INGENIERÍA DE COMPUTADORES 3

Solución al Trabajo Práctico - Junio de 2017

EJERCICIO 1

Se desea diseñar un circuito digital que implemente las funciones `F` y `G` cuya tabla de verdad se muestra a continuación, que dependen de las tres variables `x`, `y` y `z`:

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

1.a) (0.5 puntos) Obtenga las funciones lógicas `F` y `G` a partir de la tabla de verdad. Escriba en VHDL la **entity** del circuito que implemente las dos funciones lógicas. Es decir, que tenga tres entradas `x`, `y` y `z`, y dos salidas `F` y `G`.

1.b) (1 punto) Escriba en VHDL la **architecture** que describa el *comportamiento* del circuito.

1.c) (1 punto) Dibuje el diagrama de un circuito que implemente estas dos funciones lógicas al nivel de puertas lógicas. No es necesario que el circuito esté simplificado. A continuación, escriba en VHDL la **entity** y la **architecture** de cada una de las puertas lógicas que componen el circuito que acaba de dibujar.

1.d) (1 punto) Escriba en VHDL una **architecture** que describa la *estructura* del circuito que ha dibujado, instanciando y conectando las puertas lógicas que ha diseñado anteriormente.

1.e) (0.5 puntos) Escriba en VHDL un banco de pruebas que permita visualizar, para todos los posibles valores de las entradas, la salida del circuito cuya **entity** ha especificado en el Apartado 1.a. Emplee dicho banco de pruebas para comprobar mediante inspección visual que los dos diseños de los Apartado 1.b y 1.d funcionan correctamente. Incluya en la memoria los dos cronogramas obtenidos al realizar la simulación del banco de pruebas usando en un caso como circuito de test el circuito de Apartado 1.b y en el otro caso el circuito del Apartado 1.d.

Solución al Ejercicio 1

La **entity** del circuito que implementa las dos funciones lógicas se muestra en Código VHDL 1.1. El Código VHDL 1.2 muestra la **architecture** del circuito describiendo su comportamiento.

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

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

Código VHDL 1.1: Solución al Apartado 1.a: **entity** del circuito.

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

Código VHDL 1.2: Solución al Apartado 1.b: **architecture** del circuito describiendo su comportamiento.

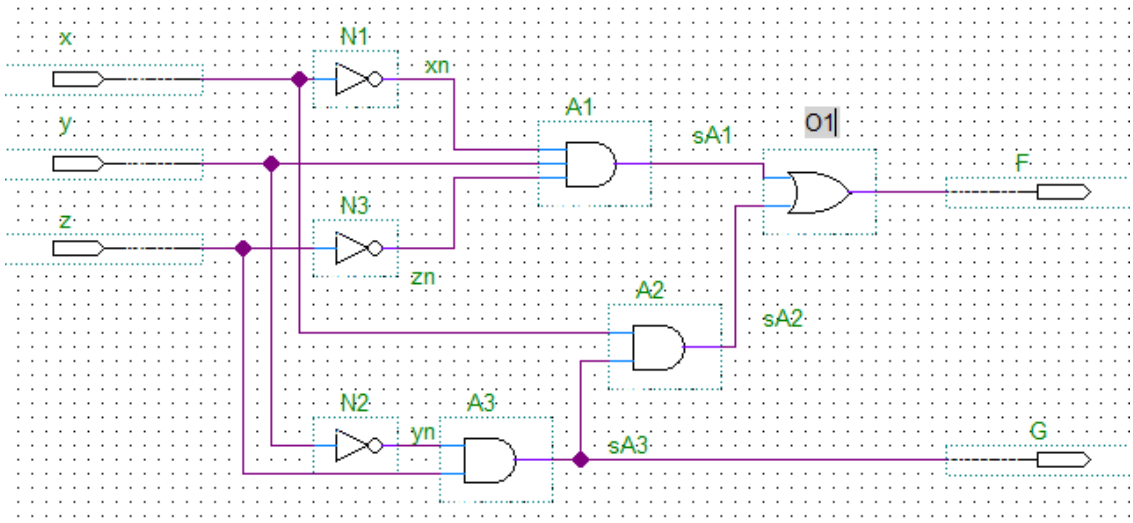

Figura 1.1: Solución al Apartado 1.c: diagrama al nivel de puertas lógicas.

