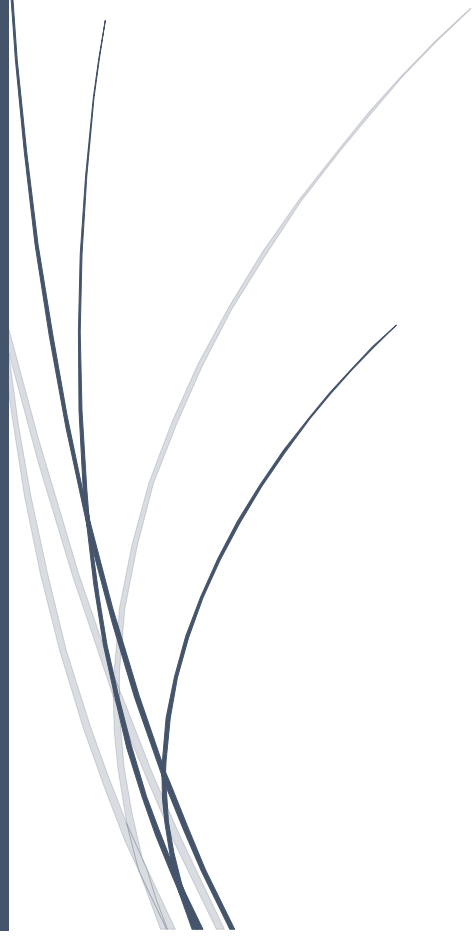
20-12-2014

Práctica 2

Tecnología de Computadores



Susana Pineda De Luelmo
DISEÑO Y DESARROLLO DE VIDEJOUEGOS – INGENIERÍA DE COMPUTADORES

Se desea diseñar un circuito secuencial que tiene una entrada de datos X, de 1 bit, además de una entrada de reset y una entrada de reloj. El sistema tiene una salida Z de 1 bit, que vale '1' cuando los últimos 5 bits recibidos en X son 10011.

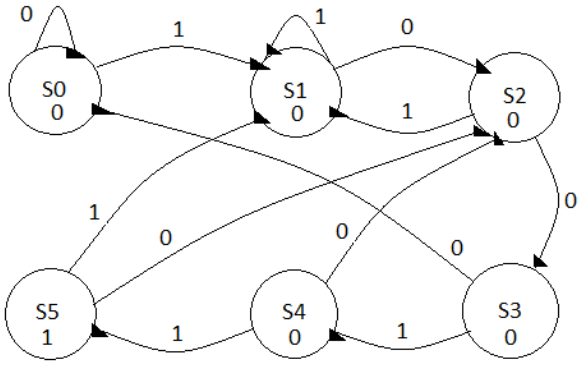
El circuito se diseñará de diversas maneras, y para cada una de ellas se hará una descripción en VHDL. Cada una de las descripciones en VHDL será una arquitectura asociada a la siguiente entidad.

```
Entity practica_2 is
      Port (clk, rst,x: in std_logic;
            Z: out std_logic);
End practica_2;
```

Para las distintas descripciones que se deben hacer se proporcionan modelos de los flip-flops en el archivo "flip-flops.vhd" y un modelo de un decodificador de 4 a 16 en el archivo "deco4a16.vhd".
Se pide hacer lo siguiente:

a) **Obtener el diagrama de estados del sistema, y la tabla de transición de estados del sistema, implementando como máquina de estados de Moore.**

Para obtener el diagrama de estados del sistema, vamos analizando las diferentes posibilidades que se pueden presentar. Estando en un estado inicial en el que el aun no tenemos ningún valor puede llegarnos tanto un 0 como un 1, en caso de que nos llegue un uno ya tendríamos el primer número de nuestra serie, en caso contrario seguiríamos en ese estado inicial a la espera de que nos llegue un 1. Estando en el estado 1 tenemos la posibilidad, otra vez, de que nos llegue o un 1 o un 0, en caso de que nos llegue un 1 nos seguiríamos manteniendo en dicho estado con nuestro primer digito a la espera del segundo, si por el contrario nos llega un 0, pasaríamos a un estado siguiente, S2, en el que ya tendríamos los dos primeros dígitos de nuestra secuencia (10), en este punto, tenemos las mismas posibilidades que antes, que nos llegue un 1 o un 0, en caso de que nos llegue un 1, tendríamos la secuencia 101, y volveríamos al estado 2 a la espera de que nos llegue otro 0. Continuamos haciendo esto hasta que tengamos todas las posibilidades de la máquina de estados y completemos la secuencia.
El resultado final será un diagrama de estados en el que nos aparezcan todas las conexiones posibles entre los diferentes estados.
Al estar implementando el circuito en una máquina de Moore, las salidas solo dependen del estado actual en el que estamos y no de las entradas que recibe el circuito.



