Ingeniería de Computadores 3

Alumno : Diego Díaz

Contenido

Introducción 1
1. Solución. Ejercicio 1. 2
1.a. Entity del circuito que implementa las dos funciones lógicas. 2
1.b. Architecture que describe el comportamiento del circuito. 2
1.c. Dibujo del diagrama del circuito con puertas lógicas en Figura 1. 3
Entity y Architecture de cada una de las puertas lógicas que se ha diseñado. 3
1.d. Architecture que describe la estructura del circuito. 4
1.e. Banco de pruebas y cronograma en Figura 2. 5
2. Solución. Ejercicio 2. 6
2.a. Codificador de prioridad 4 a 2. 6
2.a.1. Diagrama del circuito codificador con prioridad 8 a 2 en la Figura 3. 7
2.a.2. Entity y architecture de cada una de las puertas lógicas que componen el circuito. 7
2.a.3. Entity y architecture del circuito. 8
2.b. Banco de pruebas para el *codif_P4a2* y cronograma en Figura 4. 9
2.c. Codificador de prioridad de 8 a 3. 12
2.c.1. Diagrama del circuito codificador con prioridad 8 a 3 en Figura 5. 12
2.c.2. Entity y architecture del circuito. 13
2.d. Banco de pruebas para el *codif_P8a3* y cronograma en Figura 6. 14

Introducción

En este documento se responde a los ejercicios obligatorios de la asignatura de Ingeniería de computadores 3 de junio 2016, en los que se practica con código VHDL y el simulador ModelSim.

El enunciado de estos dos ejercicios prácticos se encuentra en el documento "enunciadoTrabajoIC3_jun16.pdf". Se adjunta junto a esta memoria, el código junto a los ficheros generados por ModelSim y todas las imágenes de los circuitos y cronogramas.

1. Solución. Ejercicio 1.

Se desea diseñar un circuito digital que implemente las funciones F y G cuya tabla de verdad se muestra en la **Tabla 1.**

X	Y	Z	F	G
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

Tabla1. Tabla de verdad.

Las funciones F y G son tratadas en las respuestas en su forma lógica simplificada que se representa como una suma de productos:

F = yz
G = y + (nx)z

1.a. Entity del circuito que implementa las dos funciones lógicas.

Código VHDL. Entity del circuito.[1]

```
entity funcFG is
  port ( F, G    : out std_logic;
         x, y, z : in std_logic );
end entity funcFG;
```

1.b. Architecture que describe el comportamiento del circuito.

Código VHDL. Architecture comportamiento del circuito.[1]

```
architecture Comp of funcFG is
begin
   F <= (y and z);
   G <= (not x and z) or (y);
end architecture Comp;
```

[1] Este código se encuentra en el archivo *funcionFG.vhd* de la carpeta Ejercicio1.Comportamiento.

1.c. Dibujo del diagrama del circuito con puertas lógicas en Figura 1.
Entity y Architecture de cada una de las puertas lógicas que se ha diseñado.

