Name: ADITYA BADAYALYA

Enrolment Number: 510819056

COMPUTER ORGANIZATION AND ARCHITECHTURE LAB(Hy)

ASSIGNMENT-5

**1.Design and simulate 4:1 MUX using If-else construct. Use protection for high impedance (Z) states in inputs.**

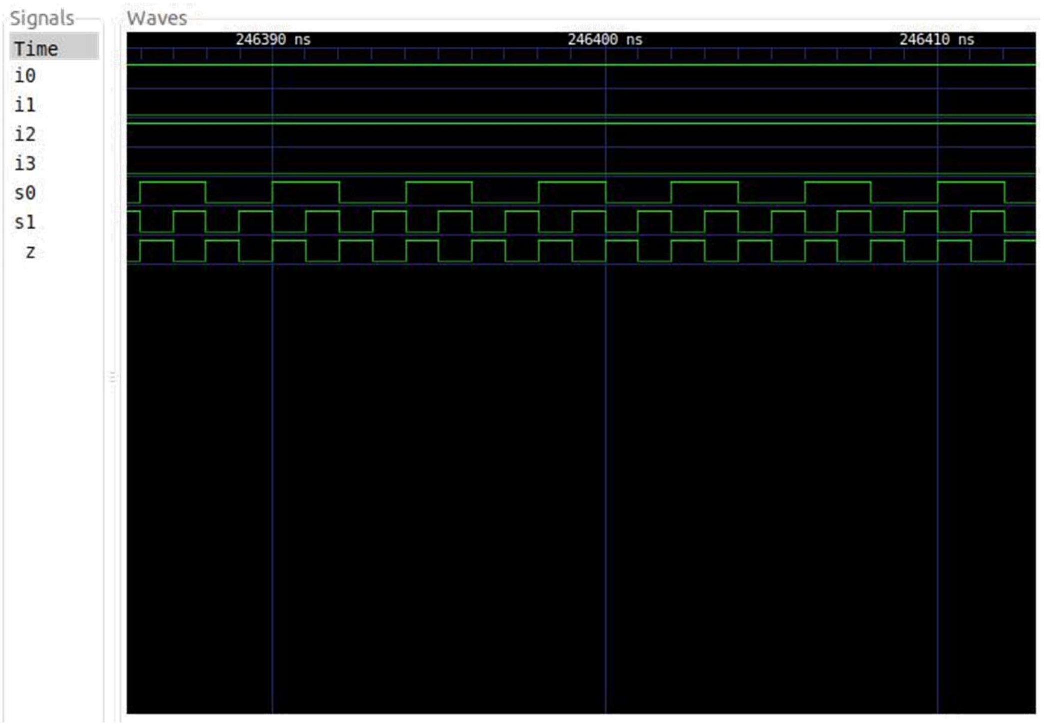
## <u>TESTBENCH</u>

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;
entity MUXif is
-- empty
end MUXif;
architecture tb of MUXif is
-- DUT component
component muxi is
port(
 s0: in std_logic;
 s1: in std_logic;
 i0: in std_logic;
 i1: in std_logic;
 i2: in std_logic;
 i3: in std_logic;
 z:  out std_logic );
end component;
signal s0,s1,i0,i1,i2,i3,z : std_logic;
```

```
begin
 -- Connect DUT
 DUT: muxi port map(s0,s1,i0,i1,i2,i3,z);
 process
 begin
 i0<='1';
 i1<='0';
 i2<='1';
 i3<='0';

 s0 <= '0';
 s1 <= '0';
 wait for 1 ns;
 s0 <= '0';
 s1 <= '1';
 wait for 1 ns;

 s0 <= '1';
 s1 <= '0';
 wait for 1 ns;

 s0 <= '1';
 s1 <= '1';
 wait for 1 ns;
 end process;
 end tb;
```

## DESIGN

```
-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;
entity muxi is port(s0,s1: in
std_logic; i0,i1,i2,i3: in
std_logic;
z: out std_logic);
end muxi;
architecture behavioral of muxi is
begin process(s0,s1,i0,i1,i2,i3)

