

# Ingeniería de Computadores III

Curso 2020-2021

Entrega de Abril

---

*María Esther Ramos Iglesias. mramos1238*

*email: [mramos1238@alumno.uned.es](mailto:mramos1238@alumno.uned.es)*

*Centro asociado: UNED Aranjuez*

# Ejercicio 1

En el ejercicio 1 se va a trabajar en el diseño de un circuito digital que implemente las funciones F1 y F2 a partir de la siguiente tabla de verdad.

x	y	z	F1	F2
'0'	'0'	'0'	'0'	'0'
'0'	'0'	'1'	'1'	'0'
'0'	'1'	'0'	'0'	'0'
'0'	'1'	'1'	'0'	'1'
'1'	'0'	'0'	'1'	'0'
'1'	'0'	'1'	'1'	'1'
'1'	'1'	'0'	'1'	'1'
'1'	'1'	'1'	'1'	'1'

Imagen 1: Tabla de verdad de las funciones F1 y F2

## Apartado a)

En el primer apartado se pide implementar la **entity** del circuito. Declaramos los puertos que serán visibles desde el exterior, que en este caso se corresponden con las entradas y salidas de la tabla de verdad.

```
--ejercicio1_a.vhdl
library IEEE;
use IEEE.std_logic_1164.all;

Entity ejercicio1_a is
  port(
    x,y,z: in std_logic;
    F1: out std_logic;
    F2: out std_logic);
end entity ejercicio1_a;
```

### Apartado b)

Para este apartado desarrollamos una **architecture** que va a definir el comportamiento del circuito para la **entity** creada previamente.

Nuestro primer paso será crear las funciones booleanas que definen el circuito utilizando para ello [Karnaugh](#).

$$F1 = x + \bar{y}z$$

$$F2 = x(y+z) + yz$$

```
--ejercicio1_b.vhdl
library IEEE;
use IEEE.std_logic_1164.all;

architecture ejercicio1_b of ejercicio1_a is
begin
    --F1: x + ((not y)z)
    --F2: x( y + z ) + yz
    F1 <= x or ((not y) and z);
    F2 <= (x and (y or z)) or (y and z);
end architecture ejercicio1_b;
```

### Apartado c)

Si diseñamos el circuito obtenido en el apartado b) obtenemos la siguiente figura