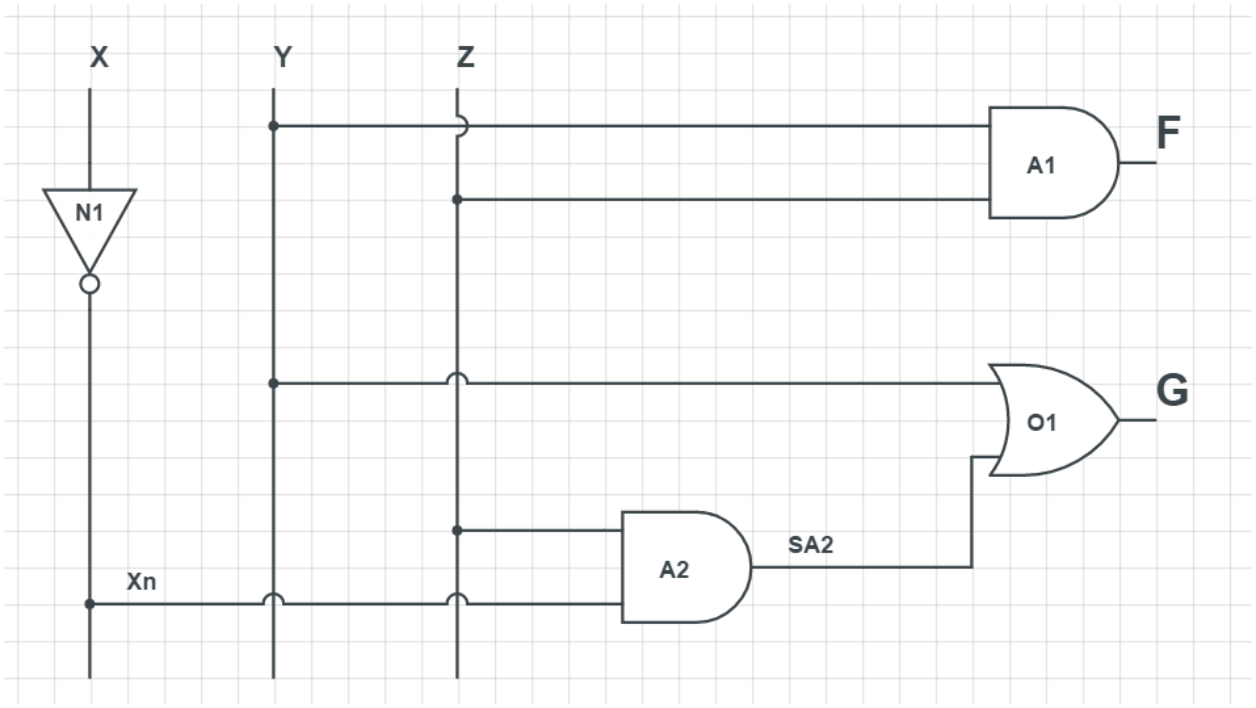

Figura 1. Diagrama al nivel de puertas lógicas de las funciones F y G.

Código VHDL. Entity y Architecture puerta digital AND de dos entradas.[2]

```
entity and2 is
  port ( y0    : out std_logic;
         x0, x1 : in std_logic );
end entity and2;

architecture and2 of and2 is
   begin
      y0 <= x0 and x1;
end architecture and2;
```

[2] Este código se encuentra en el archivo *logic_and.vhd* de la carpeta Ejercicio1.Estructura.

Código VHDL. Entity y Architecture puerta digital OR de dos entradas.[3]

```
entity or2 is
   port ( y0    : out std_logic;
          x0, x1 : in std_logic );
end entity or2;

architecture or2 of or2 is
   begin
      y0 <= x0 or x1;
end architecture or2;
```

Código VHDL. Entity y Architecture puerta digital NOT.[4]

```
entity not1 is
   port ( y0 : out std_logic;
            x0 : in std_logic );
end entity not1;

architecture not1 of not1 is
   begin
      y0 <= not x0;
end architecture not1;
```

1.d. Architecture que describe la estructura del circuito.

Código VHDL. Architecture del circuito describiendo su estructura.[5]

```
architecture circuito_Estruc of funcFG is
   signal   xn    : std_logic;
   signal   sA2   : std_logic;

-- Declaración de las clases de los componentes
   component and2 is
      port ( y0    : out std_logic;
             x0, x1 : in std_logic);
   end component and2;

   component not1 is
      port ( y0 : out std_logic;
               x0 : in std_logic );
   end component not1;
```

[3] Este código se encuentra en el archivo *logic_or.vhd* de la carpeta Ejercicio1.Estructura.
[4] Este código se encuentra en el archivo *logic_not.vhd* de la carpeta Ejercicio1.Estructura.
[5] Este código se encuentra en el archivo *funcionFG.vhd* de la carpeta Ejercicio1.Estructura.

```
component or2 is
      port ( y0    : out std_logic;
             x0, x1 : in std_logic );
   end component or2;

begin
-- Instanciación y conexión de los componentes

   N1 : component not1 port map (xn, x);
   A2 : component and2 port map (sA2, xn, z);
   A1 : component and2 port map (F, y, z);
   O1 : component or2 port map (G, y, sA2);

end architecture circuito_Estruc;
```

1.e. Banco de pruebas y cronograma en Figura 2.

Código VHDL. Banco de pruebas del circuito.[6]

```
-- Banco de pruebas del circuito Ejercicio 1
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity bp_funcFG is
        constant DELAY : time := 20 ns; -- Retardo usado en el test
end entity bp_funcFG;

architecture bp_funcFG of bp_funcFG is
        signal F, G    : std_logic;
        signal x, y, z : std_logic;

        component funcFG is
           port ( F, G : out std_logic;
                  x, y, z : in std_logic );
        end component funcFG;

begin
        UUT : component funcFG port map (F, G, x, y, z);

        vec_test : process is
                variable valor : unsigned (2 downto 0);
        begin
                -- Generar todos los posibles valores de entrada
                for i in 0 to 7 loop
                        valor := to_unsigned(i,3);
                        x <= std_logic(valor(2));
```

[6] Este código se encuentra en el archivo *bp_funcionFG.vhd* (en ambos proyectos del ejercicio1).

```
                    y <= std_logic(valor(1));
                    z <= std_logic(valor(0));
                    wait for DELAY;
                end loop;
                wait; -- Final de la simulación
        end process vec_test;
end architecture bp_funcFG;
```

Se comprueba que las dos arquitecturas funcionan correctamente con el banco de pruebas de este apartado (el cual es compartido).

