20-12-2014

Práctica 1

Tecnología de Computadores

Susana Pineda De Luelmo
DISEÑO Y DESARROLLO DE VIDEOJUEGOS-INGENIERÍA DE COMPUTADORES.

Se desea diseñar un circuito combinacional con una entrada de 4 bits y dos salidas cada una de un bit que valdrá la primera 1 si el número es primo y la segunda valdrá 1 si el número forma parte de la sucesión de Fibonacci.

El circuito se diseñará de diversas maneras, y para cada una de ellas se hará una descripción en VHDL. Cada una de las descripciones en VHDL será una arquitectura asociada a la siguiente entidad:

```
Entity practica_1 is
Port ( x: in std_logic_vector (3 downto 0),
       Z: out std_logic_vector (1 downto 0));
End practica_1;
```

Se pide hacer lo siguiente:

a) **Obtener la expresión más simplificada de Z en forma de suma de productos. Describir en VHD la expresión obtenida utilizando una asignación concurrente. El nombre de esta arquitectura debe ser concurrente_sdp.**

Para empezar hacemos la tabla de verdad de nuestro circuito, tomando como entradas X0, X1, X2 y X3 (ya que nos dice que nuestro circuito cuenta con una entrada de cuatro bits) y como salidas Z0 y Z1 (siendo Z0 la encargada de mostrar si un número es primo o no y Z1 la encargada de decir si el número es de la sucesión de Fibonnaci.

X3	X2	X1	X0	Z0	Z1
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	0	0

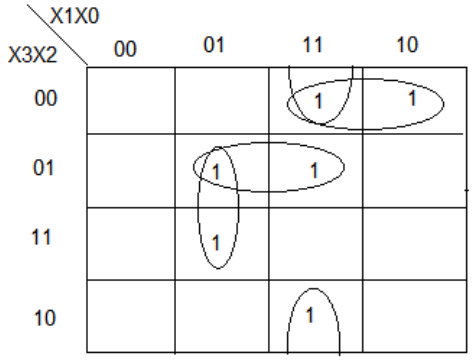
Aunque el 0 no pertenezca estrictamente a la sucesión de Fibonacci en algunos libros y páginas lo toman como el primer número de esta sucesión. 1.

Al tener que simplificar las salidas como suma de productos tenemos que hacer el sumatorio de los minterms.

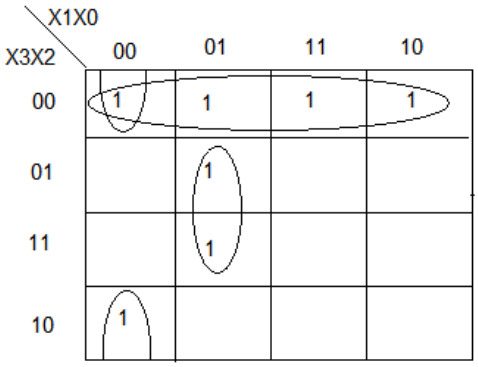
$Z0 = \sum m (2,3,5,7,11,13)$

$Z1 = \sum m (0,1,2,3,5,8,13)$

Para simplificar estas funciones utilizamos los mapas de Karnaugh y obtenemos las funciones de las salidas:



$Z = \overline{X3}\,\overline{X2}\,X1 + \overline{X3}\,X2\,X0 + X2\,\overline{X1}\,X0 + \overline{X2}\,X1\,X0$

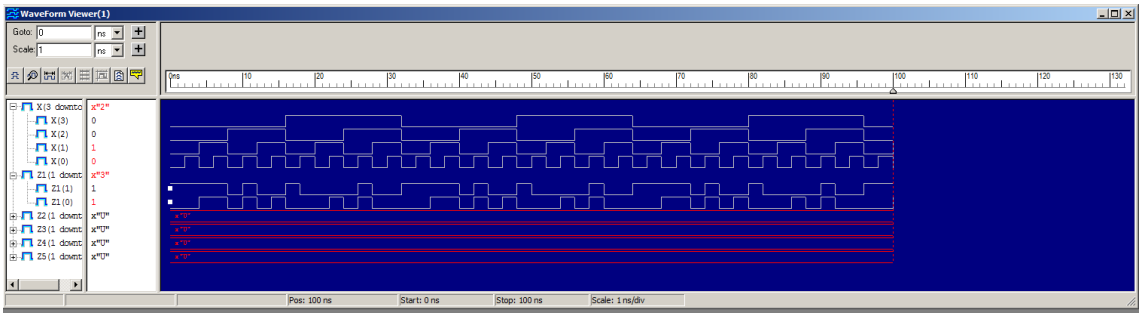


$Z1 = \overline{X3}\,\overline{X2} + X2\,\overline{X1}\,X0 + \overline{X2}\,\overline{X1}\,\overline{X0}$

