Universidad Nacional de Educación a Distancia

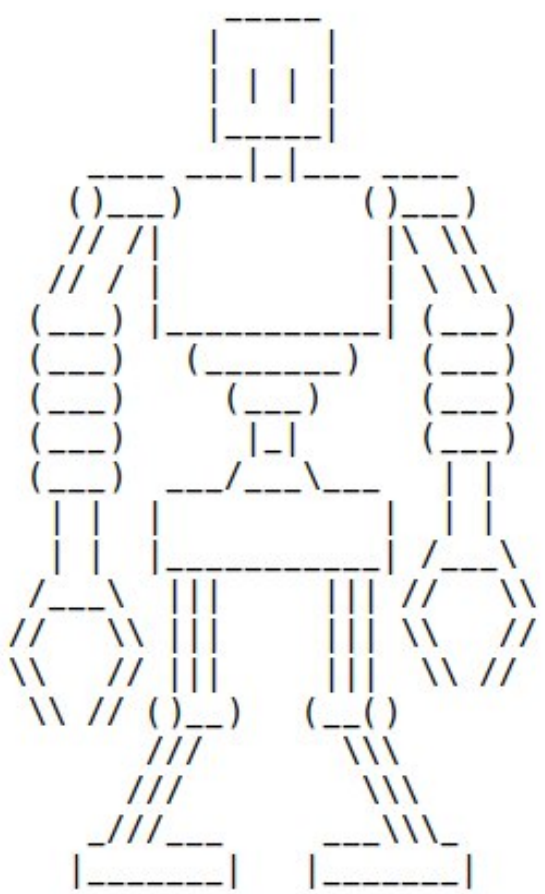
Grado en Ingeniería Informática

Trabajo Práctico - Convocatoria ordinaria

Curso 2024-2025

Ingeniería de Computadores III

Natalia Solana



Índice

1 Ejercicio 1

En este ejercicio se pide diseñar un circuito digital que implemente las funciones F y G mostradas a continuación, que dependen de las tres variables x, y y z:

$$F = x\,\overline{y}\,\overline{z} + \overline{x} + x\,y\,\overline{z}$$

$$G = x\,y + \overline{x}\,z + y\,z$$

1.1 Circuito simple

Escribimos la `entity` para ambas señales, esta `entity` servirá además para la `architecture` pedida en el ejercicio 1.c.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4
5  -- esta entity sirve para la architecture ej1b y ej1c
6  entity ej1 is
7  port( F, G : out std_logic;
8  x, y, z : in std_logic);
9  end entity ej1;
```

La siguiente `architecture` describirá el comportamiento del circuito, como se pide en el apartado 1.b:

```
1     library IEEE;
2     use IEEE.std_logic_1164.all;
3
4     architecture estructura1 of ej1 is
5     begin
6
7     F <= (x and (not y) and (not z)) or (x and y and (not z));
8     G <= (x and y) or ((not x) and z) or (y and z);
9
10    end architecture estructura1;
```

1.2 Circuito con implementación de F y G a nivel de puertas lógicas

Dibujamos un circuito que implemente estas dos funciones lógicas a nivel de puertas lógicas.

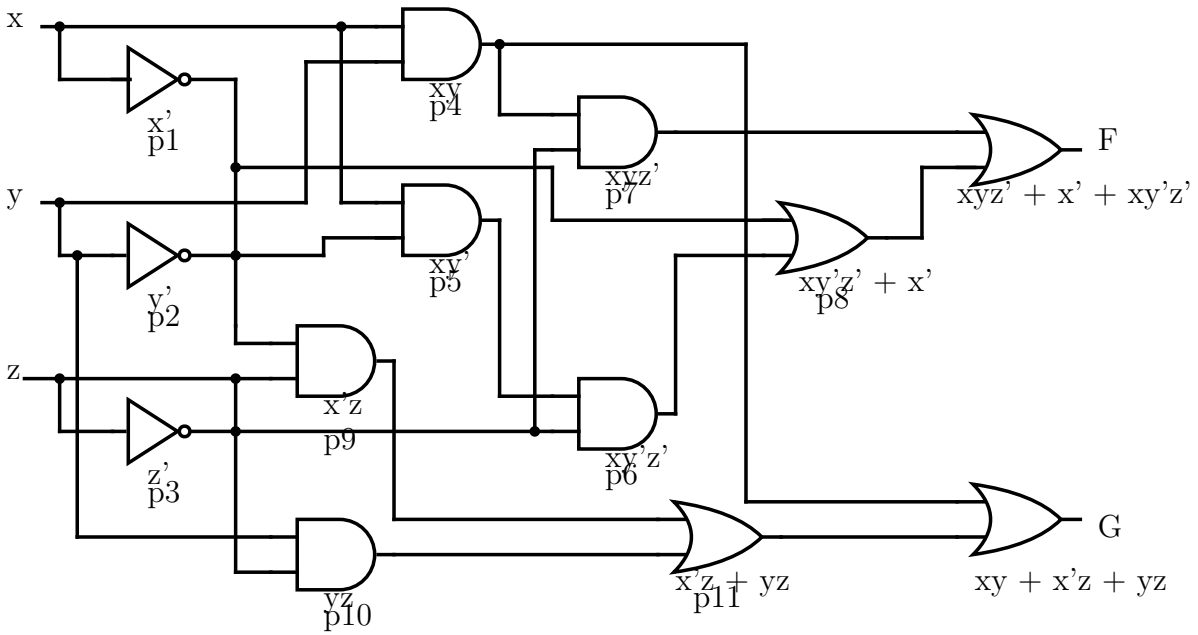


Figure 1: Diagrama Circuito Ejercicio 1

Escribimos la `entity` y `architecture` de cada una de las puertas lógicas. Solo tendremos que describir tres tipos de puerta: `AND`, `OR` y `NOT`.