Una vez diseñado el diagrama de estados, la información que nos proporciona se pasa a una tabla de cambios de estados, en la que podemos ver los cambios de un estado a otro dependiendo de las entradas.
Los estados deben estar codificados en binario, por lo que necesitaremos 3 bits para representar el estado.

Estado	**Q2**	**Q1**	**Q0**
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	0	0
S5	1	0	1

Tras codificar los estados, pasamos a hacer la tabla completa, en la que aparecen las entradas (reset, clk, X) el estado, la salida Z, y el estado siguiente.

Reset	CLK	Q2	Q1	Q0	X	Z	Q2’	Q1’	Q0’
1	x	X	X	X	X	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	1	0	0	0	1	0
0	↑	0	0	1	1	0	0	0	1
0	↑	0	1	0	0	0	0	1	1
0	↑	0	1	0	1	0	0	0	1
0	↑	0	1	1	0	0	0	0	0
0	↑	0	1	1	1	0	1	0	0
0	↑	1	0	0	0	0	0	1	0
0	↑	1	0	0	1	0	1	0	1
0	↑	1	0	1	0	1	0	1	0
0	↑	1	0	1	1	1	0	0	1

NOTA: las combinaciones de estados 1 1 0 y 1 1 1, no se tienen en cuenta, ya que son estados a los que nuestro circuito no llegará. No se toman como combinaciones que puedan representar tanto el 1 como el 0.

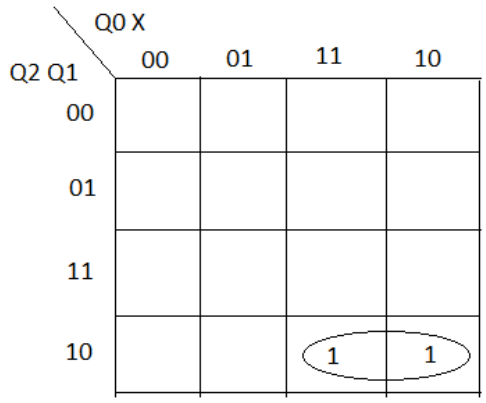
b) Implementar el sistema utilizando flip-flops tipo D además de puertas lógicas auxiliares. Describir el circuito obtenido en VHDL utilizando los flip-flops D proporcionados en el archivo “flip-flops.vhd”, y puertas lógicas o expresiones concurrentes auxiliares. El nombre de esta arquitectura debe ser estructural_D

Para implementar el circuito con biestables tipo D, necesitamos añadir a la tabla de cambios de estado las diferentes entradas que necesitaran los biestables para pasar de un estado a otro. En el caso de los biestables tipo D, la salida es igual que la entrada, es decir, si queremos cambiar de un estado 0 a un estado 1 solo tenemos que poner a la entrada del biestable un 1 y si queremos pasar de un estado 1 a un estado 0 solo tenemos que poner a la entrada del biestable un 0.
Después de añadir a la tabla de verdad las combinaciones de los diferentes Biestables tenemos que simplificar mediante karnaugh la salida Z y los biestables D2 D1 y D0.

reset	clk	Q2	Q1	Q0	X	Z	Q2’	Q1’	Q0’	D2	D1	D0
1	X	X	X	X	X	0	0	0	0	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	1
0	↑	0	0	1	0	0	0	1	0	0	1	0
0	↑	0	0	1	1	0	0	0	1	0	0	1
0	↑	0	1	0	0	0	0	1	1	0	1	1
0	↑	0	1	0	1	0	0	0	1	0	0	1
0	↑	0	1	1	0	0	0	0	0	0	0	0
0	↑	0	1	1	1	0	1	0	0	1	0	0
0	↑	1	0	0	0	0	0	1	0	0	1	0
0	↑	1	0	0	1	0	1	0	1	1	0	1
0	↑	1	0	1	0	1	0	1	0	0	1	0
0	↑	1	0	1	1	1	0	0	1	0	0	1