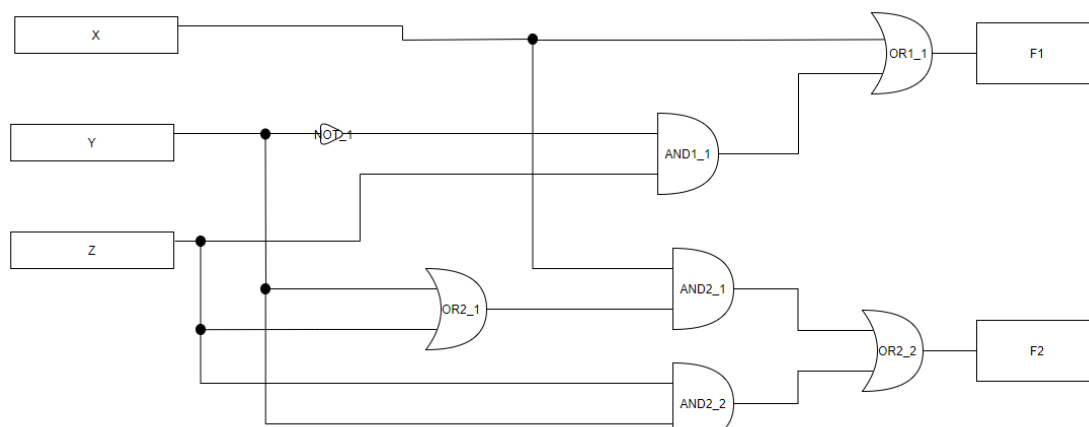


Imagen 2: Diseño del Circuito

Una vez tenemos el circuito terminado y sabemos el tipo de puertas lógicas necesitamos en el proyecto, procedemos a crear su *entity* y *architecture*

```
--ejercicio1_c.vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity not1 is    -- Puerta not: 1 entrada.
    port(
        x: in std_logic;
        F: out std_logic );
end entity not1;

architecture not1 of not1 is
begin
    F <= not x;
end architecture not1;

library IEEE;
use IEEE.std_logic_1164.all;

entity and1 is    -- Puerta lógica and: 2 entradas.
    port(
        x,y: in std_logic;
        F: out std_logic);
end entity and1;

architecture and1 of and1 is
begin
    F <= x and y;
end architecture and1;

library IEEE;
use IEEE.std_logic_1164.all;

entity or1 is    -- Puerta lógica or: 2 entradas.
    port(
        x,y: in std_logic;
        F: out std_logic );
end entity or1;

architecture or1 of or1 is
begin
    F <= x or y;
end architecture or1;
```

### Apartado d)

Escribimos en código una **architecture** que describa la estructura del circuito utilizando las puertas lógicas creadas en el apartado c). Para ello utilizaremos señales auxiliares

Para las señales se ha seguido la estructura sP1\_X o sP2\_X :

- P : Inicial de la puerta lógica a utilizar
- X : Identificador numérico (Para diferenciar sí se usa la misma puerta lógica ).

```
--ejercicio1_d.vhdl
library IEEE;
use IEEE.std_logic_1164.all;
architecture ejercicio1_d of ejercicio1_a is

    component not1 is
        port(
            x: in std_logic;
            F: out std_logic);
    end component not1;

    component and1 is
        port(
            x,y: in std_logic;
            F: out std_logic
        );
    end component and1;

    component or1 is
        port(
            x,y: in std_logic;
            F: out std_logic);
    end component or1;

    signal s02_1, sA2_1, sA2_2, s02_2, sN1_1, sA1_1, s01_1 : std_logic;
begin
    N1_1: component not1 port map (y, sN1_1);
    A1_1: component and1 port map (sN1_1, z, sA1_1);
    O1_1: component or1 port map (x, sA1_1, F1);
    O2_1: component or1 port map (y, z, s02_1);
    A2_1: component and1 port map (x, s02_1, sA2_1);
    A2_2: component and1 port map (y, z, sA2_2);
    O2_2: component or1 port map (sA2_1, sA2_2, F2);
end architecture ejercicio1_d;
```

## Apartado e)

Vamos a comprobar el correcto funcionamiento de las *architecture* creadas y para ello hemos creado una batería de pruebas.

```
--ejercicio1_e.vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity ejercicio1_e is
end entity ejercicio1_e;

architecture ejercicio1_e of ejercicio1_e is
    signal x,y,z : std_logic;
    signal F1,F2: std_logic;

    component ejercicio1_a is
        port (
            x,y,z: in std_logic;
            F1,F2: out std_logic);
    end component ejercicio1_a;
begin
    uut: component ejercicio1_a port map (x,y,z,F1,F2);
    test : process
    begin
        x <= '0', '1' after 40 ns ;
        y <= '0', '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
        z <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns,
        '0' after 40 ns, '1' after 50 ns, '0' after 60 ns, '1' after 70 ns ;
        wait;
    end process test;
end architecture ejercicio1_e;
```

Con las pruebas ya creadas nos salen los siguientes cronogramas:



# Ejercicio 2

En este ejercicio nos piden desarrollar un circuito combinacional con 4 entradas de distintos tipos y longitudes y una única salida.

## Apartado a)

Con lo descrito en el enunciado desarrollamos la siguiente **entity**:

```
--ejercicio2_b.vhdl
entity ejercicio2_a is
  generic( N: integer := 8);
  Port (
    NUM1: OUT std_logic_vector(N downto 0);
    X, Y : IN std_logic_vector (N-1 downto 0);
    sel : IN std_logic_vector (1 downto 0);
    E : IN std_logic);
end ejercicio2_a;
```

- Los vectores X e Y tendrán una longitud de 8.
- El vector **sel** tendrá una longitud de 2.
- El vector NUM1 tendrá una longitud de 9.