```
1
2   -- PUERTA AND
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity AND_gate is
7     port (
8       A, B : in std_logic;
9       Y : out std_logic
10    );
11  end entity AND_gate;
12
13  architecture estructuraAND of AND_gate is
14  begin
15      Y <= A and B;
16  end architecture estructuraAND;
```

```vhdl
-- PUERTA OR
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity OR_gate is
  port (
    A, B : in std_logic;
    Y : out std_logic
  );
end entity OR_gate;

architecture estructuraOR of OR_gate is
begin
  Y <= A or B;
end architecture estructuraOR;
```

```vhdl
-- PUERTA NOT
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NOT_gate is
  port (
    A: in std_logic;
    Y : out std_logic
  );
end entity NOT_gate;

architecture estructuraNOT of NOT_gate is
begin
  Y <= not A;
end architecture estructuraNOT;
```

Utilizamos las puertas lógicas que hemos diseñado como componentes para escribir una `architecture` que describa el comportamiento del circuito:

```vhdl
  use IEEE.std_logic_1164.all;

  architecture estructura2 of ej1 is

  signal p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11 : std_logic;

  component NOT_gate
  port (
  A : in std_logic;
  Y : out std_logic
```

```
11      );
12      end component;
13
14      component AND_gate
15      port (
16      A, B : in std_logic;
17      Y : out std_logic
18      );
19      end component;
20
21      component OR_gate
22      port (
23      A, B : in std_logic;
24      Y : out std_logic
25      );
26      end component;
27
28      begin
29
30      -- p1 <= (not x);
31      U1: NOT_gate port map (A => x, Y => p1);
32
33      -- p2 <= (not y);
34      U2: NOT_gate port map (A => y, Y => p2);
35
36      -- p3 <= (not z);
37      U3: NOT_gate port map (A => z, Y => p3);
38
39
40      -- p4 <= (x and y);
41      U4: AND_gate port map (A => x, B => y, Y => p4);
42
43      -- p5 <= (x and p2);
44      U5: AND_gate port map (A => x, B => p2, Y => p5);
45
46      -- p6 <= (p5 and p3);
47      U6: AND_gate port map (A => p5, B => p3, Y => p6);
48
49      -- p7 <= (p4 and p3);
50      U7: AND_gate port map (A => p4, B => p3, Y => p7);
51
52      -- p8 <= (p6 or p1);
53      U8: OR_gate port map (A => p6, B => p1, Y => p8);
54
55
56      -- p9 <= (p1 and z);
57      U9: AND_gate port map (A => p1, B => z, Y => p9);
```