En la **figura 2** se muestra el cronograma resultante al realizar la simulación del banco de pruebas. Dicho cronograma coincide en los dos proyectos de este ejercicio y visualmente se observa que el resultado es correcto.

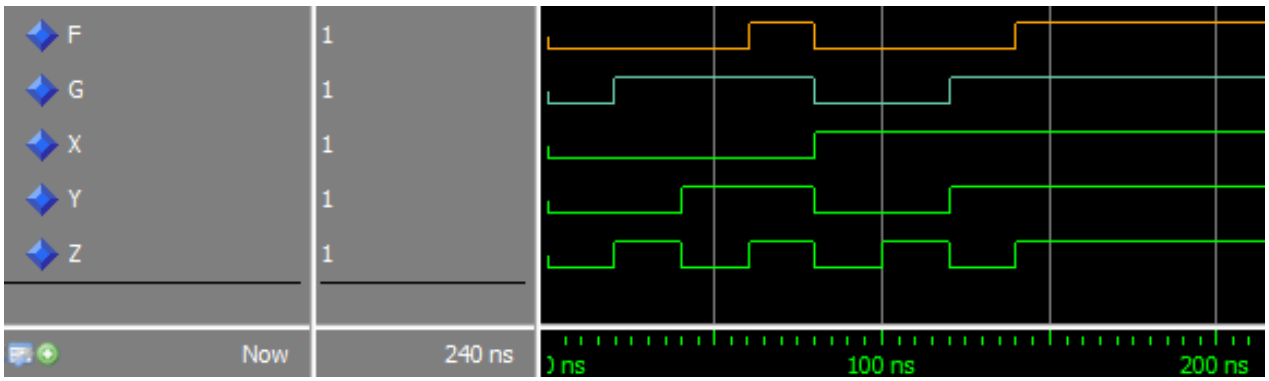

Figura 2. Cronograma resultante al realizar la simulación.

2. Solución. Ejercicio 2.

Este ejercicio se divide en dos partes, en la primera parte se trata un codificador con prioridad 4 a 2 (apartados a y b) y en la segunda parte se trata un codificador con prioridad 8 a 3 (apartados c y d).

2.a. Codificador de prioridad 4 a 2.

Se pretende diseñar un codificador de prioridad 4 a 2 empleando únicamente puertas AND de dos entradas, puertas OR de 2 entradas e inversores. El circuito codificador ha de tener una señal de entrada x de 4 bits y dos señales de salida, código de 2 bits y activo de 1 bit. El comportamiento del circuito viene descrito por la **tabla 2.**

X (3 : 0)				Código (1:0)		Activo
X(3)	**X(2)**	**X(1)**	**X(0)**	**C(1)**	**C(0)**	
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	-	0	1	1
0	1	-	-	1	0	1
1	-	-	-	1	1	1

Tabla 2. Tabla de verdad de un codificador con prioridad 4 a 2.

Las salidas código(1), código(0) y Activo son tratadas en las respuestas en su forma lógica simplificada que se representa como una suma de productos:

Código(1) = x(3) + x(2)
Código(0) = x(3) + nx(2) x(1)
Activo = x(3) + x(2) + x(1) + x(0)