La Figura 1.1 muestra el diagrama del circuito implementado empleando dos puertas AND de dos entradas, una puerta AND de tres entradas, una puerta OR de dos entradas y tres inversores.

El código VHDL de la **entity** y la **architecture** de estas cuatro puertas lógicas se muestra en Código VHDL 1.3, 1.4 1.5 y 1.6, respectivamente. El Código VHDL 1.7 muestra la **architecture** del circuito describiendo su estructura.

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

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;
-------------------------------------
```

Código VHDL 1.3: Puerta AND lógica de 2 entradas.

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

entity and3 is
   port ( y0 : out std_logic;
          x0, x1, x2 : in std_logic );
end entity and3;

architecture and3 of and3 is
begin
   y0 <= x0 and x1 and x2;
end architecture and3;
--------------------------------------
```

Código VHDL 1.4: Puerta AND lógica de 3 entradas.

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

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 1.5: Puerta OR lógica.

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

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;
-------------------------------------
```

Código VHDL 1.6: Puerta NOT lógica.

```
-------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
architecture circuito_Estruc of funcFG is
   signal xn, yn, zn: std_logic;
   signal sA1, sA2, sA3: std_logic;
-- Declaración de las clases de los componentes
   component and3 is
      port (   y0 : out std_logic ;
            x0, x1, x2 : in std_logic);
   end component and3;
   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;
begin
-- Instanciación y conexión de los componentes
   N1 : component not1 port map (xn, x);
   N2 : component not1 port map (yn, y);
   N3 : component not1 port map (zn, z);
   A1 : component and3 port map (sA1, xn, y, zn);
   A2 : component and2 port map (sA2, x, sA3);
   A3 : component and2 port map (sA3, yn, z);
   O1 : component or2 port map (F, sA1, sA2);
   G <= sA3;
end architecture circuito_Estruc;
-----------------------------------
```

Código VHDL 1.7: Solución al Apartado 1.d: **architecture** del circuito describiendo su estructura.

El código VHDL del banco de pruebas del circuito se muestra en Código VHDL 1.8. Los dos cronogramas obtenidos al simular el banco de pruebas se muestran en las Figuras 1.2–1.3.