```
58
59      -- p10 <= (y and z);
60      U10: AND_gate port map (A => y, B => z, Y => p10);
61
62
63      -- F <= (p7 or p8);
64      U11: OR_gate port map (A => p7, B => p8, Y => F);
65
66
67      -- p11 <= (p10 or p9);
68      U12: OR_gate port map (A => p10, B => p9, Y => p11);
69
70      -- G <= (p4 or p11);
71      U13: OR_gate port map (A => p4, B => p11, Y => G);
72
73      end architecture estructura2;
```

1.3 Banco de pruebas

Finalmente el `testbench` para comprobar los dos diseños es el siguiente:

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   entity testbench is
5   end entity testbench;
6
7   architecture tb of testbench is
8
9   -- Declaracion de sennales
10  signal x,y,z  : std_logic;
11  signal F, G : std_logic;
12
13  -- Seleccion de la entidad
14  component ej1 is
15  port( F, G : out std_logic;
16  x, y, z : in std_logic);
17  end component;
18
19  begin
20
21  UUT : ej1 port map (F => F, G => G, x => x, y => y, z => z);
22
23  vect_test: process
24
25  begin
26  x <= '0'; y <= '0'; z <= '0'; wait for 200 ns;
```

```
27    x <= '0'; y <= '0'; z <= '1'; wait for 200 ns;
28    x <= '0'; y <= '1'; z <= '0'; wait for 200 ns;
29    x <= '0'; y <= '1'; z <= '1'; wait for 200 ns;
30    x <= '1'; y <= '0'; z <= '0'; wait for 200 ns;
31    x <= '1'; y <= '0'; z <= '1'; wait for 200 ns;
32    x <= '1'; y <= '1'; z <= '0'; wait for 200 ns;
33    x <= '1'; y <= '1'; z <= '1'; wait for 200 ns;
34
35    end process vect_test;
36
37    -- Verificacion de las salidas
38    verif : process
39    variable error_status : boolean;
40    begin
41
42    wait on x, y, z;
43    wait for 100 ns;
44
45    if ((x = '0' and y = '0' and z = '0' and F = '1' and G = '0') or
46    (x = '0' and y = '0' and z = '1' and F = '1' and G = '1') or
47    (x = '0' and y = '1' and z = '0' and F = '1' and G = '0') or
48    (x = '0' and y = '1' and z = '1' and F = '1' and G = '1') or
49    (x = '1' and y = '0' and z = '0' and F = '1' and G = '0') or
50    (x = '1' and y = '0' and z = '1' and F = '0' and G = '0') or
51    (x = '1' and y = '1' and z = '0' and F = '1' and G = '1') or
52    (x = '1' and y = '1' and z = '1' and F = '0' and G = '1'))
53    then
54    error_status := false;
55    else
56    error_status := true;
57    end if;
58
59    assert not error_status report "Test fallado." severity note;
60    end process verif;
61    end architecture tb;
```

Este `testbench` provocará un mensaje en la pantalla en el caso de haber algún error en el test.

1.4 Simulación

La simulación enseña los siguientes datos:

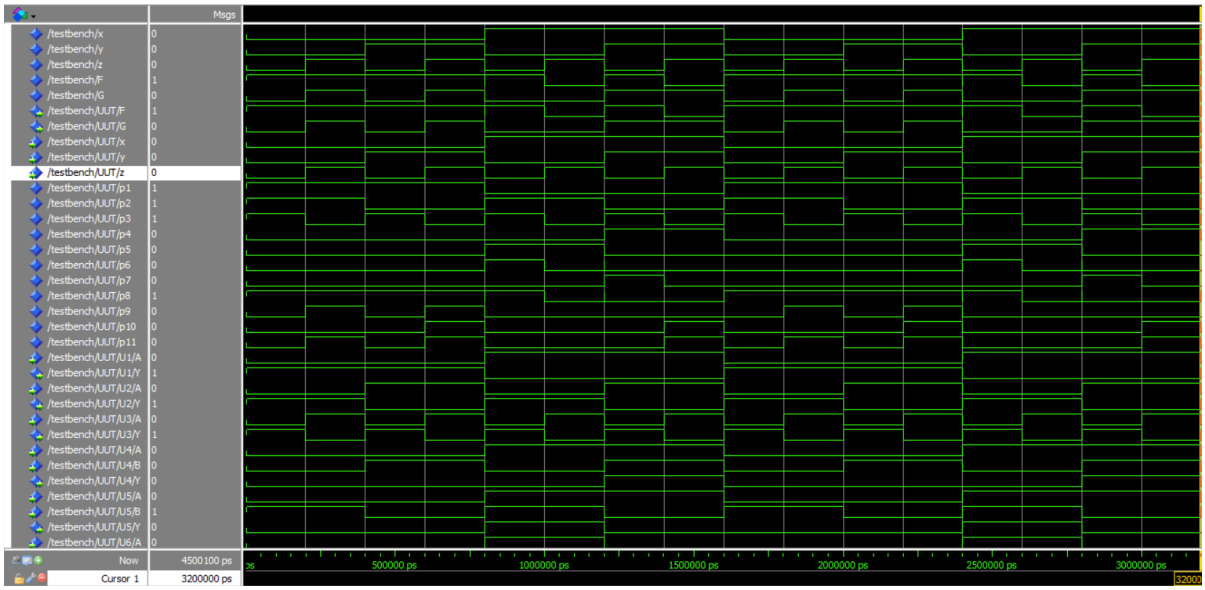


Figure 2: Simulación del Ejercicio 1

El `testbench` no muestra ningún error en la consola, además, al comparar la simulación con la tabla de verdad vemos que los diseños funcionan correctamente:

x	y	z	F	G
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	1

Table 1: Tabla de verdad para las funciones F y G

2 Ejercicio 2

Se desea diseñar en VHDL un circuito combinacional que compare dos números binarios sin signo, x e y, de N bits, donde N es un parámetro genérico ($N \geq 1$). Ambos números se representan mediante señales de tipo `std_logic_vector`. El circuito produce dos salidas: `gout`, que indica si x es mayor que y, y `eout`, que señala si ambos números son iguales. La señal `gout` toma el valor '1' únicamente cuando $x > y$, mientras que `eout` es '1' sólo si $x = y$.

El diseño se plantea de forma modular e iterativa, utilizando un bloque comparador de 1 bit como componente base. A partir de este módulo, se construyen comparadores de mayor tamaño.

2.1 Comparador de dos números de un bit

El diagrama del circuito es el siguiente:

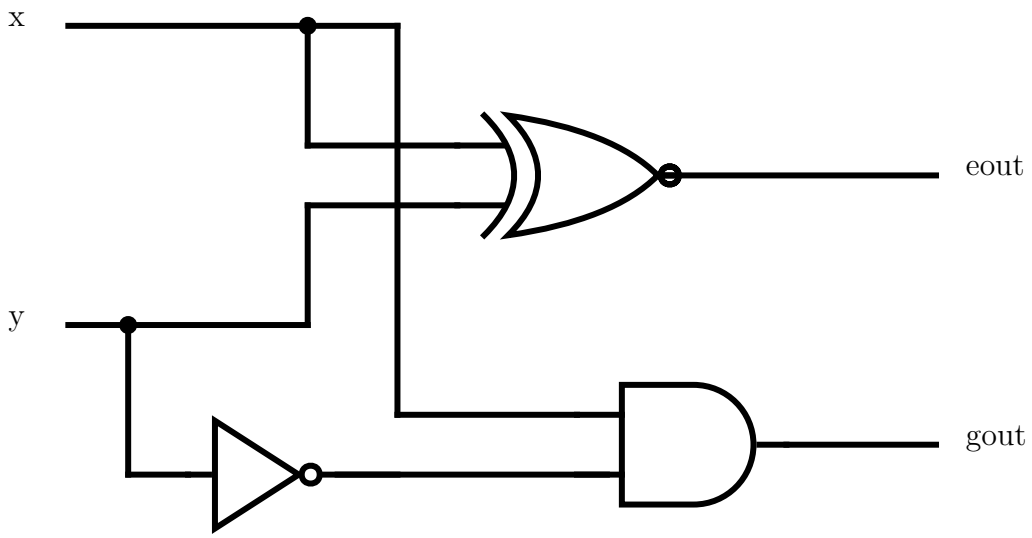


Figure 3: Diagrama Circuito Ejercicio 2

Escribimos la tabla de verdad y las funciones lógicas que describen el comportamiento de cada señal de salida.

