# Multiplexer

## Theory: -

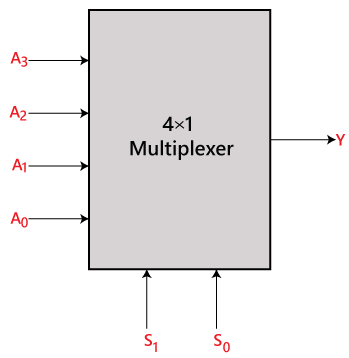
A multiplexer is a combinational circuit that has 2n input lines and a single output line. Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from the input lines and directed to the output line. On the basis of the values of the selection lines, one of these data inputs will be connected to the output.

Unlike encoder and decoder, there are n selection lines and 2n input lines. So, there is a total of 2Npossible combinations of inputs. A multiplexer is also treated as **Mux**

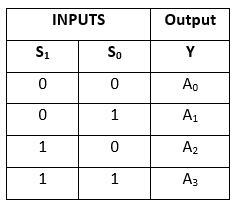
### 4×1 Multiplexer:

In the 4×1 multiplexer, there is a total of four inputs, i.e., A0, A1, A2, and A3, 2 selection lines, i.e., S0 and S1 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S0 and S1, one of these 4 inputs are connected to the output. The block diagram and the truth table of the 4**×**1 multiplexer are given below.

### Block Diagram:



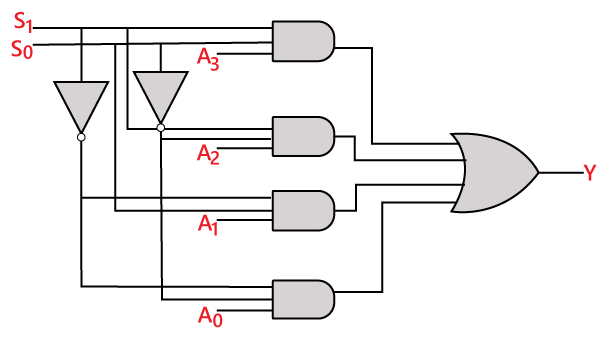
### Truth Table:



The logical expression of the term Y is as follows:

Y=S1' S0' A0+S1' S0 A1+S1 S0' A2+S1 S0 A3

Logical circuit of the above expression is given below:



## Source Code

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity mux\_4to1 is

port(

A,B,C,D : in STD\_LOGIC;

S0,S1: in STD\_LOGIC;

Z: out STD\_LOGIC

);

end mux\_4to1;

architecture bhv of mux\_4to1 is

begin

process (A,B,C,D,S0,S1) is

begin

if (S0 ='0' and S1 = '0') then

Z <= A;

elsif (S0 ='1' and S1 = '0') then

Z <= B;

elsif (S0 ='0' and S1 = '1') then

Z <= C;

else

Z <= D;

end if;

end process;

end bhv;

## Testbench Code

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

entity mux\_4to1\_tb is

end;

architecture bench of mux\_4to1\_tb is

component mux\_4to1

port(

A,B,C,D : in STD\_LOGIC;

S0,S1: in STD\_LOGIC;

Z: out STD\_LOGIC

);

end component;

signal A,B,C,D: STD\_LOGIC;

signal S0,S1: STD\_LOGIC;

signal Z: STD\_LOGIC ;

begin

uut: mux\_4to1 port map ( A => A,

B => B,

C => C,

D => D,

S0 => S0,

S1 => S1,

Z => Z );

stimulus: process

begin

-- Put initialisation code here

A <= '1';

B <= '1';

C <= '1';

D <= '1';

S0 <= '0'; S1 <= '0';

wait for 10 ns;

S0 <= '1'; S1 <= '0';

wait for 10 ns;

S0 <= '0'; S1 <= '1';

wait for 10 ns;

S0 <= '0'; S1 <= '1';

wait for 10 ns;

S0 <= '1'; S1 <= '1';

wait for 10 ns;