```
--------------------------------------------------
-- 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));
        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;
--------------------------------------------------
```

Código VHDL 1.8: Solución al Apartado 1.e: banco de pruebas del circuito.

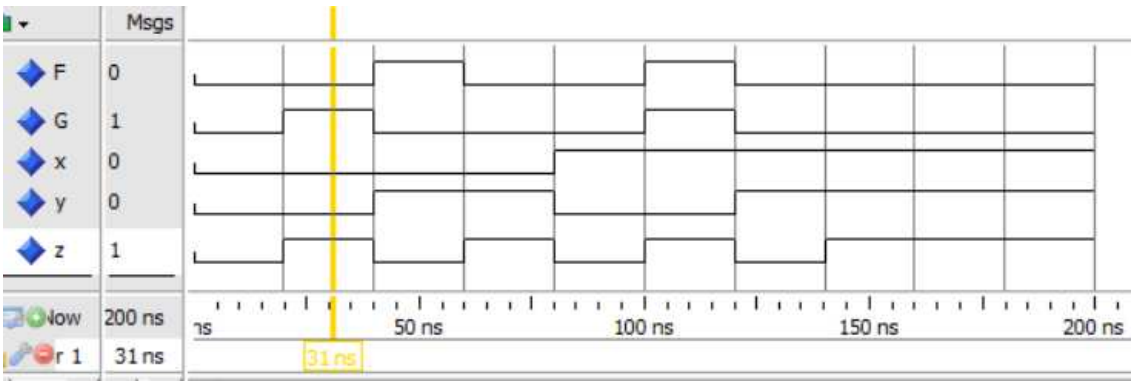

Figura 1.2: Cronograma del Apartado 1.e para el circuito de test del Apartado 1.b.

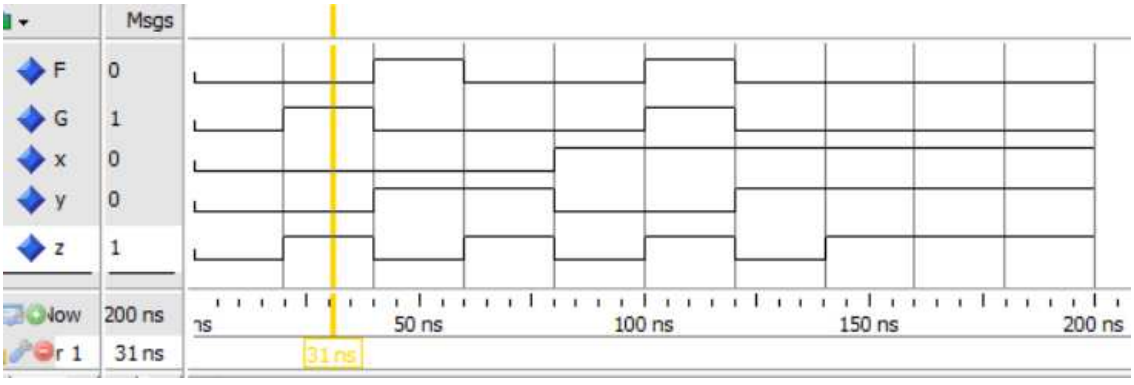

Figura 1.3: Cronograma del Apartado 1.e para el circuito de test del Apartado 1.d.

EJERCICIO 2

Se pretende diseñar un circuito incrementador del código de Gray. Este circuito combinacional tiene como entrada una palabra de código de Gray de N bits y su salida es la siguiente palabra del código de Gray de N bits. El código de Gray de 4 bits ($g_3g_2g_1g_0$), su correspondiente código binario ($b_3b_2b_1b_0$) y el código de Gray incrementado ($gInc_3gInc_2gInc_1gInc_0$) se muestran en la siguiente tabla.

Código binario `b(3:0)`	Código Gray `g(3:0)`	Código Gray incrementado `gInc(3:0)`
0000	0000	0001
0001	0001	0011
0010	0011	0010
0011	0010	0110
0100	0110	0111
0101	0111	0101
0110	0101	0100
0111	0100	1100
1000	1100	1101
1001	1101	1111
1010	1111	1110
1011	1110	1010
1100	1010	1011
1101	1011	1001
1110	1001	1000
1111	1000	0000

2.a) (1 punto) Escriba en VHDL la **entity** y la **architecture** que describa el comportamiento del incrementador de código Gray de 4 bits empleando para ello una asignación concurrente de selección (sentencia **with - select**).

2.b) (3 punto) Escriba en VHDL la **entity** y la **architecture** que describa el comportamiento del incrementador de código Gray de N bits. El número de bits N ha de ser una constante de tipo **generic**.

El comportamiento del circuito ha de ser descrito por el siguiente algoritmo:

Paso 1 Convertir la palabra de código de Gray ($g_{N-1} \dots g_0$) a la correspondiente palabra binaria ($b_{N-1} \dots b_0$). Para ello aplica la fórmula siguiente:

$b_i = g_i \oplus b_{i+1}$,

siendo $i = 0, \dots, N-1$ y $b_N = 0$.

Paso 2 Incrementa en una unidad la palabra binaria.

Paso 3 Convierte la palabra binaria resultante ($b_{N-1} \ldots b_0$) al código de Gray correspondiente ($g_{N-1} \ldots g_0$). Para ello aplica la fórmula siguiente:
$g_i = b_i \oplus b_{i+1}$,
siendo $i = 0, \ldots, N-1$ y $b_N = 0$.

2.c) (2 puntos) Programe en VHDL un banco de pruebas que testee todas las posibles entradas al circuito *incrementador* diseñado en el apartado 2.b cuando la constante de tipo **generic** N vale 4. El banco de pruebas debe comparar las salidas de la UUT con las salidas esperadas, mostrando el correspondiente mensaje de error en caso de que las salidas obtenidas de la UUT no correspondan con las esperadas.

Emplee este banco de pruebas para comprobar el diseño realizado al contestar el Apartado 2.b cuando N vale 4.

Incluya en la memoria el cronograma obtenido al realizar la simulación del banco de pruebas.

Solución al Ejercicio 2

El Código VHDL 1.9 muestra el código VHDL del circuito incrementador de código de Gray de 4 bits.

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

entity incGray4Bits is
   port ( gInc: out std_logic_vector (3 downto 0);
          g : in std_logic_vector(3 downto 0) );
end entity incGray4Bits;
-------------------------------------

architecture incGray4Bits of incGray4Bits is
begin
  with g select
    gInc <= "0001" when "0000",
            "0011" when "0001",
            "0010" when "0011",
            "0110" when "0010",
            "0111" when "0110",
            "0101" when "0111",
            "0100" when "0101",
            "1100" when "0100",
            "1101" when "1100",
            "1111" when "1101",
            "1110" when "1111",
            "1010" when "1110",
            "1011" when "1010",
            "1001" when "1011",
            "1000" when "1001",
            "0000" when others;
end incGray4Bits;
```

Código VHDL 1.9: Solución al Apartado 2.a: incrementador de código Gray de 4 bits descrito usando una asignación concurrente de selección (sentencia **with - select**).