x	y	eout	gout
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	0

Table 2: Tabla de verdad para las funciones eout y gout

$$eout = \overline{x \oplus y}$$

$$gout = x * \overline{y}$$

Escribimos la `entity` y `architecture` de cada una de las puertas lógicas. Solo tendremos que describir tres tipos de puerta: `AND`, `XNOR` y `NOT`.

```
1
2   -- PUERTA XNOR
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity XNOR_gate is
7     port (
8       A, B : in std_logic;
9       Y : out std_logic
10    );
11  end entity XNOR_gate;
12
13  architecture estructuraXNOR of XNOR_gate is
14  begin
15    Y <= A xnor B;
16  end architecture estructuraXNOR;
```

```
1
2   -- PUERTA AND
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity AND_gate is
7     port (
8       A, B : in std_logic;
9       Y : out std_logic
10    );
11  end entity AND_gate;
12
```

```
architecture estructuraAND of AND_gate is
begin
  Y <= A and B;
end architecture estructuraAND;
```

```
-- PUERTA NOT
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NOT_gate is
  port (
    A: in std_logic;
    Y : out std_logic
  );
end entity NOT_gate;

architecture estructuraNOT of NOT_gate is
begin
  Y <= not A;
end architecture estructuraNOT;
```

Utilizamos las puertas lógicas que hemos diseñado como componentes para escribir el código vhdl del comparador de dos números de un bit:

```
library IEEE;
use IEEE. std_logic_1164.all;

entity comp1bit is port
( eout, gout : out std_logic; -- gout a 1 si x>y, eout a 1 si x = y
x, y : in std_logic); -- bits a comparar
end entity comp1bit;

architecture comp1bit of comp1bit is
begin
eout <= (x xnor y); -- eout=1 si x=y, eot=0 en caso contrario
gout <= x and (not y); -- gout =1 si x=1, y =0, gout=0 en otro caso
end architecture comp1bit;
```

2.2 Comparador de dos números de N bits

El comparador de N bits utiliza el comparador de un bit a modo de componente.
Se crean dos variables: `p` que corresponde a `eout` y `g` que corrresponde a `gout`. Se asume que en un principio ambos números son iguales, por tanto `p=1`, `g=0`. La lógica de comparación funciona con un bucle desde N-1 hasta 0, en el que primero comprueba si `g=1`, en ese caso `gout=1` y `eout=0` y se sale del bucle. En el caso de que `g=0`, comprueba si `p` es igual a cero. Si es así, `gout=0`, `eout=0` y sale del bucle.

```
1   library IEEE;
2   use IEEE. std_logic_1164.all;
3
4   entity compNbits is
5   generic (N: integer :=4);
6
7   port ( eout, gout : out std_logic;
8   x, y : in std_logic_vector(N-1 downto 0));
9   end entity compNbits;
10
11  architecture compNbits of compNbits is
12  signal p: std_logic_vector(N-1 downto 0):= (others => '1');
13  -- Variable interna para eout, se asume que x e y son iguales
14  signal g: std_logic_vector(N-1 downto 0):= (others => '0');
15  -- Variable interna para gout, se asume que x no es mayor que y al inicio
16
17  component comp1bit is port
18  ( eout, gout : out std_logic; -- gout a 1 si x>y, eout a 1 si x = y
19  x, y : in std_logic); -- bits a comparar
20  end component comp1bit;
21
22  begin
23
24  assert N >= 1 report "N tiene que ser mayor o igual que 1"
25  severity failure; -- si N es menor que 1 la simulacion termina
26
27
28  conexion: for i in N-1 downto 0 generate
29  compNbits: comp1bit
30  port map(
31  eout => p(i),
32  gout => g(i),
33  x => x(i),
34  y => y(i)
35  );
36  end generate conexion;
37
38  process(x, y, p, g)
```

```
39      begin
40      eout <= '1';
41      gout <= '0';
42
43      for i in N-1 downto 0 loop
44      -- va desde el bit mas significativo hasta el menos
45      if g(i) = '1' then
46      eout <= '0'; -- Si hay algun bit mayor que otro entonces eout = 0
47      gout <= '1';
48      exit;
49
50      elsif p(i) = '0' then
51      eout <= '0';
52      exit;
53      end if;
54      end loop;
55      end process;
56      end architecture compNbits;
```

2.3 Banco de pruebas

El banco de pruebas testea todas las posibles entradas al circuito y muestra al final de la simulación un mensaje con el número total de errores producidos y los detalles de cada error. Para ello, se utiliza una función llamada `to_string` que convierte un dato en `std_logic` a un `string`, el cual sí se puede mostrar en pantalla con el resto de datos.