Tras escribirlo en VHDL (archivo adjunto a la práctica) nos aparece una simulación en la que podemos ver las diferentes salidas para las diferentes entradas.

Descripción VHDL:

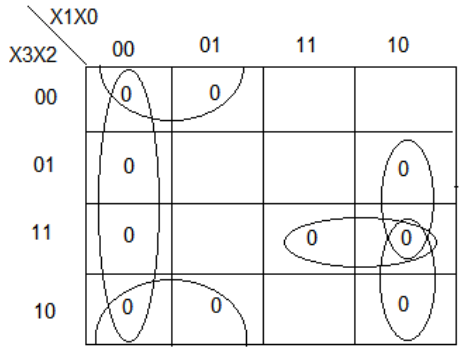
```
library ieee;
use ieee.std_logic_1164.all;

entity practica_1 is
port (x: in std_logic_vector(3 downto 0);
 z: out std_logic_vector(1 downto 0));
end practica_1;

architecture concurrente_sdp of practica_1 is
begin

      z(0) <= ((x(2) and (not x(1)) and x(0)) or ((not x(3)) and x(2)
and x(0)) or ((not x(3)) and (not x(2)) and x(1)) or ((not x(2)) and
x(1) and x(0)));
      z(1) <= (((not x(2)) and (not x(1)) and (not x(0))) or (( not
x(3)) and ( not x(2)) ) or (x(2) and (not x(1)) and x(0)));
end concurrente_sdp;
```



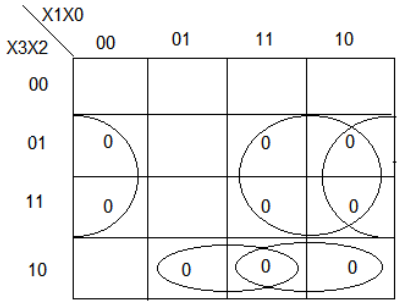
b) **Obtener la expresión más simplificada de Z en forma de producto de sumas. Describir en VHDL la expresión obtenida utilizando una asignación concurrente. El nombre de esta arquitectura debe ser concurrente_pds.**
Para obtener la expresión más simplificada de Z en forma de producto de sumas tenemos que multiplicar los maxterms de la función.

Z0 = ΠM (0,1,4,6,8,9,10,12,14,15)
Z1 = ΠM (4,6,7,9,10,11,12,14,15)

Para simplificar estas funciones utilizamos los mapas de karnaugh indicando los maxterms:



$Z0 = (X1+X0)(X2+X1)(\overline{X2}+\overline{X1}+X0)(\overline{X3}+\overline{X2}+\overline{X1})(\overline{X3}+\overline{X1}+X0)$

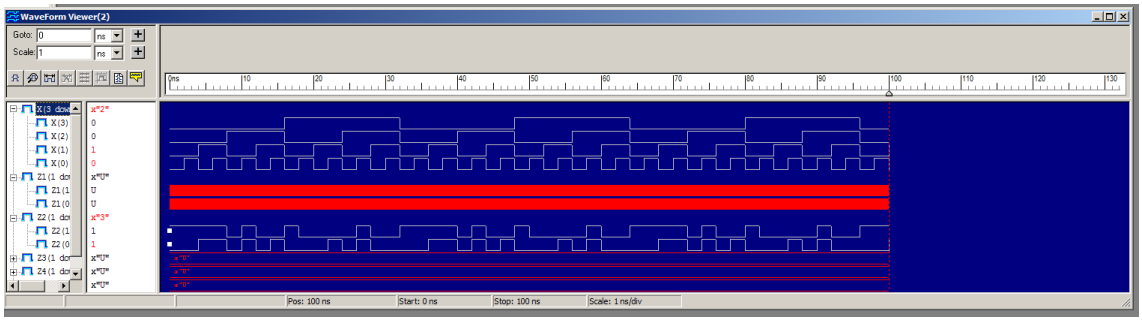


$Z1 = (\overline{X2}+X0)(\overline{X2}+\overline{X1})(\overline{X3}+X2+\overline{X0})(\overline{X3}+X2+\overline{X1})$