2.a.1. Diagrama del circuito codificador con prioridad 8 a 2 en la **Figura 3.**

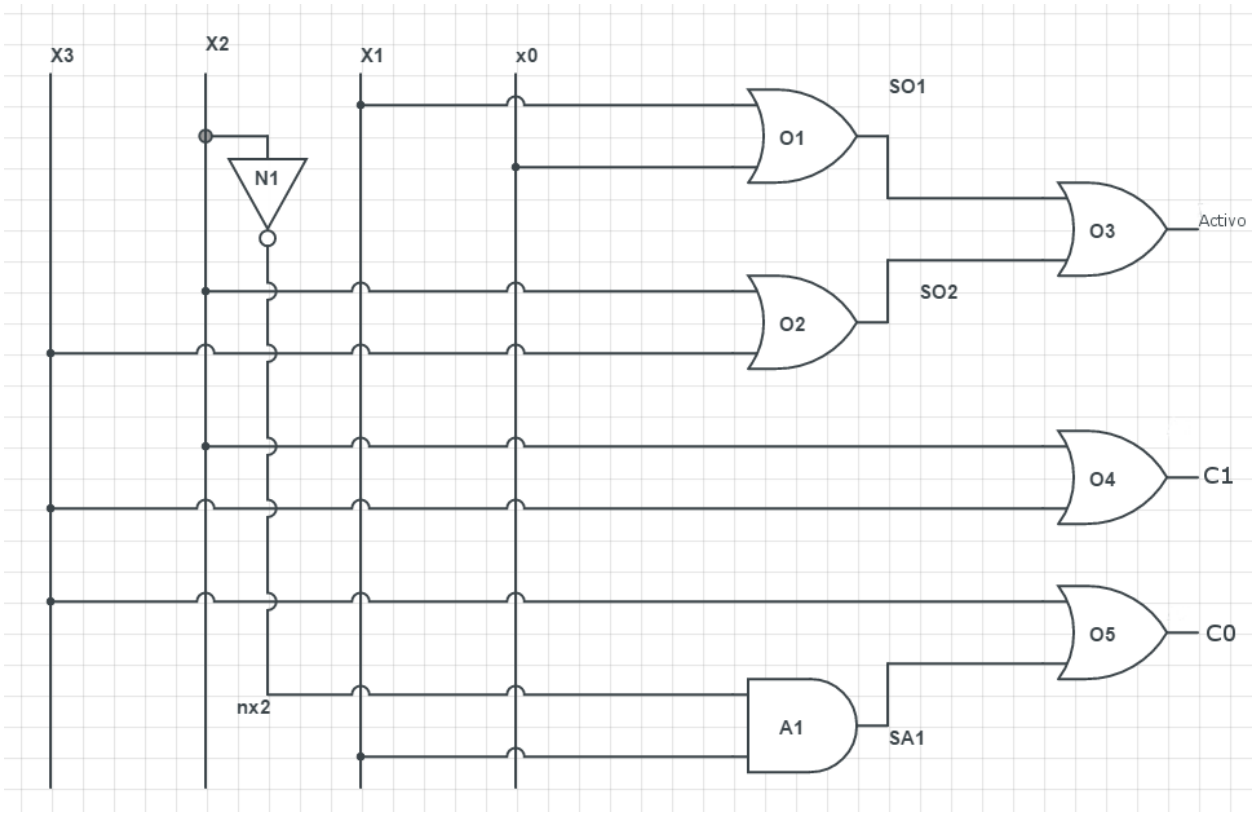

Figura 3. Diagrama del circuito codificador con prioridad 4 a 2 con puertas lógicas.

2.a.2. Entity y architecture de cada una de las puertas lógicas que componen el circuito.

Código VHDL. Entity y Architecture puerta digital AND de dos entradas.[7]

```
entity and2 is
  port ( y0    : out std_logic;
         x0, x1 : in std_logic );
end entity and2;

architecture and2 of and2 is
   begin
      y0 <= x0 and x1;
end architecture and2;
```

[7] Este código se encuentra en el archivo *logic_and.vhd* de la carpeta Ejercicio2.Codif_p4a2.

Código VHDL. Entity y Architecture puerta digital OR de dos entradas.[8]

```
entity or2 is
   port ( y0    : out std_logic;
          x0, x1 : in std_logic );
end entity or2;

architecture or2 of or2 is
   begin
      y0 <= x0 or x1;
end architecture or2;
```

Código VHDL. Entity y Architecture puerta digital NOT.[9]

```
entity not1 is
   port ( y0 : out std_logic;
            x0 : in std_logic );
end entity not1;

architecture not1 of not1 is
   begin
      y0 <= not x0;
end architecture not1;
```

2.a.3. Entity y architecture del circuito.

