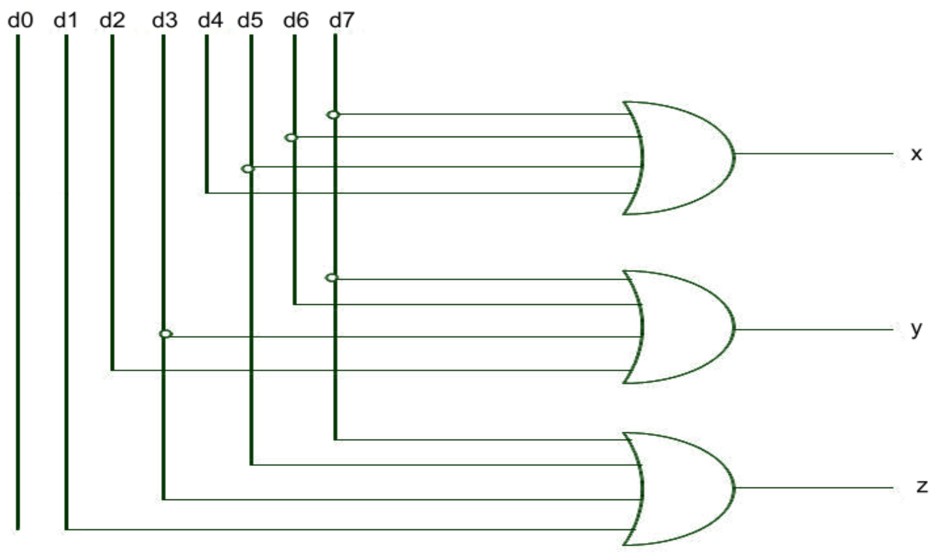
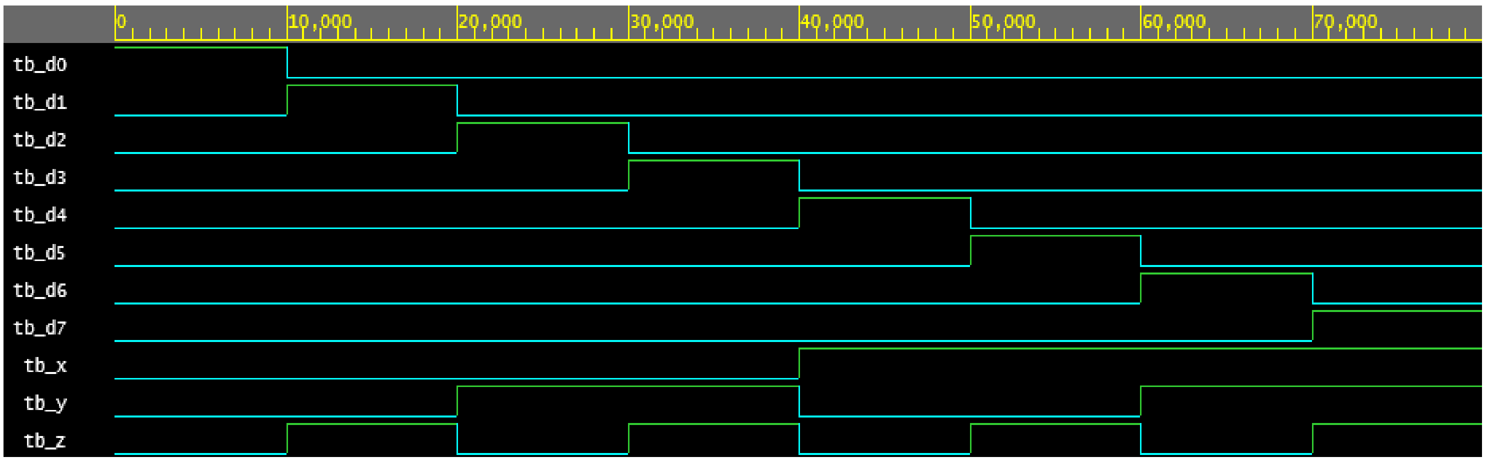
8 to 3 Encoder

d7	d6	d5	d4	d3	d2	d1	d0	x	y	z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



Based on the circuit we need 3 OR gates of four inputs where: x = d7+d6+d5+d4, y = d7+d6+d3+d2 and z = d7+d5+d3+d1. When one of the input gates is equal to 1 the outputs x,y,z (depending on the logic expressions) are forced to become equal to 1 as well. In all 3 circuits the simulation results are as follows:

Run Time: 80ns



> For instance:
10 ns: d0 = 0, **d1 = 1** , d2 = 0 , d3 = 0 , d4 = 0 , d5 = 0 , d6 = 0, d7 = 0 → **x = 0, y = 0, z = 1**
30 ns: d0 = 0, d1 = 0 , d2 = 0 , **d3 = 1** , d4 = 0 , d5 = 0 , d6 = 0, d7 = 0 → **x = 0, y = 1, z = 1**
40 ns: d0 = 0, d1 = 0 , d2 = 0 , d3 = 0 , **d4 = 1** , d5 = 0 , d6 = 0, d7 = 0 → **x = 1, y = 0, z = 0**
70 ns: d0 = 0, d1 = 0 , d2 = 0 , d3 = 0 , d4 = 0 , d5 = 0 , d6 = 0, **d7 = 1** → **x = 1, y = 1, z = 1**

Circuit Implementations:
*The testbench remains the same for all three models.