Tras describir el circuito nos aparece una simulación que coincide con la del apartado anterior ya que es el mismo circuito implementado de forma diferente:

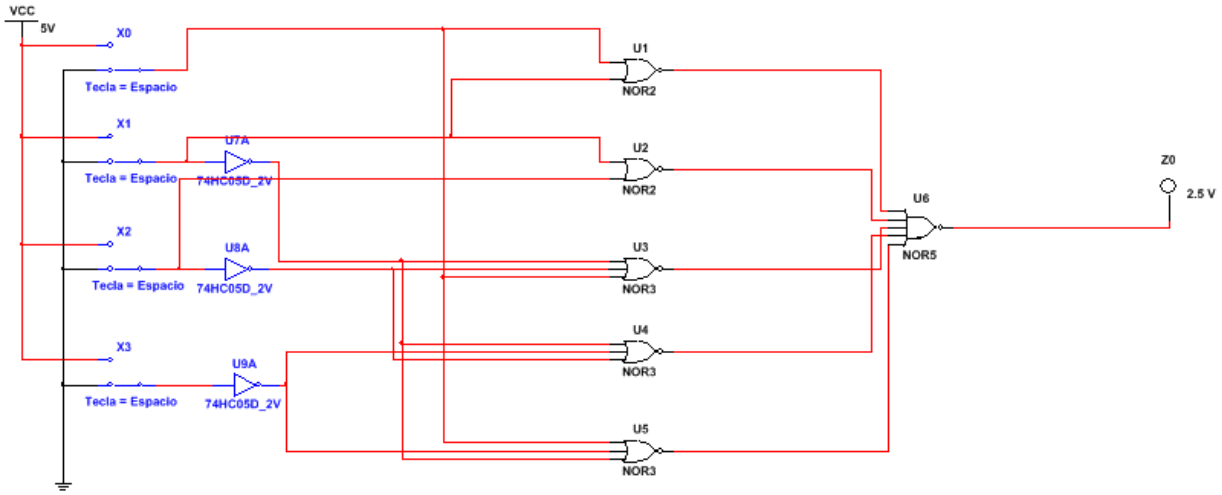
Descripción VHDL:

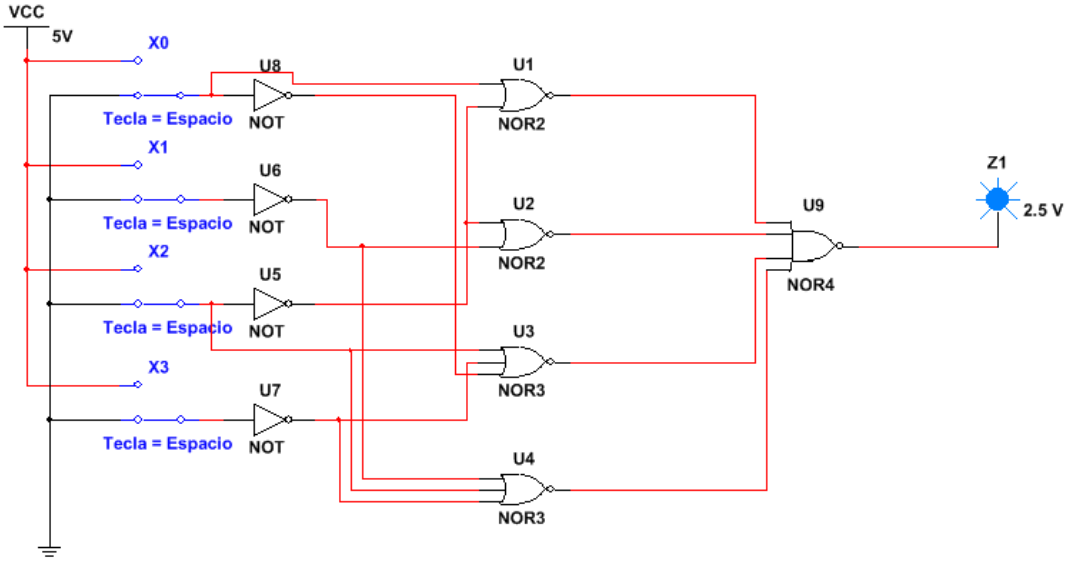
```
library ieee;
use ieee.std_logic_1164.all;

entity practica_1 is
port (x: in std_logic_vector(3 downto 0);
 z: out std_logic_vector(1 downto 0));
end practica_1;

architecture concurrente_pds of practica_1 is
begin
      z(0) <= ((x(1) or x(0)) and (x(2) or x(1)) and (( not x(2)) or
(not x(1)) or x(0)) and ((not x(3)) or (not x(2)) or (not x(1))) and
((not x(3)) or (not x(1)) or x(0)));
      z(1) <= (((not x(2)) or x(0)) and ((not x(2)) or (not x(1))) and
((not x(3)) or x(2) or (not x(0))) and ((not x(3)) or x(2) or (not
x(1))));

end concurrente_pds;
```