Código VHDL. Entity y Architecture del circuito describiendo su estructura.[10]

```
entity codif_P4a2 is
      port ( codigo  : out std_logic_vector(1 downto 0);
               activo  : out std_logic;
               x       : in std_logic_vector(3 downto 0) );
end entity codif_P4a2;

architecture codif_P4a2 of codif_P4a2 is
   signal   nx2              : std_logic;
   signal   sO1, sO2, sA1  : std_logic;

-- Declaracion de las clases de los componentes

   component and2 is
      port ( y0    : out std_logic;
             x0, x1 : in std_logic);
   end component and2;
```

[8] Este código se encuentra en el archivo *logic_or.vhd* de la carpeta Ejercicio2.Codif_p4a2.
[9] Este código se encuentra en el archivo *logic_not.vhd* de la carpeta Ejercicio2.Codif_p4a2.
[10] Este código se encuentra en el archivo *codif_P4a2.vhd* de la carpeta Ejercicio2.Codif_p4a2.

```
   component not1 is
      port ( y0 : out std_logic;
             x0 : in std_logic );
   end component not1;

   component or2 is
      port ( y0    : out std_logic;
             x0, x1 : in std_logic );
   end component or2;

begin

-- Instanciacion y conexion de los componentes

   N1 : component not1 port map (nx2, x(2));
   A1 : component and2 port map (sA1, nx2, x(1));
   O1 : component or2 port map (sO1, x(1), x(0));
   O2 : component or2 port map (sO2, x(2), x(3));
   O3 : component or2 port map (activo, sO1, sO2);
   O4 : component or2 port map (codigo(1), x(2), x(3));
   O5 : component or2 port map (codigo(0), x(3), sA1);

end architecture codif_P4a2;
```

2.b. Banco de pruebas para el *codif_P4a2* y cronograma en Figura 4.

Código VHDL. Banco de pruebas del circuito.[11]

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

entity bp_codif_P4a2 is
        constant MAX_COMB : integer := 16; -- Num. combinac. entrada (2**4)
        constant DELAY : time := 20 ns; -- Retardo usado en el test
end entity bp_codif_P4a2;



architecture bp_codif_P4a2 of bp_codif_P4a2 is
-- Salidas UUT
        signal activo : std_logic;
        signal codigo : std_logic_vector(1 downto 0);
-- Entradas UUT
        signal x : std_logic_vector(3 downto 0);
```

[11] Este código se encuentra en el archivo *bp_codif_P4a2.vhd* de la carpeta Ejercicio2.Codif_p4a2.

```
        component codif_P4a2 is
                port (  codigo : out std_logic_vector(1 downto 0);
                        activo : out std_logic;
                        x      : in std_logic_vector(3 downto 0) );
        end component codif_P4a2;

begin   -- Cuerpo de la arquitectura
        UUT : component codif_P4a2 port map (codigo,activo,x);

        main : process is

                variable esperado_activo : std_logic;
                variable esperado_codigo : std_logic_vector(1 downto 0);
                variable error_count     : integer := 0;

        begin
                report "Comienza la simulacion";

                -- Generar todos los posibles valores de entrada
                for i in 0 to (MAX_COMB-1) loop
                        x <= std_logic_vector(TO_UNSIGNED(i,4));

                -- Calcular el valor esperado
                        if (i=0) then
                                esperado_activo := '0';
                                esperado_codigo := "00";
                        else
                                esperado_activo := '1';
                                if (i=1) then esperado_codigo := "00";
                                elsif (i<=3) then esperado_codigo := "01";
                                elsif (i<=7) then esperado_codigo := "10";
                                else esperado_codigo := "11";
                                end if;
                        end if;

                        wait for DELAY; -- Espera y compara con las salidas de UUT
                        if ( esperado_activo /= activo ) then
                                report "ERROR en la salida valida. Valor esperado: " &
                                std_logic'image(esperado_activo) &
                                ", valor actual: " &
                                std_logic'image(activo) &
                                " en el instante: " &
                                time'image(now);
                                error_count := error_count + 1;
                        end if;
```

```
                        if ( esperado_codigo /= codigo ) then
                                report "ERROR en la salida codificada. Valor esperado: " &
                                std_logic'image(esperado_codigo(1)) &
                                std_logic'image(esperado_codigo(0)) &
                                ", valor actual: " &
                                std_logic'image(codigo(1)) &
                                std_logic'image(codigo(0)) &
                                " en el instante: " &
                                time'image(now);
                        error_count := error_count + 1;
                        end if;
                end loop; -- Final del bucle for de posibles valores de entrada