 begin
if (s0 = '0' and s1 = '0') then
z<=i0;
elsif ( s0 = '0' and s1 = '1') then
z<=i1;
elsif ( s0 = '1' and s1 = '0') then
z<=i2;
 elsif (s0 = '1' and s1 = '1') then
z<=i3;
 end if;
 end process;
end behavioral;
```


Signals
Waves
Time
i0
i1
i2
i3
s0
s1
z
246390 ns
246400 ns
246410 ns

## 2. <u>Design and simulate 4:1 MUX using Case construct. Use protection for high impedance (Z) states in inputs.</u>

**TESTBENCH**

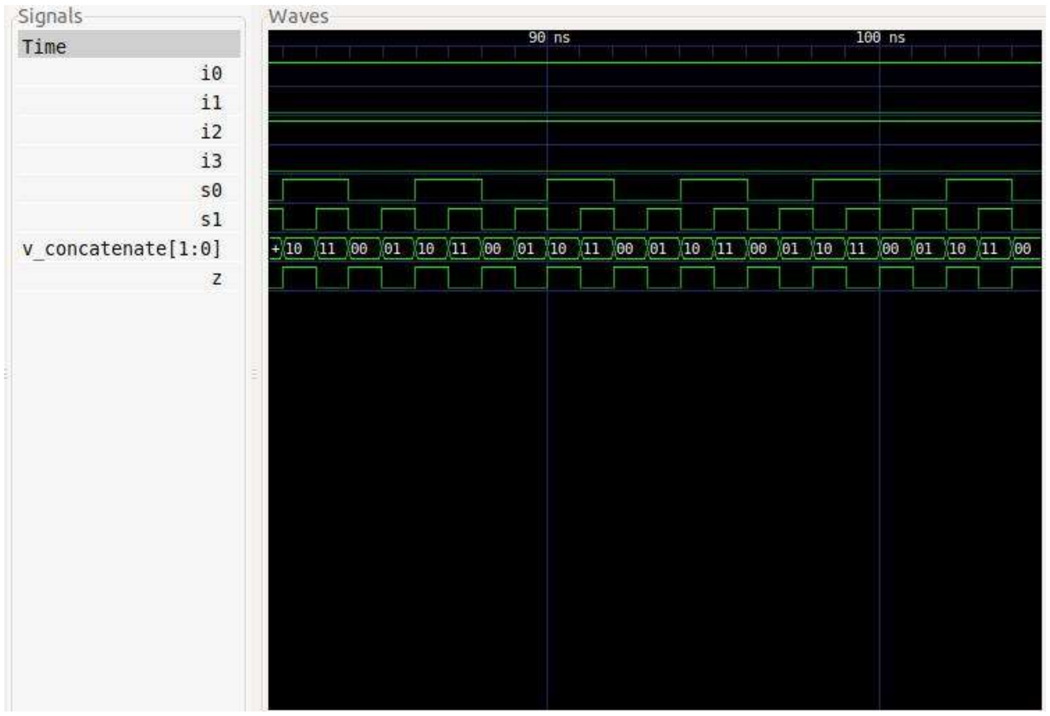
```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;
entity MUXcase is
-- empty
end MUXcase;
architecture tb of MUXcase is
-- DUT component
component muxcas is
port(
 s0: in std_logic;
 s1: in std_logic;
 i0: in std_logic;
 i1: in std_logic;
 i2: in std_logic;
 i3: in std_logic;
 z: out std_logic );
end component;
signal s0,s1,i0,i1,i2,i3,z : std_logic;
begin
 -- Connect DUT
```

```
DUT: muxcas port map(s0,s1,i0,i1,i2,i3,z);
process
begin
i0<='1';
i1<='0';
i2<='1';
i3<='0';

s0 <= '0';
s1 <= '0';
wait for 1 ns;
s0 <= '0';
s1 <= '1';
wait for 1 ns;

s0 <= '1';
s1 <= '0';
wait for 1 ns;

s0 <= '1';
s1 <= '1';
wait for 1 ns;
end process;
end tb;
```

## DESIGN

```
-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all; use
ieee.numeric_std.all; entity muxcas is
port(s0,s1,i0,i1,i2,i3: in std_logic; z:
out std_logic);

end muxcas;
architecture behavioral of muxcas is
 signal v_CONCATENATE : std_logic_vector(1 downto 0);

begin
 v_CONCATENATE <=s0 & s1;
 process (v_CONCATENATE,s0,s1)
 begin
case (v_CONCATENATE ) is
when ("00") =>
z<=i0;
when ("01") =>
z<=i1;
when ("10") =>
z<=i2;
when ("11") =>
 z<=i3;
 when others =>
z<='0';
end case;
```