c) Implementar Z utilizando sólo puertas NOR (y opcionalmente inversores). Describir en VHDL el circuito obtenido utilizando las puertas lógicas proporcionadas en el archivo puertas_basicas.vhd. El nombre de esta arquitectura debe ser estructura_nor.

Para Implementar Z utilizando solo puertas NOR lo que haremos es utilizar la función simplificada de Z como producto de sumas y aplicarle un doble negador, pasando así las puertas OR a NOR y la puerta AND al negar todas sus entradas se transforma en una puerta NOR. Los circuitos implementados solo con puertas NOR quedarían:





(Las simulaciones en Multisim también van incluidas dentro del archivo .rar que contiene la práctica)

Tras describir el circuito en VHDL la simulación es la siguiente:

d) Implementar Z utilizando un decodificador de 3 a 8 y puertas lógicas auxiliares. Describir en VHDL el circuito obtenido utilizando en decodificador proporcionado en el archivo componentes.vhd y las asignaciones concurrentes o puertas lógicas que se considere oportuno. El nombre de esta arquitectura debe de ser estructural_deco.

Para implementar este circuito con un decodificador necesitaríamos uno de 4 a 16, ya que tenemos 4 entradas. Para poder implementar el circuito con un decodificador de 3 a 8 lo que haremos es utilizar 3 de esas entradas como entradas del decodificador y la otra quedará en función de las otras 3.

Para ello, lo primero que haremos, será poner Z en función de X3, X2 y X1.

Tomamos X3, X2 y X1 , vamos contando en binario y tomando los valores que toma X0 para esos números, quedando así las salidas de nuestro decodificador.

$$Z0 = \overline{X3}\ \overline{X2}\ X1\ \overline{X0} + \overline{X3}\ \overline{X2}\ X1\ X0 + \overline{X3}\ X2\ \overline{X1}\ X0 + \overline{X3}\ X2\ X1\ X0 + X3\ \overline{X2}\ X1\ X0 + X3\ X2\ \overline{X1}\ X0$$

↓

$$Z0 = \overline{X3}\,\overline{X2}\,\overline{X1}\,(0) + \overline{X3}\,\overline{X2}\,X1\,(1) + \overline{X3}\,X2\,\overline{X1}\,(X0) + \overline{X3}\,X2\,X1\,(X0) + X3\,\overline{X2}\,\overline{X1}\,(0) + X3\,\overline{X2}\,X1\,(X0) + X3\,X2\,\overline{X1}\,(X0) + X3\,X2\,X1\,(0)$$