Simplificamos Z, D2, D1, D0 en función de Q2, Q1, Q0 y X

Z

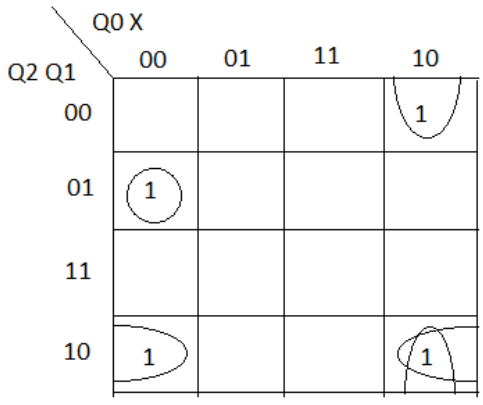


$Z = Q2\ \overline{Q1}\ Q0$

D2

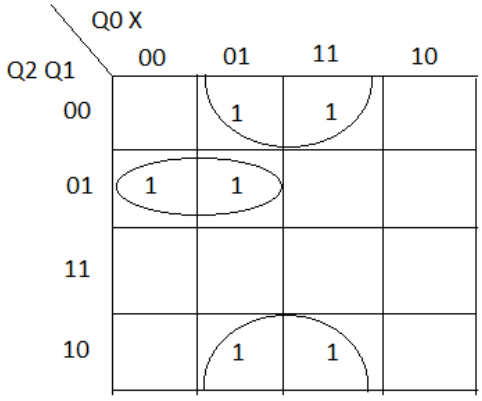
$$D2 = \overline{Q2}\ Q1\ Q0\ X + Q2\ \overline{Q1}\ \overline{Q0}\ X$$