```
 end process;
end behavioral;
```



## **3.Design and simulate 3-to-8 DECODER using If-else construct.**

### **TESTBENCH**

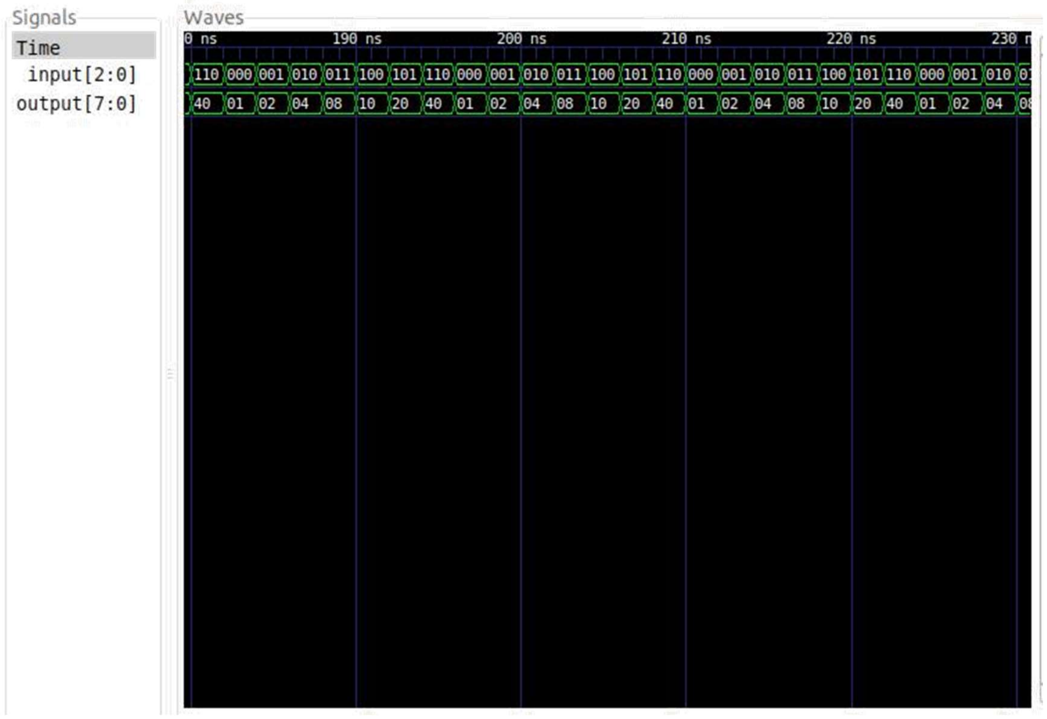
```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;
entity decoder38 is
-- empty
end decoder38;
architecture tb of decoder38 is
-- DUT component
component dec3 is
port(
input: in std_logic_vector(2 downto 0):=(others=>'0');
output: out std_logic_vector(7 downto 0):=(others=>'0') );

end component;
signal input: std_logic_vector(2 downto 0);
signal output: std_logic_vector(7 downto 0);
begin
 -- Connect DUT
 DUT: dec3 port map(input,output);
 process
 begin
input<="000"; --input = 0.
```

```
wait for 2 ns;

input<="001"; --input = 1.
wait for 2 ns;

input<="010"; --input = 2.

wait for 2 ns;

input<="011"; --input = 3.

wait for 2 ns;

input<="100"; --input = 4.

wait for 2 ns;

input<="101"; --input = 5.

wait for 2 ns;

input<="110"; --input = 6.

wait for 2 ns;

input<="111"; --input = 7.
 end process;
end tb;
```

## **DESIGN**

```
--libraries to be used are specified here
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--entity declaration with port definitions entity
dec3 is
port( input : in std_logic_vector(2 downto 0); --3 bit input output :

 out std_logic_vector(7 downto 0) -- 8 bit ouput );
```

```
end dec3;
--architecture of entity

architecture Behavioral of dec3 is

begin

process(input)

begin

if input = "000" then

output <= "00000001";

elsif input = "001" then

output<="00000010";

elsif input = "010" then

output<="00000100";

elsif input = "011" then

output<="00001000";

elsif input = "100" then

output<="00010000";

elsif input = "101" then

output<="00100000";

elsif input = "110" then

output<="01000000";

elsif input = "111" then

output<="10000000";

end if;

end process;

