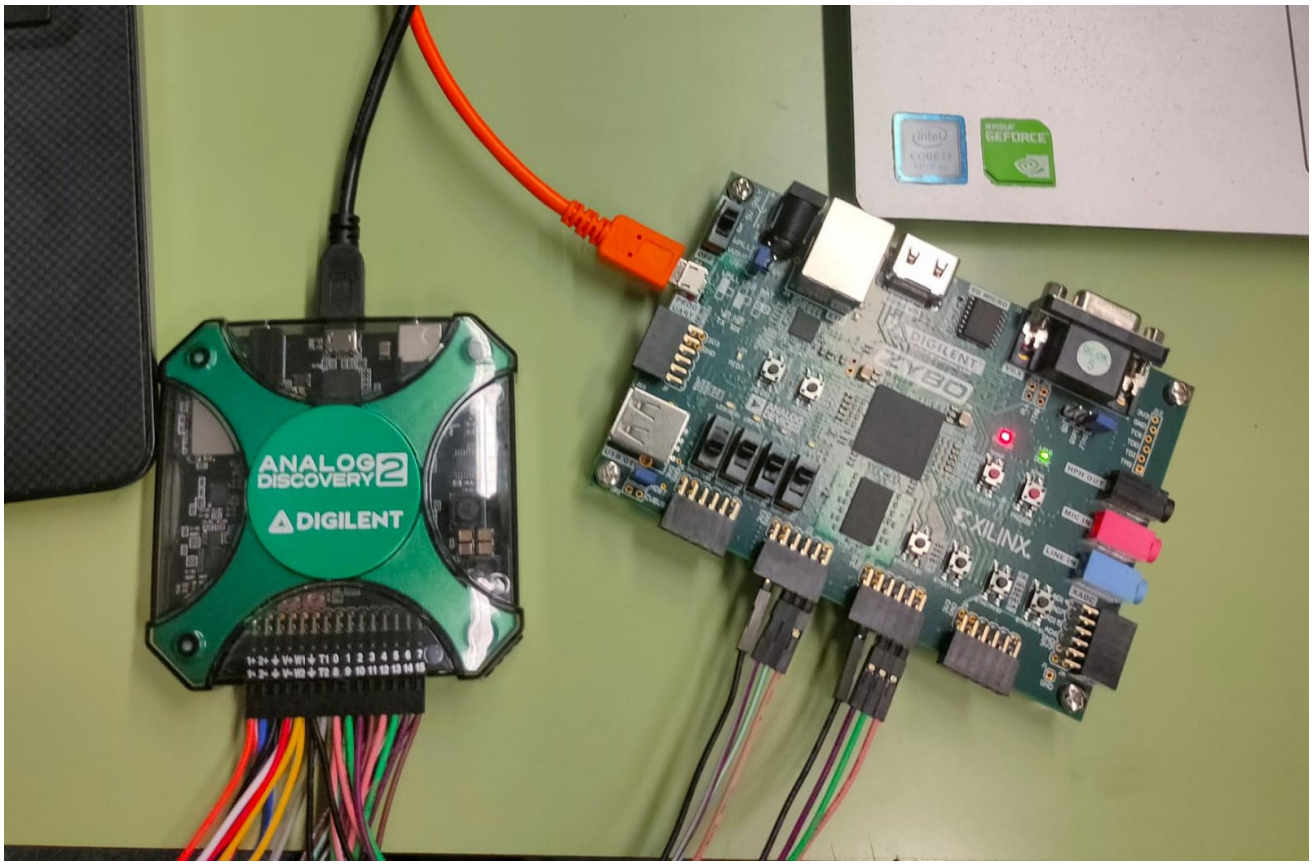


Figura 13: Montaje para verificación del registro universal genérico con Analog Discovery

Por lo tanto, se concluyen las fases de diseño, simulación y síntesis del registro universal genérico en VHDL satisfactoriamente con las funcionalidades requeridas sin usar el paquete *numeric_std*.