D1

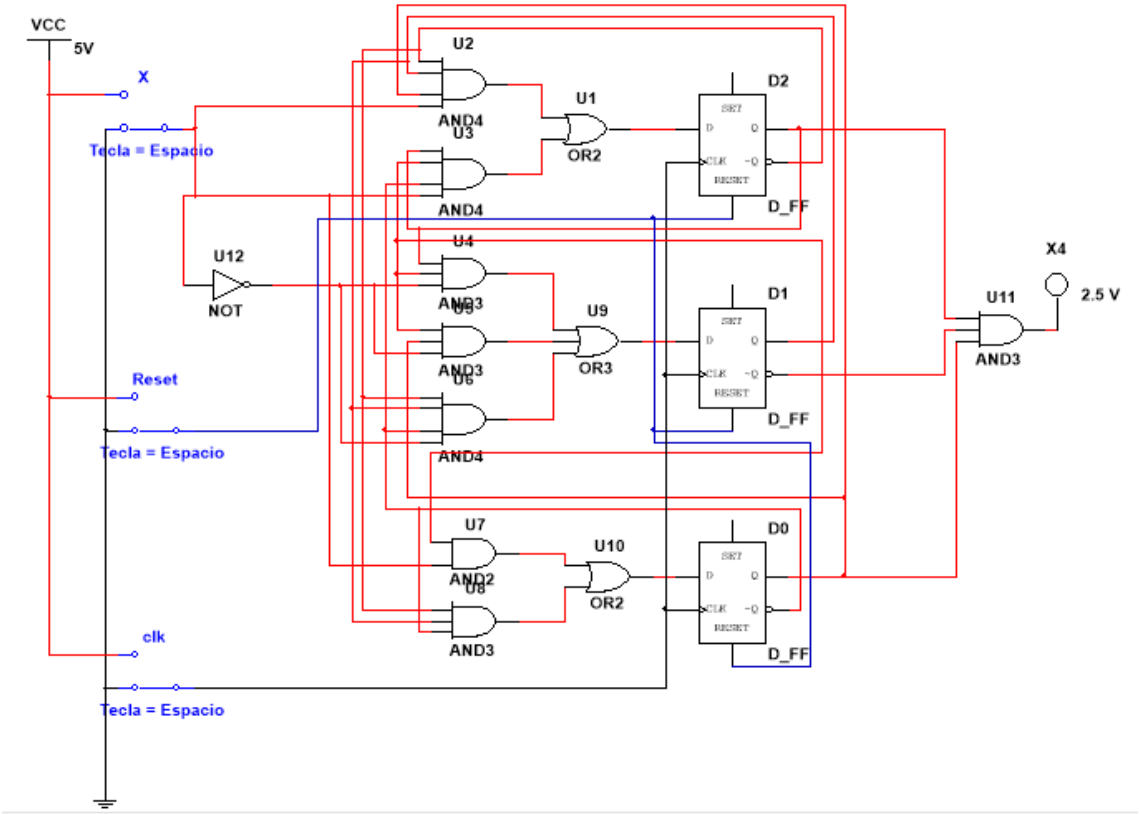


$D1 = Q2\ \overline{Q1}\ \overline{X} + \overline{Q1}\ Q0\ \overline{X} + \overline{Q2}\ Q1\ \overline{Q0}\ \overline{X}$

D0



$$D0 = \overline{Q1}\, x + \overline{Q2}\, Q1\, \overline{Q0}$$

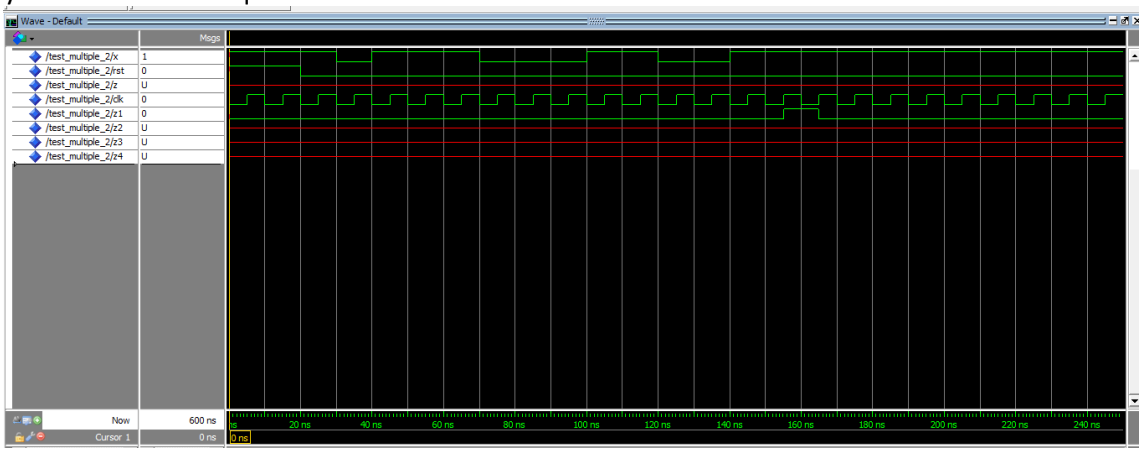


La descripción en VHDL es:

```
use ieee.std_logic_1164.all;

entity practica_2 is
  port (clk, rst, x: in std_logic;
        z: out std_logic);
end practica_2;


architecture estructural_D of practica_2 is
signal inD2, inD1, inD0: std_logic;
signal Q2, Q1, Q0, notQ2, notQ1, notQ0, notX: std_logic;
```

```
begin
            inv_x: entity work.not1 port map(x, notX);
            inv_q2: entity work.not1 port map(Q2, notQ2);
            inv_q1: entity work.not1 port map(Q1, notQ1);
            inv_q0: entity work.not1 port map(Q0, notQ0);
            inD2 <= ((notQ2 AND Q1 AND Q0 AND X)  OR (Q2
AND notQ1 AND notQ0 AND X));
            D2: entity work.flipflopD port map(clk, rst,
inD2, Q2);
            inD1 <= ((Q2 AND notQ1 AND notX) OR ( notQ1 AND
Q0 AND notX) OR (notQ2 AND Q1 AND notQ0 AND notX) );
            D1: entity work.flipflopD port map(clk, rst,
inD1, Q1);
            inD0 <= ((notQ1 AND X) OR (notQ2 AND Q1 AND
notQ0));
            D0: entity work.flipflopD port map(clk, rst,
inD0, Q0);
            Z <= (Q2 AND notQ1 AND Q0);

end estructural_D;
```

y la simulación nos queda:



c) Implementar el sistema utilizando flip-flops tipo JK además de puertas lógicas auxiliares. Describir el circuito obtenido en VHDL utilizando los flip-flops JK proporcionados en el archivo "flip-flops.vhd", y puertas lógicas o expresiones concurrentes auxiliares. El nombre de esta arquitectura debe ser estructural_jk.