end Behavioral;
```


Signals
Time
input[2:0]
output[7:0]
Waves
0 ns
190 ns
200 ns
210 ns
220 ns
230 n
110 000 001 010 011 100 101 110 000 001 010 011 100 101 110 000 001 010 011 100 101 110 000 001 010 01
40 01 02 04 08 10 20 40 01 02 04 08 10 20 40 01 02 04 08 10 20 40 01 02 04 08

## 4. Design and simulate 3-to-8 DECODER using Case construct. Use protection for high impedance (Z) states in inputs.

## TESTBENCH

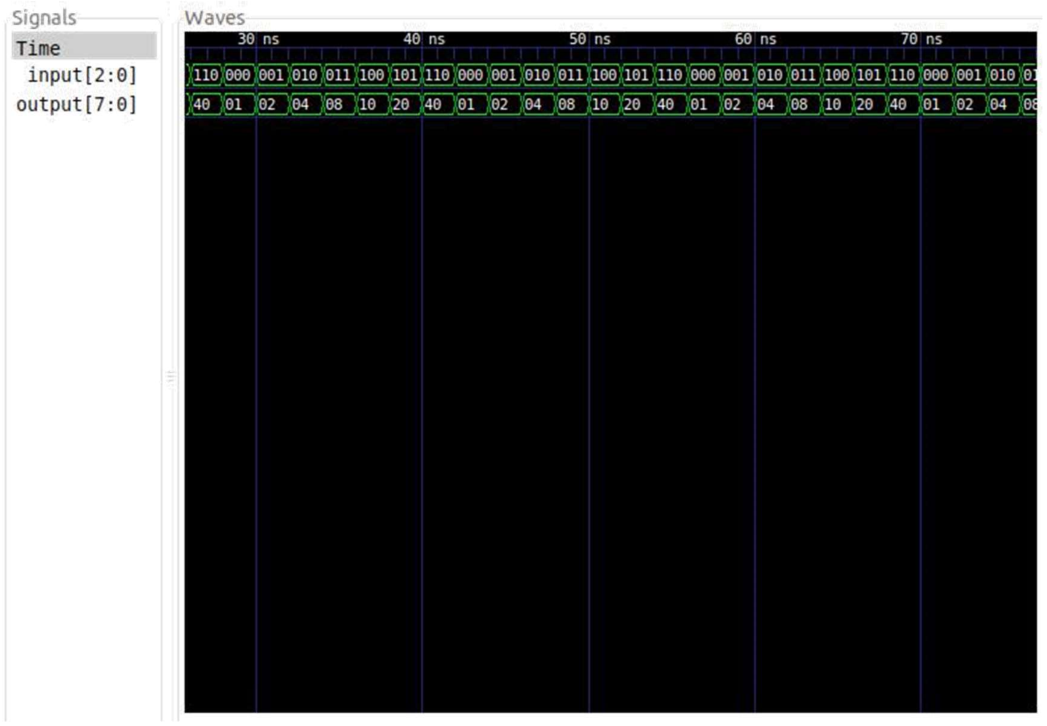
```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;
entity deccase is
-- empty
end deccase ;
architecture tb of deccase is
-- DUT component
component DECc is
port(
input: in std_logic_vector(2 downto 0):=(others=>'0');
output: out std_logic_vector(7 downto 0):=(others=>'0') );

end component;
signal input: std_logic_vector(2 downto 0);
signal output: std_logic_vector(7 downto 0);
begin
 -- Connect DUT
 DUT: DECc port map(input,output);
 process
 begin
```

```
input<="000"; --input = 0.
wait for 2 ns;
input<="001"; --input = 1.
wait for 2 ns;
input<="010"; --input = 2.
wait for 2 ns;
input<="011"; --input = 3.
wait for 2 ns;
input<="100"; --input = 4.
wait for 2 ns;
input<="101"; --input = 5.
wait for 2 ns;
input<="110"; --input = 6.
wait for 2 ns;
input<="111"; --input = 7.
 end process;
end tb;
```

## **<u>DESIGN</u>**

```
--libraries to be used are specified here
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--entity declaration with port definitions entity
DECc is
port( input : in std_logic_vector(2 downto 0); --3 bit input output :
 out std_logic_vector(7 downto 0) -- 8 bit ouput );
```

```
end DECc ;
--architecture of entity
architecture Behavioral of DECc is
begin
process(input)
begin
case (input) is
when "000"=>
output<="00000001";
when "001"=>
output<="00000010";
when "010"=>
output<="00000100";
when "011"=>
output<="00001000";
when "100"=>
output<="00010000";
when "101"=>
output<="00100000";
when "110"=>
output<="01000000";
when "111"=>
output<="10000000";
when others=>
output<="00000000";
end case;
end process;
end Behavioral;
```


Signals
Time
input[2:0]
output[7:0]
Waves
30 ns
40 ns
50 ns
60 ns
70 ns
110 000 001 010 011 100 101 110 000 001 010 011 100 101 110 000 001 010 011 100 101 110 000 001 010
40 01 02 04 08 10 20 40 01 02 04 08 10 20 40 01 02 04 08 10 20 40 01 02 04