> DataFlow Model:
```
library IEEE;
use IEEE.std_logic_1164.all;

entity encoder8x3 is
        port(d0,d1,d2,d3,d4,d5,d6,d7: in bit;
     x,y,z: out bit);
end encoder8x3;

architecture dataflow of encoder8x3 is
        begin
        x <= d7 or d6 or d5 or d4;
        y <= d7 or d6 or d3 or d2;
        z <= d7 or d5 or d3 or d1;
end dataflow;
```

> Behavioural/Algorithmic Model:
```
library IEEE;
use IEEE.std_logic_1164.all;

entity encoder8x3 is
port (d0,d1,d2,d3,d4,d5,d6,d7: in bit;
x,y,z: out bit);
end encoder8x3;

architecture algorithmic of encoder8x3 is
begin
p0: process (d7,d6,d5,d4,d3,d2,d1,d0)
variable s: bit_vector (7 downto 0);

begin
s:= d7&d6&d5&d4&d3&d2&d1&d0;
 if s = "00000001" then x <= '0'; y <= '0'; z <= '0';
 elsif s = "00000010" then x <= '0'; y <= '0'; z <= '1';
 elsif s = "00000100" then x <= '0'; y <= '1'; z <= '0';
 elsif s = "00001000" then x <= '0'; y <= '1'; z <= '1';
 elsif s = "00010000" then x <= '1'; y <= '0'; z <= '0';
 elsif s = "00100000" then x <= '1'; y <= '0'; z <= '1';
 elsif s = "01000000" then x <= '1'; y <= '1'; z <= '0';
 elsif s = "10000000" then x <= '1'; y <= '1'; z <= '1';
 end if;

end process;
end algorithmic;
```

> Structural Model:

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

entity encoder8x3 is
        port(d0,d1,d2,d3,d4,d5,d6,d7: in bit;
     x,y,z: out bit);
end encoder8x3;

architecture structure of encoder8x3 is

   component or4 -- πύλη or 4 εισόδων
    port(i0,i1,i2,i3: in bit; F: out bit);
   end component;

        begin
        u0: or4 port map (d7,d6,d5,d4,x);
        u1: or4 port map (d7,d6,d3,d2,y);
         u2: or4 port map (d7,d5,d3,d1,z);
end structure;

entity or4 is
        port (i0,i1,i2,i3: in bit; F: out bit);
end or4;

architecture behave of or4 is
        begin F<= i0 or i1 or i2 or i3;
end behave;
```

Testbench

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity test_encoder8x3 is
end test_encoder8x3;

architecture testbench of test_encoder8x3 is

component encoder8x3
port(d0,d1,d2,d3,d4,d5,d6,d7: in bit; x,y,z: out bit);
end component;
```

```
signal tb_d0:bit; signal tb_d1:bit;
signal tb_d2:bit; signal tb_d3:bit;
signal tb_d4:bit; signal tb_d5:bit;
signal tb_d6:bit; signal tb_d7:bit;
signal tb_x:bit; signal tb_y:bit; signal tb_z:bit;

begin
u0: encoder8x3 port map(
d0 => tb_d0, d1 => tb_d1, d2 => tb_d2, d3 => tb_d3,
d4 => tb_d4, d5 => tb_d5, d6 => tb_d6, d7 => tb_d7,
x => tb_x, y => tb_y, z => tb_z);

process
begin
--input: 00000001 -> output:000
tb_d0 <= '1'; tb_d1 <= '0';
tb_d2 <= '0'; tb_d3 <= '0';
tb_d4 <= '0'; tb_d5 <= '0';
tb_d6 <= '0'; tb_d7 <= '0';
wait for 10 ns;

--input: 00000010 -> output:001
tb_d0 <= '0'; tb_d1 <= '1';
tb_d2 <= '0'; tb_d3 <= '0';
tb_d4 <= '0'; tb_d5 <= '0';
tb_d6 <= '0'; tb_d7 <= '0';
wait for 10 ns;

--input: 00000100 -> output:010
tb_d0 <= '0'; tb_d1 <= '0';
tb_d2 <= '1'; tb_d3 <= '0';
tb_d4 <= '0'; tb_d5 <= '0';
tb_d6 <= '0'; tb_d7 <= '0';
wait for 10 ns;

--input: 00001000 -> output:011
tb_d0 <= '0'; tb_d1 <= '0';
tb_d2 <= '0'; tb_d3 <= '1';
tb_d4 <= '0'; tb_d5 <= '0';
tb_d6 <= '0'; tb_d7 <= '0';
wait for 10 ns;

--input: 00010000 -> output:100
tb_d0 <= '0'; tb_d1 <= '0';
tb_d2 <= '0'; tb_d3 <= '0';
tb_d4 <= '1'; tb_d5 <= '0';
tb_d6 <= '0'; tb_d7 <= '0';
wait for 10 ns;
```

```
--input: 00100000 -> output:101
tb_d0 <= '0'; tb_d1 <= '0';
tb_d2 <= '0'; tb_d3 <= '0';
tb_d4 <= '0'; tb_d5 <= '1';
tb_d6 <= '0'; tb_d7 <= '0';
wait for 10 ns;

--input: 01000000 -> output:110
tb_d0 <= '0'; tb_d1 <= '0';
tb_d2 <= '0'; tb_d3 <= '0';
tb_d4 <= '0'; tb_d5 <= '0';
tb_d6 <= '1'; tb_d7 <= '0';
wait for 10 ns;

--input: 10000000 -> output:111
tb_d0 <= '0'; tb_d1 <= '0';
tb_d2 <= '0'; tb_d3 <= '0';
tb_d4 <= '0'; tb_d5 <= '0';
tb_d6 <= '0'; tb_d7 <= '1';
wait for 10 ns;

end process;
end testbench;
```