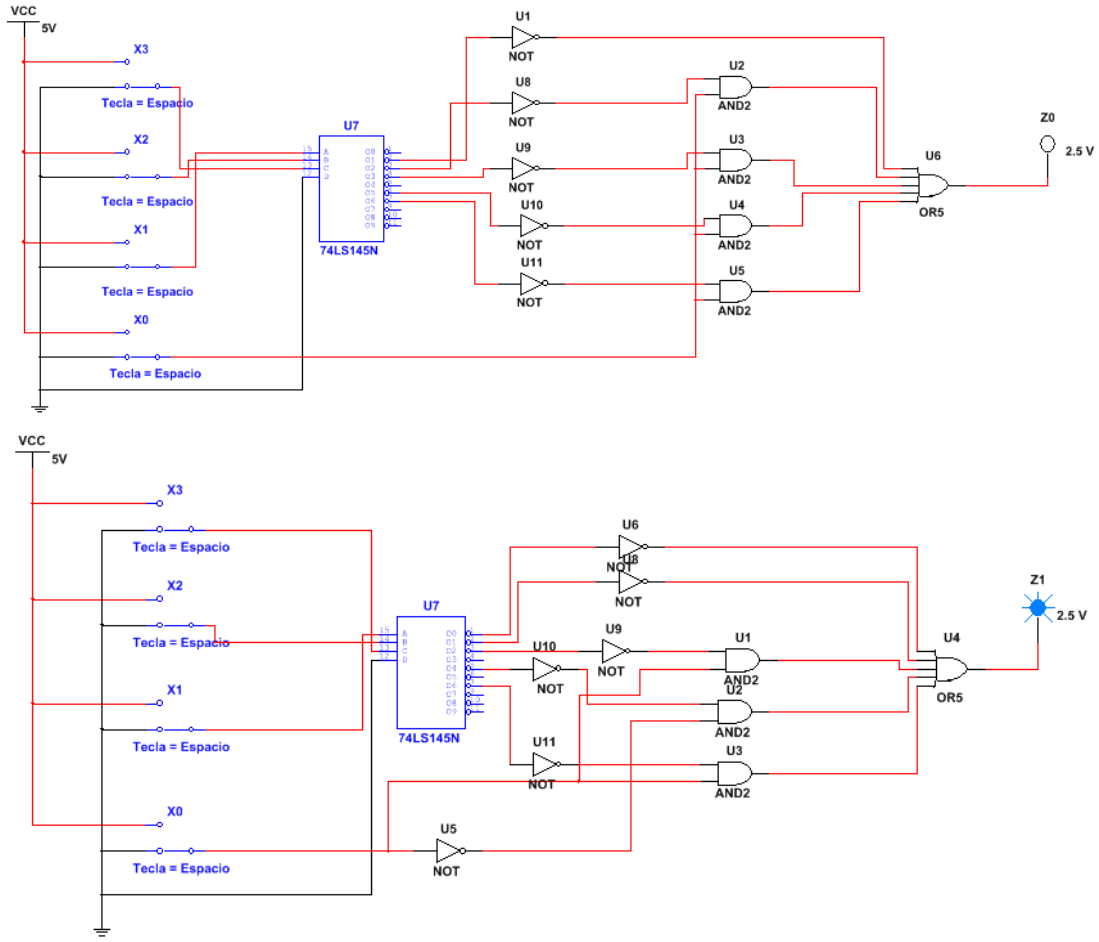
Hacemos lo mismo con Z1:

$$Z1 = \overline{X3}\,\overline{X2}\,\overline{X1}\,\overline{X0} + \overline{X3}\,\overline{X2}\,\overline{X1}\,X0 + \overline{X3}\,\overline{X2}\,X1\,\overline{X0} + \overline{X3}\,\overline{X2}\,X1\,X0 + \overline{X3}\,X2\,\overline{X1}\,X0 + X3\,\overline{X2}\,\overline{X1}\,\overline{X0} + X3\,\overline{X2}\,\overline{X1}\,X0 + X3\,X2\,\overline{X1}\,X0$$

$$\downarrow$$

$$Z1 = \overline{X3}\,\overline{X2}\,\overline{X1}\,(1) + \overline{X3}\,\overline{X2}\,X1\,(1) + \overline{X3}\,X2\,\overline{X1}\,(X0) + \overline{X3}\,X2\,X1\,(0) + X3\,\overline{X2}\,\overline{X1}\,(\overline{X0}) + X3\,\overline{X2}\,X1\,(0) + X3\,X2\,\overline{X1}\,(X0) + X3\,X2\,X1\,(0)$$

Implementamos el circuito:



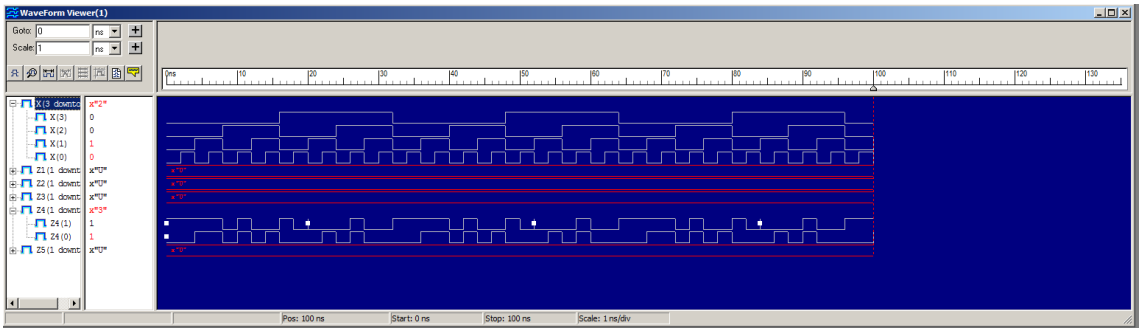
Nota: Como el decodificador de 4 a 8 de Multisim venía con las salidas negadas, en la simulación y en la implementación hay negadores que en la práctica de VHDL no aparecen. (Es solo por el decodificador de Multisim)

La simulación en VHDL sigue manteniendo los mismos valores:

Descripción VHDL:

```
library ieee;
use ieee.std_logic_1164.all;

entity practica_1 is
port (x: in std_logic_vector(3 downto 0);
 z: out std_logic_vector(1 downto 0));
end practica_1;
```

```
architecture estructural_deco of practica_1 is
signal a: std_logic_vector(2 downto 0);
signal b: std_logic_vector(7 downto 0);

begin

a(0) <= x (1);
a(1) <= x (2);
a(2) <= x (3);

deco: entity work.deco3a8(funcional) port map ('1',a,b);
z(0) <= (b(1) or (b(2) and x(0)) or (b(3) and x(0)) or (b(5) and x(0))
or (b(6) and x(0));
z(1) <= (b(0) or b(1) or (b(2) and x(0)) or (b(4) and (not x(0))) or
(b(6) and x(0));
end estructural_deco;
```