                -- Informe del numero total de errores
                report "Hay " &
                integer'image(error_count) &
                " errores.";
                wait; -- Final de la simulacion

        end process main;
end architecture bp_codif_P4a2;
```

En la **figura 4** se muestra el cronograma resultante al realizar la simulación del banco de pruebas. Visualmente se observa que el resultado es correcto comparando con la tabla de verdad.

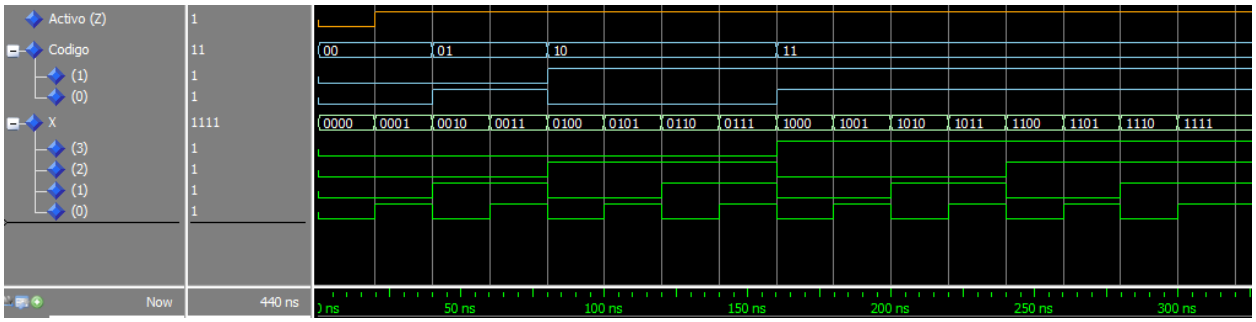

Figura 4. Cronograma resultante de la simulación del codificador con prioridad 4 a 2.

2.c. Codificador de prioridad de 8 a 3.

La tabla de verdad se muestra en la **tabla 3.**

X (7 : 0)								Código (2 :0)			Activo
X(7)	X(6)	X(5)	X(4)	X(3)	X(2)	X(1)	X(0)	A2	A1	A0	Z
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	-	0	0	1	1
0	0	0	0	0	1	-	-	0	1	0	1
0	0	0	0	1	-	-	-	0	1	1	1
0	0	0	1	-	-	-	-	1	0	0	1
0	0	1	-	-	-	-	-	1	0	1	1
0	1	-	-	-	-	-	-	1	1	0	1
1	-	-	-	-	-	-	-	1	1	1	1

Tabla 3. Tabla de verdad de un circuito codificador de prioridad de 8 a 3.

2.c.1. Diagrama del circuito codificador con prioridad 8 a 3 en **Figura 5.**

Para obtener el resultado se emplean los codificadores de prioridad 4 a 2 iguales a los diseñados en el Apartado 2.a y puertas lógicas (AND de dos entradas, OR de dos entradas e inversores). [12]

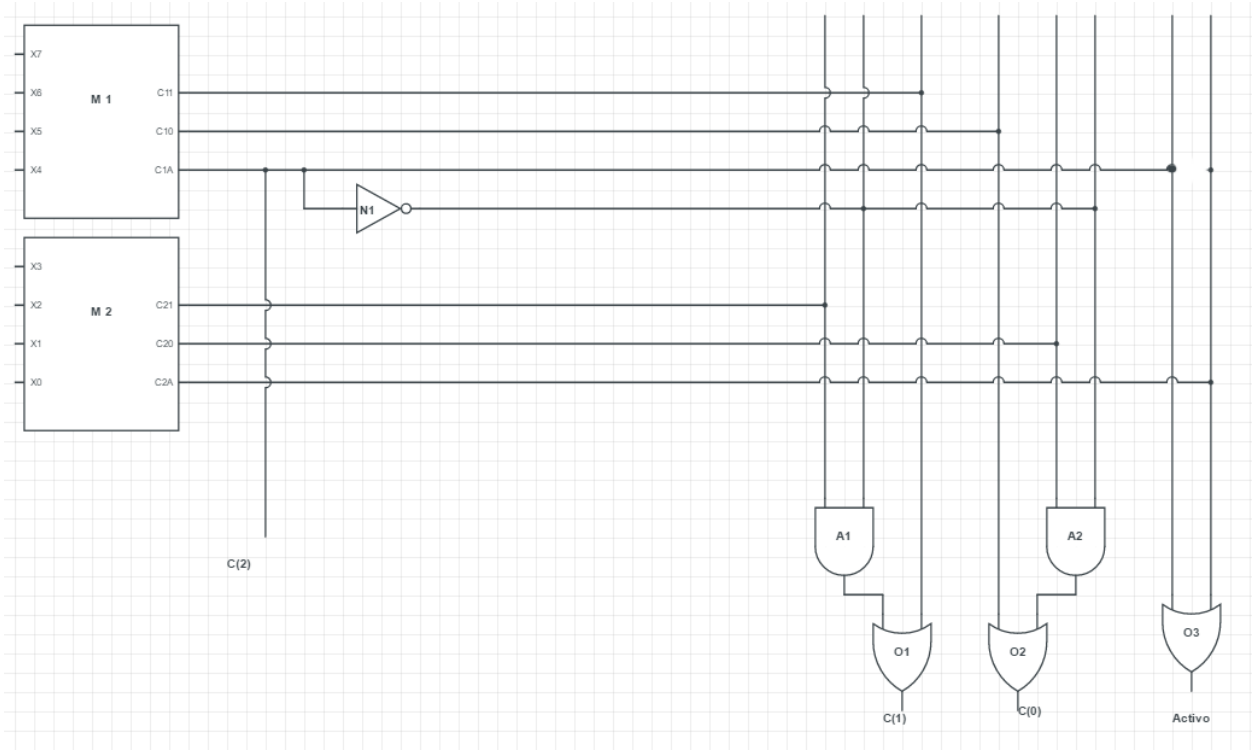

Figura 5. Diagrama del circuito codificador con prioridad 8 a 3 propuesto.

[12] La simplificación del codificador con prioridad 8a3 se obtiene de este documento.

2.c.2. Entity y architecture del circuito.

Código VHDL. Entity y architecture del circuito.[13]