Para implementar el circuito con biestables tipo JK, tenemos que añadir a la tabla otras dos opciones por cada biestable, la entrada J y la entrada K.

En el caso de los biestables JK podemos utilizar 1 para cambiar de 0 a 1 y viceversa o en algunos casos se nos admite tanto utilizar 1 como utilizar 0, por lo que esas opciones quedan marcadas con X. A la hora de simplificar podremos utilizar estas X como si fuesen 1 o 0, según nos convenga.

Los pasos a realizar son los mismos que en el apartado anterior, hacemos la tabla de verdad con los nuevos biestables y después simplificamos en función de Q2, Q1, Q0 y X.

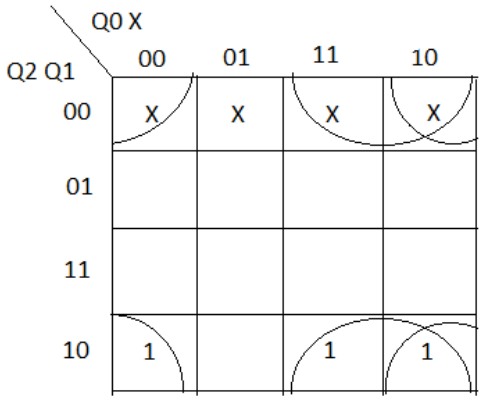
reset	clk	Q2	Q1	Q0	X	Z	Q2’	Q1’	Q0’	J2	K2	J1	K1	J0	K0
1	X	X	X	X	0	X	0	0	0	0	0	0	0	0	0
0	↑	0	0	0	0	0	0	0	0	0	X	0	x	0	X
0	↑	0	0	0	1	0	0	0	1	0	X	0	X	1	X
0	↑	0	0	1	0	0	0	1	0	0	X	1	X	1	X
0	↑	0	0	1	1	0	0	0	1	0	X	0	X	0	X
0	↑	0	1	0	0	0	0	1	1	0	X	0	X	1	X
0	↑	0	1	0	1	0	0	0	1	0	X	1	X	1	X
0	↑	0	1	1	0	0	0	0	0	0	X	1	X	1	X
0	↑	0	1	1	1	0	1	0	0	1	X	1	X	1	X
0	↑	1	0	0	0	0	0	1	0	X	1	X	1	X	0
0	↑	1	0	0	1	0	1	0	1	X	0	X	0	X	1
0	↑	1	0	1	0	1	0	1	0	X	1	X	1	X	1
0	↑	1	0	1	1	1	0	0	1	x	1	x	0	X	0

Simplificamos:

J2

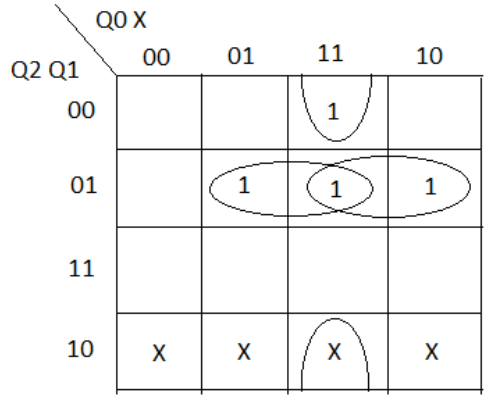
$$J2 = \overline{Q2}\ Q1\ Q0\ X$$

K2



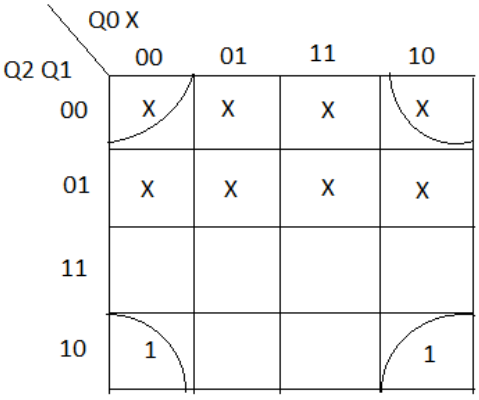
$$K2 = \overline{Q1}\ Q0 + \overline{Q1}\ \overline{X}$$