La arquitectura será la siguiente:

```
--ejercicio2_a.vhdl
architecture ejercicio2_a of ejercicio2_a is
  begin
    circuito : process(X,Y,sel)
      variable aux_count: integer;      --
      Variable a utilizar como contador de unos
      variable aux_concat: std_logic_vector(15 downto 0); --
      Variable para concatenar X e Y
      variable aux_x: std_logic_vector(N-1 downto 0); --
      Variable que realizará la suma de X e Y

--ejercicio2_a.vhdl
  begin
    if (E='0') then --E=0
      NUM1 <= "000000000";
```

```

else --E=1
    case sel is
        when "00" => --
Contaremos el número de unos que hay en X e Y concatenando ambas variables.

        aux_concat(15 downto 8) := X;
        aux_concat(N-1 downto 0) := Y;
        aux_count:=0;
        for i in 0 to 15 loop --
Creamos un bucle que recorra todos los números de la variable auxiliar
            if (aux_concat(i) = '1') then
                aux_count := aux_count + 1; --
Si la variable tiene un uno, sumamos uno al contador.
            end if;
        end loop;
        NUM1 <= std_logic_vector(to_unsigned(aux_count, 9));
--Asignamos a NUM1 el número de unos en binario.
        when "01" => --
Contamos si X e Y son pares. Los valores que puede tomar NUM1 son 0 (ambos impares) 1 (uno par otro impar) o 2 (ambos pares)
            if ((unsigned(X) mod 2) = 0) then --
> Hacemos la comprobación mediante ifs anidados.
                if ((unsigned(Y) mod 2) = 0) then
                    NUM1 <= std_logic_vector(to_signed(2,9));
                else
                    NUM1 <= std_logic_vector(to_signed(1,9));
                end if;
            else
                if ((unsigned(Y) mod 2) = 0) then
                    NUM1 <= std_logic_vector(to_signed(1,9));
                else
                    NUM1 <= std_logic_vector(to_signed(0,9));
                end if;
            end if;
        when "10" => --
Realizamos la operación and sobre X e Y y la almacenamos en NUM1
        NUM1(N) <= '0';
        NUM1(N-
1 downto 0) <= std_logic_vector(signed(X) and signed(Y));
        when others => --
Sumamos con signo X e Y y lo almacenamos en NUM1
        aux_x := std_logic_vector(signed(X) + signed(Y));
        NUM1 <= std_logic_vector(resize(signed(aux_x), N+1));
    end case;
end if;
end process circuito;
end architecture ejercicio2_a;

```



## Apartado b)

Para el apartado b se ha realizado un banco de pruebas. Para ello hemos creado un **procedure** dentro de la **architecture** que compare el resultado obtenido mediante el apartado a) con el esperado según el procedure.

```
--ejercicio2_b.vhdl
procedure check_ALU
( X, Y : in std_logic_vector (7 downto 0);
  sel : in std_logic_vector (1 downto 0);
  E : in std_logic;
  actual_NUM1: in std_logic_vector (8 downto 0);
  error_count: inout integer) is
  variable expected_NUM1 : std_logic_vector(8 downto 0);
  variable aux_count : integer;
begin
  case E is
    when '0' =>
      expected_NUM1 := std_logic_vector(to_signed(0,9)); --
      - NUM1 debe valor 0 mientras E sea 0.
    when others =>
      if (sel="00") then
        aux_count:=0;
        for i in 0 to 7 loop --
          -- Creamos un bucle que recorra todos los números de la variable auxiliar
          if (X(i) = '1') then
            aux_count := aux_count + 1; --
            -- Si la variable tiene un uno, sumamos uno al contador.
          end if;
          if (Y(i) = '1') then
            aux_count := aux_count + 1; --
            -- Si la variable tiene un uno, sumamos uno al contador.
          end if;
        end loop;
        expected_NUM1(8 downto 5) := "0000";
        expected_NUM1(4 downto 0) := std_logic_vector(to_signed(aux_count,5));
      end if;
      if (sel="01") then
        if ((unsigned(X) mod 2) = 0) then --
          -- > Hacemos la comprobación mediante ifs anidados.
          if ((unsigned(Y) mod 2) = 0) then
            expected_NUM1 := "0000000010";
          else
            expected_NUM1 := "0000000001";
          end if;
        else

