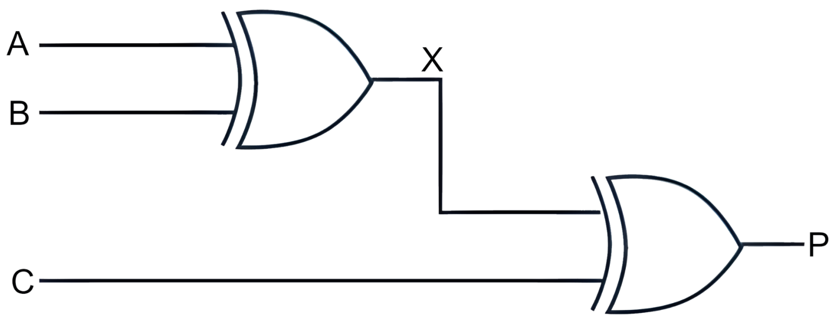
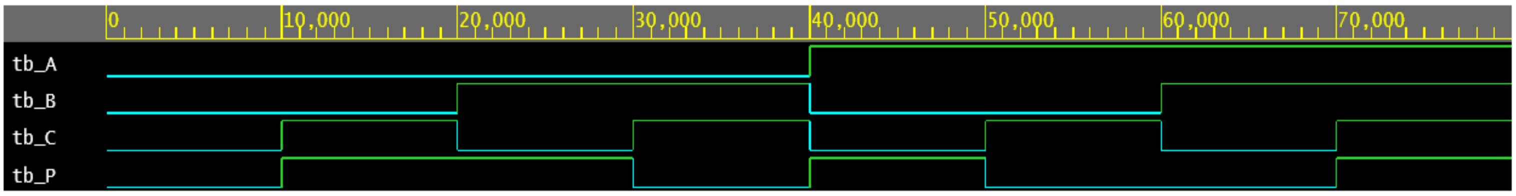
3 bit Even Parity Generator.

A	B	C	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Assume we have a 3-bit even parity generator. Suppose the three inputs A, B and C are applied to the circuit and the output bit is the parity bit P. The total number of 1's must be even to generate the even parity bit P. Based on the above circuit, we see that we need 2 gates XOR where X = A xor B and P = X xor C. X acts as the output of the first XOR and input of the second. In all 3 circuits the result of the simulation is as follows

Run Time: 80ns



> For instance:
At 0 ns A = 0, B = 0, C = 0 → **P = 0**
At 10 ns έχουμε A = 0, B = 0, C = 1 → **P = 1**
At 40 ns έχουμε A = 1, B = 0, C = 0 → **P = 1**
At 60 ns έχουμε A = 1, B = 1, C = 0 → **P = 0**

By comparing the simulation results with the truth table we see that we have achieved the desired result

Circuit Implementations:

*The testbench remains the same for all three models.

> DataFlow Model:

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

entity ParityGenenerator is
        port(A,B,C: in bit;X: inout bit; P: out bit);
end ParityGenenerator;

architecture dataflow of ParityGenenerator is
begin
        X <= A xor B;
        P <= X xor C;
end dataflow;
```

> Behavioural/Algorithmic Model:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity ParityGenenerator is
        port(A,B,C: in bit; X: inout bit; P: out bit);
end ParityGenenerator;

architecture algorithmic of ParityGenenerator is
begin
-- Process for assigning value to X
p0: process (A,B)
variable s: bit_vector (1 downto 0);
begin
s:= A&B;
   if s ="00" then X <= '0';
        elsif  s ="01" then X <= '1';
        elsif  s ="10" then X <= '1';
        elsif  s ="11" then X <= '0';
 end if;
end process;
-- Process for assigning value to P
p1: process (X,C)
variable s: bit_vector (1 downto 0);
begin
s:= X&C;
   if s ="00" then P <= '0';
        elsif  s ="01" then P <= '1';
        elsif  s ="10" then P <= '1';
        elsif  s ="11" then P <= '0';
 end if;
end process;
end algorithmic;
```

> Structural Model:

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

entity ParityGenenerator is
	port(A,B,C: in bit; X: inout bit; P: out bit);
end ParityGenenerator;

entity xor2 is -- XOR 2 εισόδων
  port (X1,X2: in bit; O: out bit);
end xor2;

architecture structure of ParityGenenerator is

  component xor2
     port(X1,X2: in bit; O: out bit);
  end component;

begin
  u0: xor2 port map (A,B,X);
  u1: xor2 port map (X,C,P);
end structure;

architecture behave of xor2 is
begin
  O<= X1 xor X2;
end behave;
```

Testbench:

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

entity test_ParityGenenerator is
end test_ParityGenenerator;

architecture testbench of test_ParityGenenerator is
  component ParityGenenerator
	port(A,B,C: in bit; X: inout bit; P: out bit);
  end component;
signal tb_A: bit;
signal tb_B: bit;
signal tb_C: bit;
signal tb_P: bit;

begin
u0: ParityGenenerator port map(
	A => tb_A,
	B => tb_B,
	C => tb_C,
	P => tb_P);
```

```
process
  begin
  tb_C <= '0' ;
  tb_B <= '0' ;
  tb_A <= '0' ;
  wait for 10 ns;

  tb_C <= '1' ;
  tb_B <= '0' ;
  tb_A <= '0' ;
  wait for 10 ns;

  tb_C <= '0' ;
  tb_B <= '1' ;
  tb_A <= '0' ;
  wait for 10 ns;

  tb_C <= '1' ;
  tb_B <= '1' ;
  tb_A <= '0' ;
  wait for 10 ns;

  tb_C <= '0' ;
  tb_B <= '0' ;
  tb_A <= '1' ;
  wait for 10 ns;

  tb_C <= '1' ;
  tb_B <= '0' ;
  tb_A <= '1' ;
  wait for 10 ns;

  tb_C <= '0' ;
  tb_B <= '1' ;
  tb_A <= '1' ;
  wait for 10 ns;

  tb_C <= '1' ;
  tb_B <= '1' ;
  tb_A <= '1' ;
  wait for 10 ns;

  end process;
end testbench;
```