J1



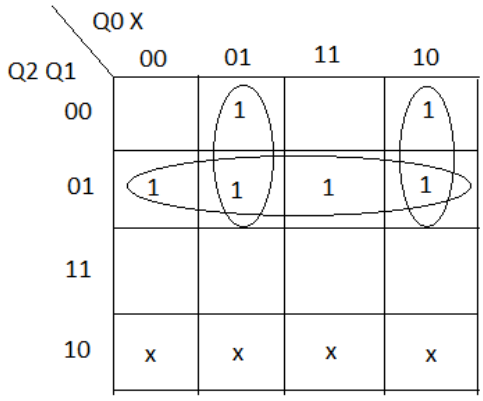
$$J1 = \overline{Q2}\ Q1\ X + \overline{Q2}\ Q1\ Q0 + \overline{Q1}\ Q0\ X$$

K1



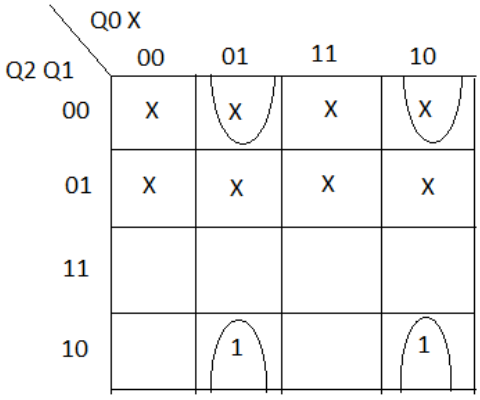
$K1 = \overline{Q1}\,\overline{X}$

J0

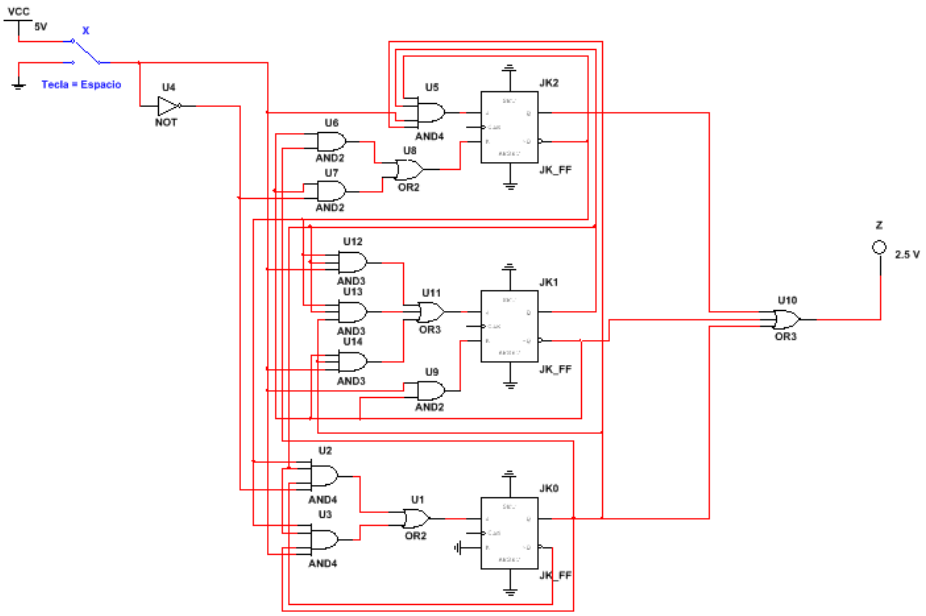


$J0 = \overline{Q2}\,Q1 + \overline{Q2}\,\overline{Q0}\,X + \overline{Q2}\,Q0\,\overline{X}$

K0



$K0 = \overline{Q1}\,\overline{Q0}\,X + \overline{Q1}\,Q0\,\overline{X}$



La descripción en VHDL es:

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

entity practica_2 is
  port (clk, rst, x: in std_logic;
        z: out std_logic);
end practica_2;

architecture estructural_JK of practica_2 is
signal inJ2, inJ1, inJ0: std_logic;
signal inK2, inK1, inK0: std_logic;
signal Q2, Q1, Q0, notQ2, notQ1, notQ0, notX: std_logic;
begin

           inv_x: entity work.not1 port map(x, notX);
           inv_q2: entity work.not1 port map(Q2, notQ2);
           inv_q1: entity work.not1 port map(Q1, notQ1);
           inv_q0: entity work.not1 port map(Q0, notQ0);
           inJ2 <= (Q1 AND Q0 AND X);
           inK2 <= ((Q0 AND X) OR notX);
           JK2: entity work.flipflopJK port map(clk, rst,
inJ2, ink2, Q2);
           inJ1 <= ((Q0 AND notX) OR (notQ2 AND Q1 AND X));
           inK1 <= (notX);
           JK1: entity work.flipflopJK port map(clk, rst,
inJ1, ink1, Q1);
           inJ0 <= (Q1 OR (notQ0 AND X) OR (Q0 AND notX));
           inK0 <= ((notQ0 AND X) OR (Q0 AND notX));
           JK0: entity work.flipflopJK port map(clk, rst,
inJ0, ink0, Q0);
           Z <= (Q2 AND notQ1 AND Q0);
```