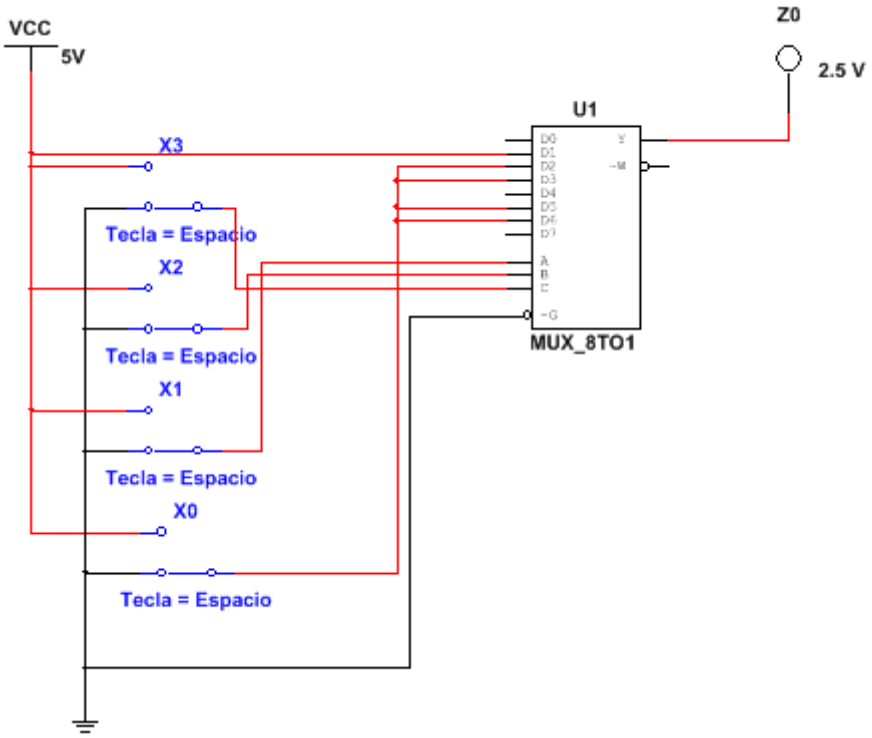


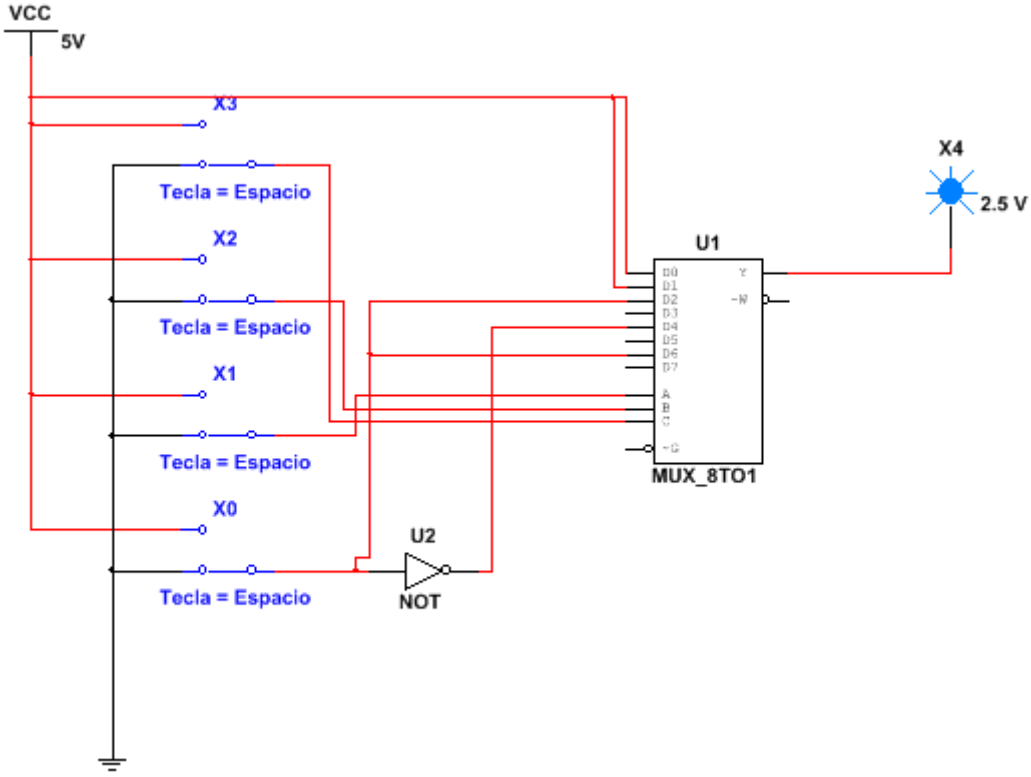
e) Implementar Z utilizando un multiplexor de 8 a 1 y puertas lógicas auxiliares. Describir en VHDL el circuito obtenido utilizando el multiplexor proporcionado en el archivo componentes.vhd y las asignaciones concurrentes o puertas lógicas que se considere oportuno. El nombre de esta arquitectura debe ser estructural_mux.

Al igual que en el caso anterior, necesitaríamos un multiplexor mayor para las entradas que tenemos, así que para simplificarlo utilizaremos X3, X2 y X1 como entradas de selección y X0 dependerá de las otras tres. Para esto, utilizamos las funciones anteriores en las que X0 estaba en función de X3, X2 y X1 e implementamos:

$$Z0 = \overline{X3}\,\overline{X2}\,\overline{X1}\,(0) + \overline{X3}\,\overline{X2}\,X1\,(1) + \overline{X3}\,X2\,\overline{X1}\,(X0) + \overline{X3}\,X2\,X1\,(X0) + X3\,\overline{X2}\,\overline{X1}\,(0) + X3\,\overline{X2}\,X1\,(X0) + X3\,X2\,\overline{X1}\,(X0) + X3\,X2\,X1\,(0)$$

$$Z1 = \overline{X3}\,\overline{X2}\,\overline{X1}\,(1) + \overline{X3}\,\overline{X2}\,X1\,(1) + \overline{X3}\,X2\,\overline{X1}\,(X0) + \overline{X3}\,X2\,X1\,(0) + X3\,\overline{X2}\,\overline{X1}\,(\overline{X0}) + X3\,\overline{X2}\,X1\,(0) + X3\,X2\,\overline{X1}\,(X0) + X3\,X2\,X1\,(0)$$