El Código VHDL 1.10 muestra el código VHDL del circuito incrementador de código de Gray de N bits empleando el algoritmo descrito en el Apartado 2.b.

El banco de pruebas del incrementador de código de Gray de n bits se muestra en Código VHDL 1.11–1.12. El cronograma obtenido al simular el banco de pruebas se muestra en la Figura 1.4.

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

entity incGraynBits is
   generic(N: integer:=4);
   port ( gInc: out std_logic_vector (N-1 downto 0);
          g : in std_logic_vector(N-1 downto 0) );
end entity incGraynBits;
architecture incGraynBits of incGraynBits is
    signal b, bl: std_logic_vector(N-1 downto 0) ;
begin
-- Gray a binario
    b <= g xor ('0' & b(N-1 downto 1));
-- Binario incrementado
    bl <= std_logic_vector((unsigned(b)) + 1);
-- binario a Gray
    gInc<= bl xor ('0' & bl(N-1 downto 1));
end incGraynBits ;
```

Código VHDL 1.10: Solución al Apartado 2.b: incrementador de código Gray de N bits.

```
  ----------------------------------------------------------
-- Banco de pruebas del incrementador de Gray para N=4
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity bp_incGray is
    constant DELAY      : time      := 10 ns; -- Retardo usado en el test
    constant WIDTH :integer := 4;
end entity bp_incGray;

architecture bp_incGray of bp_incGray is
    signal gInc : std_logic_vector(WIDTH-1 downto 0);
    signal g: std_logic_vector (WIDTH-1 downto 0);

    component incGraynBits is
    generic(N: integer:=WIDTH);
    port ( gInc: out std_logic_vector (N-1 downto 0);
           g : in std_logic_vector(N-1 downto 0) );
    end component incGraynBits;
   -- Procedure que calcula la salida (expected_gInc) y lo compara con el
   -- valor de salida que se pasa como argumento (actual_gInc)
   -- Si ambos valores no coinciden, se muestra un mensaje y se
   -- incrementa el contador de errores (error_count)
   procedure check_salida
       ( g      : in std_logic_vector(WIDTH-1 downto 0);
         gInc         : in std_logic_vector(WIDTH-1 downto 0);
         error_count: inout integer ) is
         variable gInc_esp: std_logic_vector(WIDTH-1 downto 0);
    begin

         if g = "0000" then gInc_esp := "0001";
         elsif g = "0001" then gInc_esp := "0011";
         elsif g =  "0011" then gInc_esp := "0010";
         elsif g = "0010" then gInc_esp := "0110";
         elsif g = "0110" then gInc_esp := "0111";
         elsif g = "0111" then gInc_esp := "0101";
         elsif g = "0101" then gInc_esp := "0100";
         elsif g = "0100" then gInc_esp := "1100";
         elsif g = "1100" then gInc_esp :=  "1101";
         elsif g =  "1101" then gInc_esp := "1111";
         elsif g = "1111" then gInc_esp :=  "1110";
         elsif g = "1110" then gInc_esp := "1010";
         elsif g = "1010" then gInc_esp := "1011";
         elsif g = "1011" then gInc_esp := "1001";
         elsif g = "1001" then gInc_esp := "1000";
         else gInc_esp :=  "0000";
         end if;
```

Código VHDL 1.11: Solución al Apartado 2.b: banco de pruebas del incrementador de código de Gray de 4 bits.