```

```

        if ((unsigned(Y) mod 2) = 0) then
            expected_NUM1 := "000000001";
        else
            expected_NUM1 := "000000000";
        end if;
    end if;
    if (sel="10") then
        expected_NUM1 :=std_logic_vector(resize((signed(X) and
d signed(Y)),9));
    end if;
    if (sel="11") then
        expected_NUM1 :=std_logic_vector(resize(((signed(X))
+(signed(Y)),9));
    end if;
end case;
if (expected_NUM1 /= actual_NUM1) then
    report "ERROR: Valor esperado: " & integer'image(to_integer(s
igned(expected_NUM1))) & " en binario. Valor obtenido: " & integer'image(
to_integer(signed(NUM1)));
    error_count := error_count + 1;
end if;
end procedure check_ALU;

```

Y a su vez declaramos las pruebas que haremos. Como queremos abarcar todos los posibles valores de las señales de entrada hemos utilizado tres bucles de tipo for anidados:

```

--ejercicio2_b.vhdl

begin
    uut: component ejercicio2_a port map (NUM1,X,Y,sel,E);
    test : process is
        variable error_count: integer:=0;
    begin
        report "Comienza la simulacion";
        E <= '0';
        for g in 0 to 3 loop
            for i in -2**7 to 2**7-1 loop
                for j in -2**7 to 2**7-1 loop
                    sel <= std_logic_vector(to_unsigned(g,2)); --
Bucle sel 00|01|10|11
                    X <= std_logic_vector(to_signed(i,8)); --Bucle X
                    Y <= std_logic_vector(to_signed(j,8)); --Bucle Y
                    wait for 10 ns;
                    check_ALU(X,Y,sel,E,NUM1,error_count);
                end loop;
            end loop;
        end loop;
    end process;
end

```

```
        end loop;
    end loop;
    E <= '1';
    for g in 0 to 3 loop
        for i in -2**7 to 2**7-1 loop
            for j in -2**7 to 2**7-1 loop
                sel <= std_logic_vector(to_unsigned(g,2)); --
Bucle sel 00|01|10|11
                X <= std_logic_vector(to_signed(i,8)); --Bucle X
                Y <= std_logic_vector(to_signed(j,8)); --Bucle Y
                wait for 10 ns;
                check_ALU(X,Y,sel,E,NUM1,error_count);
            end loop;
        end loop;
    end loop;
```

- E=1. Sel=0-1. Y=0-1, X=0-1-2

- E=1. Sel=2-3. Y=0-1, X=0-1-2

- E=0. Sel=2-3. Y=0-1, X=0-1-2

[illegible]

Si lanzamos el código nos dará distintos mensajes dependiendo de si acaba con errores o sin errores.

```
VSIM 105> run
# ** Note: Comienza la simulacion
# Time: 0 ps Iteration: 0 Instance: /ejercicio2_b
# ** Note: Finaliza la simulacion sin errores
# Time: 1310730 ns Iteration: 0 Instance: /ejercicio2_b
```

```
# ** Note: ERROR: Valor esperado: 3 en binario. Valor obtenido: 1
# Time: 983040 ns Iteration: 0 Instance: /ejercicio2_b
# ** Note: ERROR: Valor esperado: 1 en binario. Valor obtenido: 0
# Time: 983050 ns Iteration: 0 Instance: /ejercicio2_b
# ** Note: Finaliza la simulacion:16384errores
# Time: 1310730 ns Iteration: 0 Instance: /ejercicio2_b
```

```
if (error_count=0) then
    report "Finaliza la simulacion sin errores";
else
    report "Finaliza la simulacion: " & integer'image(error_c
ount) & " errores";
end if; -- Termina la simulación
```