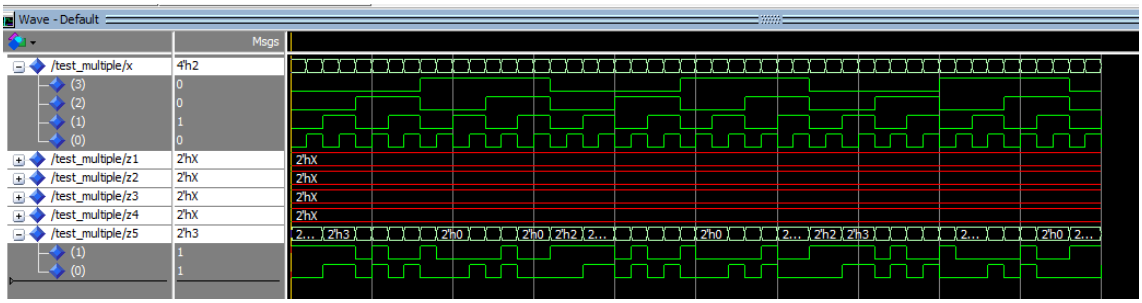

VCC
5V
Z0
2.5 V
U1
X3
Tecla = Espacio
X2
Tecla = Espacio
X1
Tecla = Espacio
X0
Tecla = Espacio
MUX_8TO1


VCC
5V
X3
Tecla = Espacio
X2
Tecla = Espacio
X1
Tecla = Espacio
X0
Tecla = Espacio
U2
NOT
X4
2.5 V
U1
MUX_8TO1

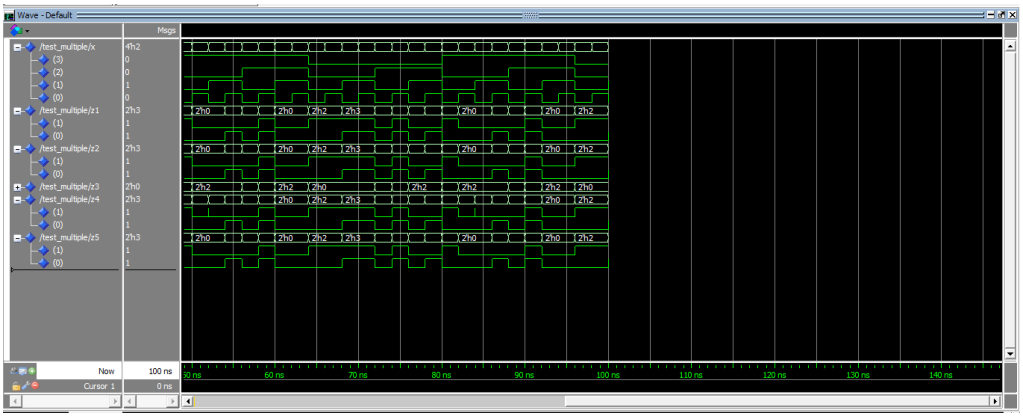
Tras la descripción en VHDL el cronograma sigue siendo el mismo:

Descripción VHDL:

```
architecture estructural_mux of practica_1 is
signal b: std_logic_vector (7 downto 0);
signal v: std_logic_vector (7 downto 0);
signal sel: std_logic_vector (2 downto 0);

begin

sel (0) <= x(1);
sel (1) <= x(2);
sel (2) <= x(3);

mux0: entity work.mux8a1(funcional) port map ('1', b, sel, z(0));
b(0) <= '0';
b(1) <= '1';
b(2) <= x(0);
b(3) <= x(0);
b(4) <= '0';
b(5) <= x(0);
b(6) <= x(0);
b(7) <= '0';

mux1: entity work.mux8a1(funcional) port map ('1', v, sel, z(1));

v(0) <= '1';
v(1) <= '1';
v(2) <= x(0);
v(3) <= '0';
v(4) <= (not x(0));
v(5) <= '0';
v(6) <= x(0);
v(7) <= '0';

end estructural_mux;
```



f) Utilizando el test-bench que se proporciona en el archivo test_bench_practica1.vhd simular simultáneamente los distintos circuitos de los apartados a) a e). si no se ha realizado alguno de los apartados anteriores se debe comentar la línea correspondiente en el archivo test_bench_practica1.vhd para que no de error. Es importante que en los apartados anteriores se respeten los nombres propuestas para las arquitecturas con el fin de que el test_bench no de errores.



Bibliografía

1. http://curiosidades.batanga.com/4461/que-es-la-sucesion-de-fibonacci