```
          assert( gInc = gInc_esp)
          report "ERROR. Entrada: " & std_logic'image(g(3))
                  & std_logic'image(g(2)) & std_logic'image(g(1)) &
                  std_logic'image(g(0)) &
                  ", resultado esperado: " &
                  std_logic'image(gInc_esp(3)) & std_logic'image(gInc_esp(2)) &

                  std_logic'image(gInc_esp(1)) & std_logic'image(gInc_esp(0)) &

                  ", resultado actual: "
                  &  std_logic'image(gInc(3))  &  std_logic'image(gInc(2))  &

                   std_logic'image(gInc(1)) & std_logic'image(gInc(0)) &
                  " en el instante "
                  & time'image(now);

     if (gInc /= gInc_esp) then
       error_count := error_count + 1;
     end if;

  end procedure check_salida;
  -- Fin de la definición del procedure


begin
    UUT : component incGraynBits
          generic map (WIDTH)
          port map (gInc, g);

vec_test : process is
      variable error_count : integer := 0;
    begin
      report "Comienza la simulación";
      -- Generar todos los posibles valores de entrada
      for i in 0 to 2**WIDTH-1 loop
        g <= std_logic_vector(to_unsigned(i,WIDTH));
        wait for DELAY;
        check_salida(g, gInc, error_count);
      end loop;

     report "Simulación finalizada con "
     & integer'image(error_count) &
     " errores";
      wait;      -- Final de la simulación
  end process vec_test;
end architecture bp_incGray;
-------------------------------------------------
```

Código VHDL 1.12: Continuación del banco de pruebas del incrementador de código de Gray de 4 bits.

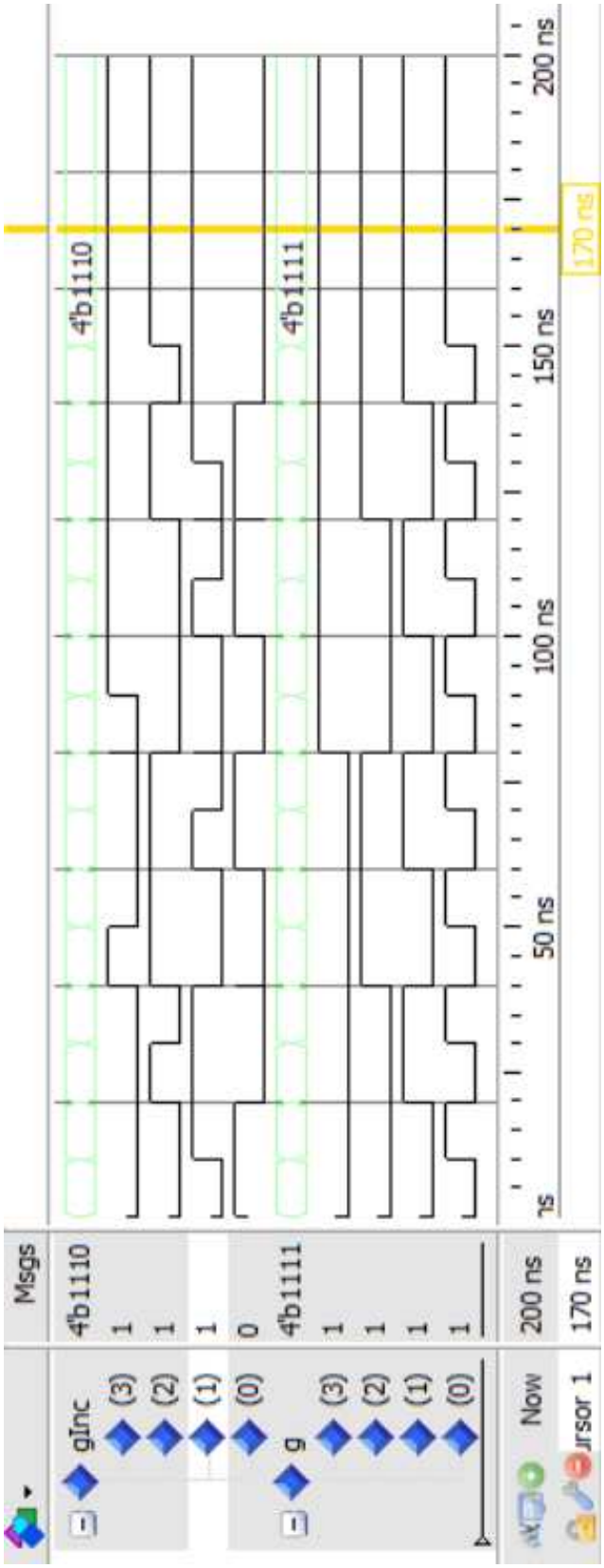

Figura 1.4: Cronograma del Apartado 2.C.