```
end estructural_JK;
```

d) Implementar el sistema utilizando flip-flops tipo T, un decodificador de 4 a 16 y las puertas lógicas que sean necesarias. Describir el circuito obtenido en VHDL utilizando los flip-flops T proporcinados en el archivo "flip-flops.vhd", el decodificador proporcionado en el archivo "deco4a16" y puertas lógicas o expresiones concurrentes auxiliares. El nombre de esta arquitectura debe ser estructural_T.

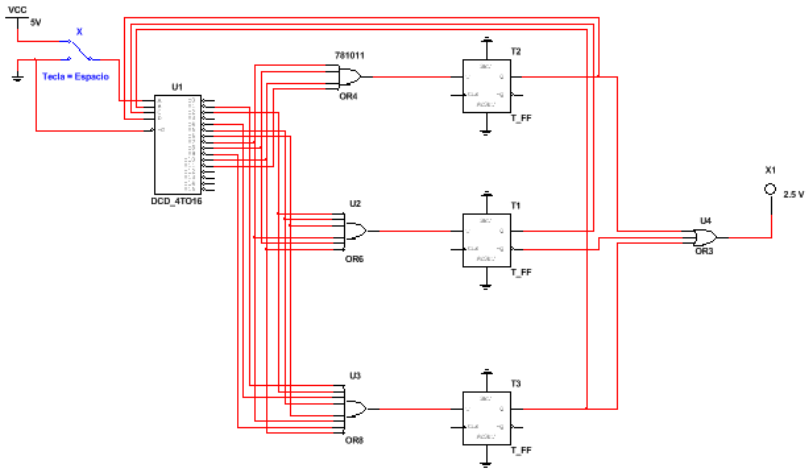
Para implementar el circuito utilizando biestables tipo T tenemos que añadir a la tabla de verdad dichos biestables para ver que tiene que entrar en el biestable para que nos de el siguiente estado.

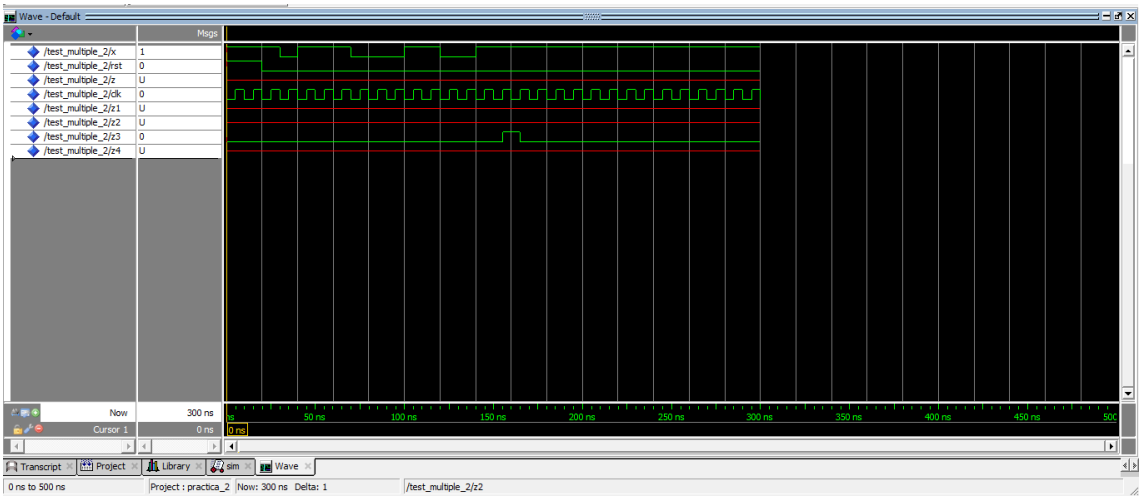
Al ser un biestable tipo T, para pasar de un estado 0 a un estado 1 o viceversa tenemos que poner 1 en el biestable, si queremos permanecer en el estado tenemos que poner 0 a la entrada del biestable. Asi, la tabla de verdad que nos quedaes la siguiente:

Como a la entrada de los biestables tenemos un decodificador 4 a 16 no nos hace falta simplificar T2 T1 y T0, simplemente tenemos que ver que salidas del decodificador son las encargadas de cambiar al estado siguiente.

reset	clk	Q2	Q1	Q0	X	Z	Q2'	Q1'	Q0'	T2	T1	T0
1	X	X	X	X	X	0	0	0	0	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	1
0	↑	0	0	1	0	0	0	1	0	0	1	1
0	↑	0	0	1	1	0	0	0	1	0	0	0
0	↑	0	1	0	0	0	0	1	1	0	0	1
0	↑	0	1	0	1	0	0	0	1	0	1	1
0	↑	0	1	1	0	0	0	0	0	0	1	1
0	↑	0	1	1	1	0	1	0	0	1	1	1
0	↑	1	0	0	0	0	0	1	0	1	1	0
0	↑	1	0	0	1	0	1	0	1	0	0	1
0	↑	1	0	1	0	1	0	1	0	1	1	1
0	↑	1	0	1	1	1	0	0	1	1	0	0

El biestable T2 cambiará de estado cuando le lleguen las salidas del decodificador 7,8,10 o 11. EL biestable T1 cambiara cuando le lleguen las salidas del decodificador 2,5,6,7,8 o 10 y el biestable T0 cambiará cuando le lleguen las entradas 1,2,4,5,6,7,9 0 10.





El código en VHDL es:

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

architecture estructural_T of practica_2 is
  signal inT2, inT1, inT0 : std_logic;
  signal Q2, Q1, Q0, notQ2, notQ1, notQ0, notX: std_logic;
      signal inDeco: std_logic_vector(3 downto 0);
      signal outDeco: std_logic_vector(15 downto 0);
      signal enabl: std_logic;
  begin
              inv_x: entity work.not1 port map(x, notX);
              inv_q2: entity work.not1 port map(Q2, notQ2);
              inv_q1: entity work.not1 port map(Q1, notQ1);
              inv_q0: entity work.not1 port map(Q0, notQ0);

              inDeco(3) <= Q2;
              inDeco(2) <= Q1;
              inDeco(1) <= Q0;
              inDeco(0) <= X;
              Deco_4a16: entity work.deco4a16 port map(enabl,
inDeco, outDeco);
```

```
                inT2 <= (outDeco(7) OR outDeco(8) OR outDeco(10) OR
outDeco(11));
                T2: entity work.flipflopT port map(clk, rst, inT2,
Q2);
                inT1 <= (outDeco(2) OR outDeco(5) OR outDeco(6) OR
outDeco (7) OR outDeco(8) OR outDeco(10));
                T1: entity work.flipflopT port map(clk, rst, inT1,
Q1);
                inT0 <= (outDeco(1) OR outDeco(2) OR outDeco(4) OR
outDeco (5) OR outDeco(6) OR outDeco(7) OR outDeco (9) OR
outDeco(10));
                T0: entity work.flipflopT port map(clk, rst, inT0,
Q0);


                Z <= (Q2 AND notQ1 AND Q0) ;

end estructural_T;
```

f) Utilizando el test-bench que se proporciona en el archivo test_multiple_practica2.vhd simular simultáneamente los distintos circuitos de los apartados b a e.