```
entity codif_P8a3 is
      port ( codigo  : out std_logic_vector(2 downto 0);
           activo  : out std_logic;
           x       : in std_logic_vector(7 downto 0) );
end entity codif_P8a3;

architecture codif_P8a3 of codif_P8a3 is
   signal   C1, C2      : std_logic_vector(1 downto 0);
   signal   C1A, C2A : std_logic;
   signal   nA1         : std_logic;
   signal   sA1, sA2  : std_logic;

-- Declaracion de las clases de los componentes

   component and2 is
      port ( y0    : out std_logic;
           x0, x1 : in std_logic);
   end component and2;

   component not1 is
      port ( y0 : out std_logic;
           x0 : in std_logic );
   end component not1;

   component or2 is
      port ( y0    : out std_logic;
           x0, x1 : in std_logic );
   end component or2;

   component codif_P4a2 is
      port ( codigo     : out std_logic_vector(1 downto 0);
           activo     : out std_logic;
           x          : in std_logic_vector(3 downto 0) );
   end component codif_P4a2;

begin
-- Instanciacion y conexion de los componentes
          M1 : component codif_P4a2 port map (C1(1 downto 0), C1A, x(7 downto 4));
          M2 : component codif_P4a2 port map (C2(1 downto 0), C2A, x(3 downto 0));
          N1 : component not1 port map (nA1, C1A);
          A1 : component and2 port map (sA1, nA1, C2(1));
          A2 : component and2 port map (sA2, nA1, C2(0));
```

[13] Este código se encuentra en el archivo *codif_P8a3.vhd* de la carpeta Ejercicio2.Codif_p8a3.

```
        codigo(2) <= C1A;
        O1 : component or2 port map (codigo(1), C1(1), sA1);
        O2 : component or2 port map (codigo(0), C1(0), sA2);
        O3 : component or2 port map (activo, C1A, C2A);

end architecture codif_P8a3;
```

2.d. Banco de pruebas para el *codif_P8a3* y cronograma en Figura 6.

Código VHDL. Banco de pruebas del circuito.[14]

```
-- Banco de pruebas del codificador 8:3 con prioridad
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity bp_codif_P8a3 is
        constant MAX_COMB : integer := 256; -- Num. combinac. entrada (2**8)
        constant DELAY    : time := 10 ns;  -- Retardo usado en el test
end entity bp_codif_P8a3;


architecture bp_codif_P8a3 of bp_codif_P8a3 is
-- Salidas UUT
        signal activo : std_logic;
        signal codigo : std_logic_vector(2 downto 0);
-- Entradas UUT
        signal x : std_logic_vector(7 downto 0);

        component codif_P8a3 is
                port ( codigo : out std_logic_vector(2 downto 0);
                    activo : out std_logic;
                    x      : in std_logic_vector(7 downto 0) );
        end component codif_P8a3;


begin -- Cuerpo de la arquitectura
        UUT : component codif_P8a3 port map (codigo,activo,x);

        main : process is

        variable esperado_activo : std_logic;
        variable esperado_codigo : std_logic_vector(2 downto 0);
        variable error_count     : integer := 0;
```

[14] Este código se encuentra en el archivo *bp_codif_P8a3.vhd* de la carpeta Ejercicio2.Codif_p8a3.