# 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
```

```
7   -- Module Name: registro
8   -- Project Name: practica2
9   -- Target Devices: Zybo
10  -- Tool Versions: Vivado 2022.1
11  -- Description: Universal register with several operations without numeric library.
12  --
13  -- 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_0: std_logic_vector(n-1 downto 0):= (others=> '0');
59      variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
60      variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');
61
62    begin
63      internal_numA := numA;
64      internal_numB := numB;
65      for i in n-1 downto 0 loop
66        carry := internal_numA and internal_numB;
67        internal_numA := internal_numA xor internal_numB;
68        internal_numB := my_Shift_left(carry);
69      end loop;
70      return internal_numA;
71    end function;
72
73    function Bitwise_Subs(numA, numB:std_logic_vector) return std_logic_vector is
74      variable borrow: std_logic_vector(n-1 downto 0):= (others=> '0');
75      variable internal_0: std_logic_vector(n-1 downto 0):= (others=> '0');
76      variable internal_numA: std_logic_vector(n-1 downto 0):= (others=> '0');
77      variable internal_numB: std_logic_vector(n-1 downto 0):= (others=> '0');
78    begin
79      internal_numA := numA;
80      internal_numB := numB;
81
82      for i in n-1 downto 0 loop
83        borrow := ((not internal_numA) and internal_numB);
84        internal_numA := internal_numA xor internal_numB;
85        internal_numB := my_Shift_left(borrow);
```

```
86      end loop;
87      return internal_numA;
88    end function;
89  ----------------------------------------------------------------------------
90  begin
91    Secuencial: process (clk, reset, control)
92    begin
93      if reset = '1' then
94        d_aux <= (others => '0');
95      elsif rising_edge( clk ) and cke = '1' then
96        if control = "000" then           -- Carga
97          d_aux <= d;
98        elsif control = "001" then        -- Cuenta ascendente
99          d_aux <= Bitwise_Add(d_aux,number_1);
100       elsif control = "010" then        -- Cuenta descendente
101         d_aux <= Bitwise_Subs(d_aux,number_1);
102       elsif control = "011" then        -- Desplazamiento izquierda
103         d_aux <= my_Shift_left(d_aux);
104       elsif control = "100" then        -- Desplazamiento derecha
105         d_aux <= my_Shift_right(d_aux);
106       elsif control = "101" or control = "110" or control = "111" then -- Mantiene
107         d_aux <= d_aux;
108       end if;
109     end if;
110   end process;
111
112   q <= d_aux;
113 end behavioral;
```

Código 10: Entidad del top

## 7.3. Test Bench

```
1   ----------------------------------------------------------------------------
2   -- Company: Universidad de Málaga
3   -- Engineer: Izan Amador, Jorge L. Benavides
4   --
5   -- Create Date: 23.11.2022 17:47:41
6   -- Design Name: Debouncer
7   -- Module Name: Test_Bench - Behavioral
8   -- Project Name: Sincronizador
9   -- Target Devices: Zybo
10  -- Tool Versions: Vivado 2022.1
11  -- Description: Debouncer for a button.
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  ----------------------------------------------------------------------------
19
20
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23  use STD.textIO.ALL;                     -- Se va a hacer uso de ficheros.
24
25  entity Test_Bench is
26  --  Port ( );
27  end Test_Bench;
28
29  architecture Comportamiento of Test_Bench is
30
31    component registro
32      generic (n : integer := 8);
33      port(d       : in  std_logic_vector(n-1 downto 0);
34           control : in  std_logic_vector (2 downto 0);
35           clk     : in  std_logic;
36           cke     : in  std_logic;
37           reset   : in  std_logic;
38           q       : out std_logic_vector(n-1 downto 0));
39    end Component registro;
```

```
40
41
42    constant semiperiodo    : time    := 10 ns;
43    constant tiempo_control : time    := 50 ns;
44    constant n              : integer := 3;
45
46
47    signal d_interno, q_interno       : std_logic_vector(n-1 downto 0) := (others => 'U');
48    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