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4
5   entity compNbits_tb is
6   end entity compNbits_tb;
7
8
9
10  architecture compNbits_tb of compNbits_tb is
11
12  function to_string(s: std_logic) return string is
13  --convierte std_logic en string para poder imprimir las salidas
14  begin
15  if s = '0' then
16  return "0";  -- Devuelve '0' como string
17  else
18  return "1";  -- Devuelve '1' como string
19  end if;
20  end function;
```

```
21
22  constant N : integer := 4;
23  signal eout, gout : std_logic; -- Conectar salidas UUT
24  signal x, y : std_logic_vector(3 downto 0); -- Conectar entradas UUT
25
26  component compNbits is
27  generic ( N: integer := 4);
28  port ( eout, gout : out std_logic;
29  x, y : in std_logic_vector(N-1 downto 0));
30  end component compNbits;
31
32  begin
33  -- Instanciar
34  uut : component compNbits port map
35  ( eout => eout, gout => gout, x => x, y => y );
36
37  gen_vec_test : process
38  variable num_errores : integer := 0; -- Numero de errores
39
40  begin
41  for op_X in 0 to (2**N)-1 loop
42  for op_Y in 0 to (2**N)-1 loop
43  x <= std_logic_vector(to_unsigned(op_x,N));
44  y <= std_logic_vector(to_unsigned(op_y,N));
45  wait for 100 ns;
46
47  -- Comprueba resultado
48  if (x=y and eout='0') or (x/=y and eout='1')
49  or (x>y and gout='0') or (x<y and gout ='1') then
50
51  report "Error. Valores: x=" &
52  integer'image(op_x) &
53  ", y=" &
54  integer'image(op_y) &
55  ". Salidas: gout=" & to_string(gout) &
56  ", eout=" & to_string(eout);
57  num_errores := num_errores + 1;
58  end if;
59  end loop;
60  end loop;
61
62  report "Test completo. Hay " &
63  integer'image(num_errores) & "errores." ;
64  wait; --Final simulacion
65  end process gen_vec_test;
66  end architecture compNbits_tb;
```

2.4 Simulación

La simulación no muestra ningún error al terminar.

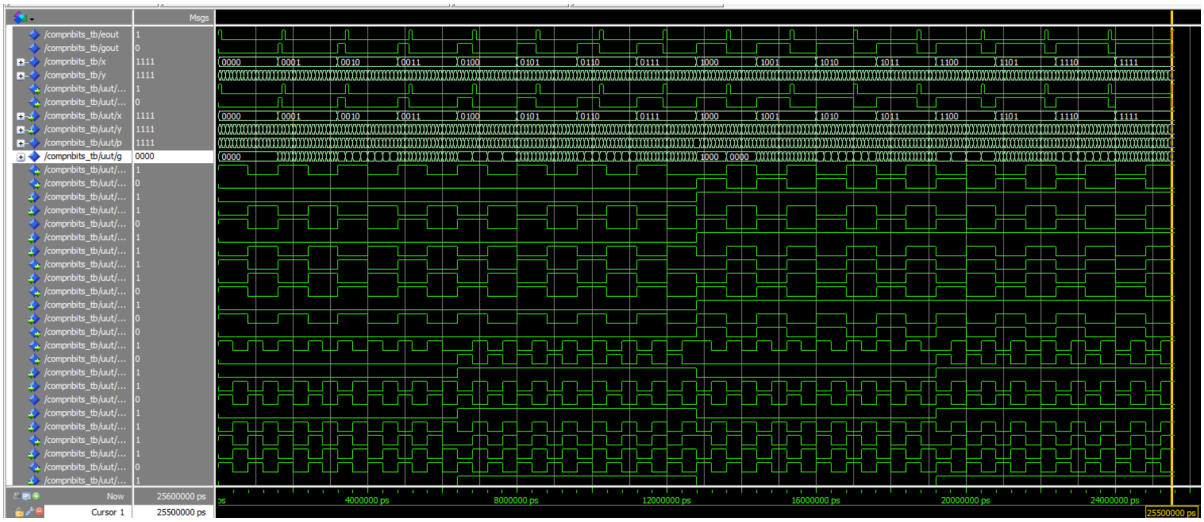


Figure 4: Simulación del Ejercicio 2