```
begin
        report "Comienza la simulacion";

        -- Generar todos los posibles valores de entrada
        for i in 0 to (MAX_COMB-1) loop
                x <= std_logic_vector(TO_UNSIGNED(i,8));

        -- Calcular el valor esperado
                if (i=0) then
                        esperado_activo := '0';
                        esperado_codigo := "000";
                else
                        esperado_activo := '1';
                        if (i=1) then esperado_codigo := "000";
                        elsif (i<=3) then esperado_codigo := "001";
                        elsif (i<=7) then esperado_codigo := "010";
                        elsif (i<=15) then esperado_codigo := "011";
                        elsif (i<=31) then esperado_codigo := "100";
                        elsif (i<=63) then esperado_codigo := "101";
                        elsif (i<=127) then esperado_codigo := "110";
                        else esperado_codigo := "111";
                        end if;
                end if;

                wait for DELAY; -- Espera y compara con las salidas de UUT
                if ( esperado_activo /= activo ) then
                        report "ERROR en la salida valida. Valor esperado: " &
                        std_logic'image(esperado_activo) &
                        ", valor actual: " &
                        std_logic'image(activo) &
                        " en el instante: " &
                        time'image(now);
                        error_count := error_count + 1;
                end if;

                if ( esperado_codigo /= codigo ) then
                        report "ERROR en la salida codificada. Valor esperado: " &
                        std_logic'image(esperado_codigo(2)) &
                        std_logic'image(esperado_codigo(1)) &
                        std_logic'image(esperado_codigo(0)) &
                        ", valor actual: " &
                        std_logic'image(codigo(2)) &
                        std_logic'image(codigo(1)) &
                        std_logic'image(codigo(0)) &
                        " en el instante: " &
                        time'image(now);
                        error_count := error_count + 1;
```

```
                    end if;
                end loop; -- Final del bucle for de posibles valores de entrada

                -- Informe del numero total de errores
                report "Hay " &
                integer'image(error_count) &
                " errores.";
                wait; -- Final de la simulacion

        end process main;

end architecture bp_codif_P8a3;
```

En la **figura 6** se muestra el cronograma resultante al realizar la simulación del banco de pruebas. Visualmente se observa que el resultado es correcto.

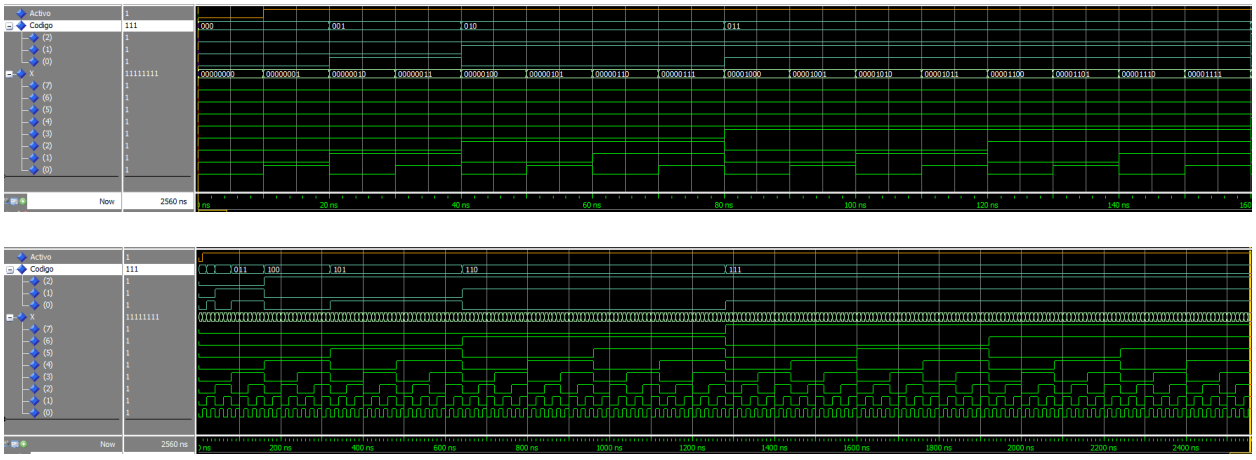

Figura 6. Cronograma resultante al realizar la simulación.