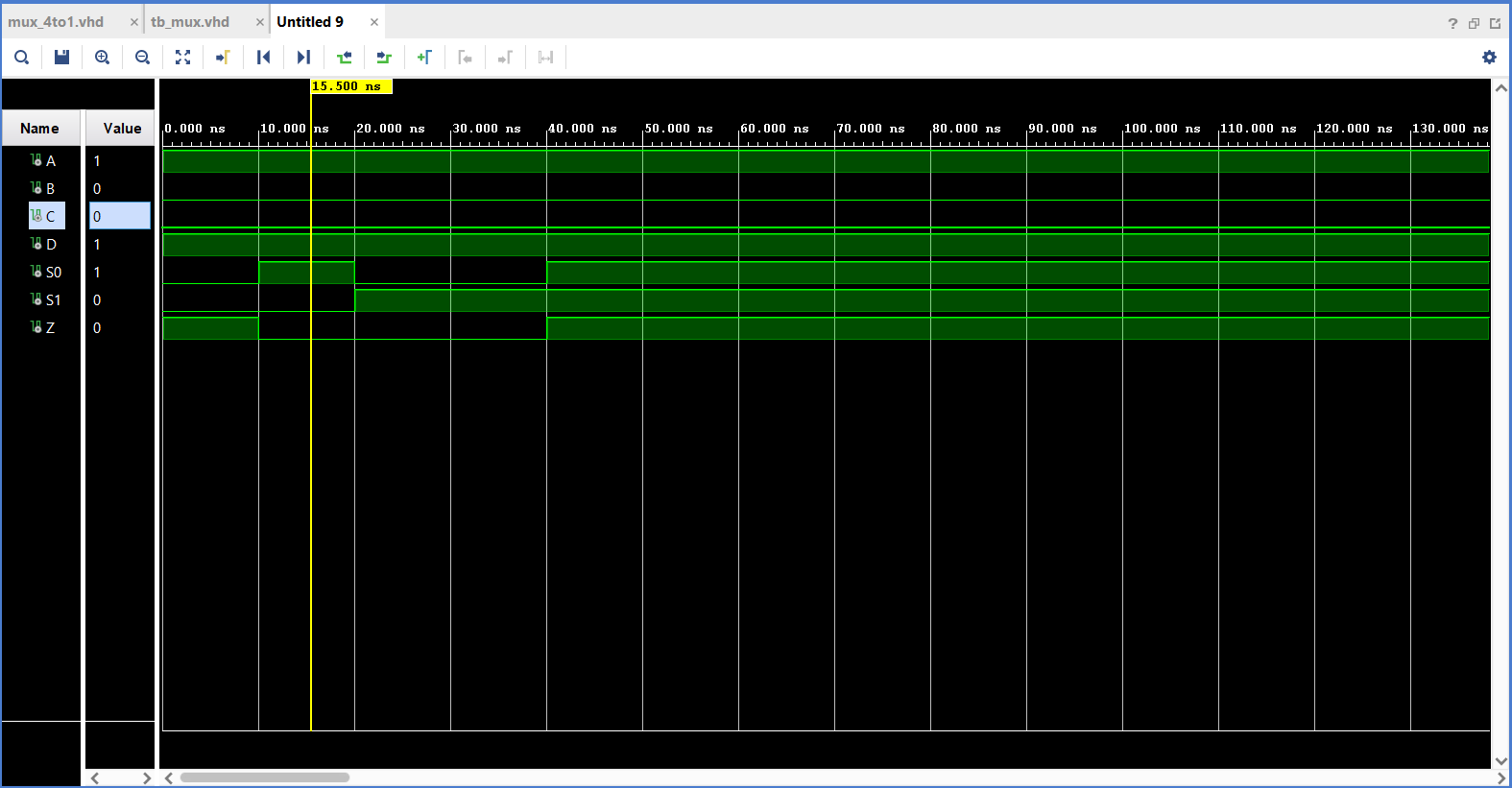
-- Put test bench stimulus code here

wait;

end process;

end;

## Observation



## Output

