**INDEX**

**S.**

**NO**

**DATE**

**NAME OF**

**THE**

**EXPERIMENT**

**PAGE**

**NO.**

**FACULTY**

**SIGN**

1

Half Adder Logic Implementation and

simulation in FPGA

05

2

Full Adder Logic Implementation and

simulation in FPGA

07

3

Ripple Carry Adder Logic Implementation and

simulation in FPGA

10

4

4

Bit Up Counter Logic Implementation and

simulation in FPGA

13

5

4

Bit Down Counter Logic Implementation and

simulation in FPGA

15

6

Encoder Logic Implementation and simulation

in FPGA

17

7

Decoder Logic Implementation and simulation

in FPGA

20

8

Multiplexer Logic Implementation and

simulation in FPGA

23

9

Demultiplexer Logic Implementation and

simulation in FPGA

26

29

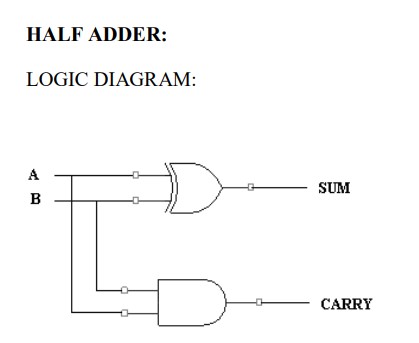
10

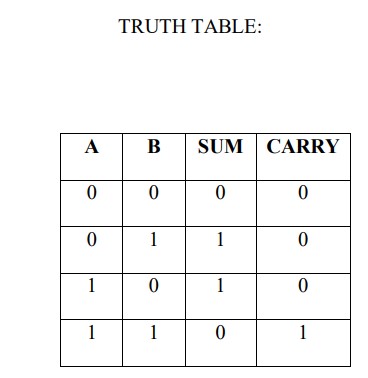
Shift register Serial In Serial Out Logic

Implementation and simulation in FPGA

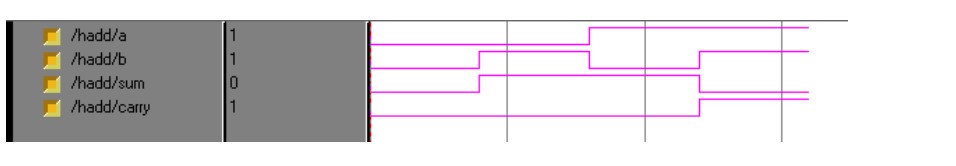
# Experiment 1

**Half Adder Logic Implementation and simulation in FPGA**





**Simulation Output**



# Experiment 1

**Half Adder Logic Implementation and simulation in FPGA**

**Aim:**

To develop the source code for Half adder by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Dataflow Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL; use IEEE.STD\_LOGIC\_UNSIGNED.ALL; entity hadd is

Port ( a : in std\_logic; b : in std\_logic; sum : out std\_logic; carry : out std\_logic); end hadd;

architecture dataflow of hadd is begin sum <= a xor b; carry <= a and b; end dataflow;

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity haddbehavioral is

Port ( a : in std\_logic;

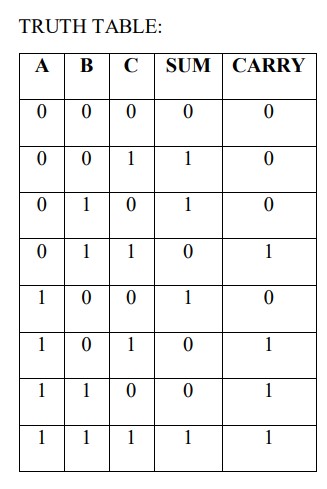
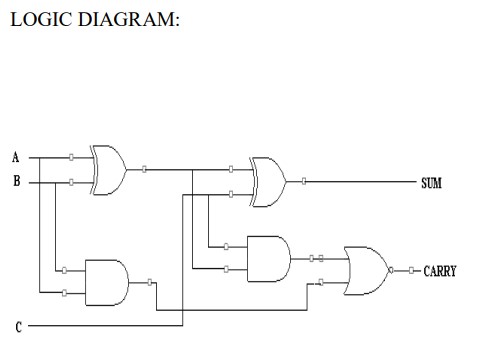
b : in std\_logic; sum : out std\_logic; carry : out std\_logic); end haddbehavioral;

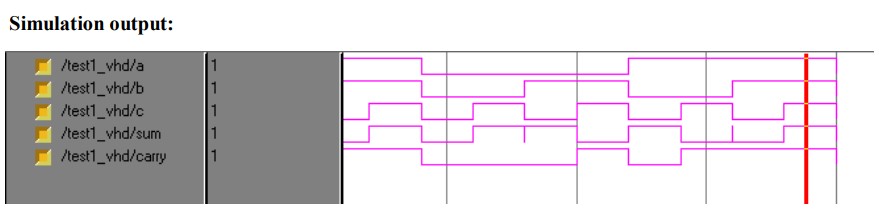
architecture Behavioral of haddbehavioral is begin p1:process (a,b) begin sum<= a xor b; carry<= a and b; end process p1; end Behavioral;

**Result:** Half Adder was implemented and the result was simulated in FPGA ISE Design Suite.

# Experiment 2

**Full Adder Logic Implementation and simulation in FPGA**





# Experiment 2

**Full Adder Logic Implementation and simulation in FPGA**

**Aim:**

To develop the source code for Full adder by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Dataflow Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity fadd\_dataflow is Port ( a : in std\_logic; b : in std\_logic; c : in std\_logic; sum : out std\_logic; carry : out std\_logic); end fadd\_dataflow;

architecture dataflow of fadd\_dataflow is

signal p,q,r,s:std\_logic;

begin p<= a xor b; q<= a and b; r<= b and c; s<= c and a; sum<= p xor c; carry<= q or r or s; end dataflow; **Behavioral Modeling:**

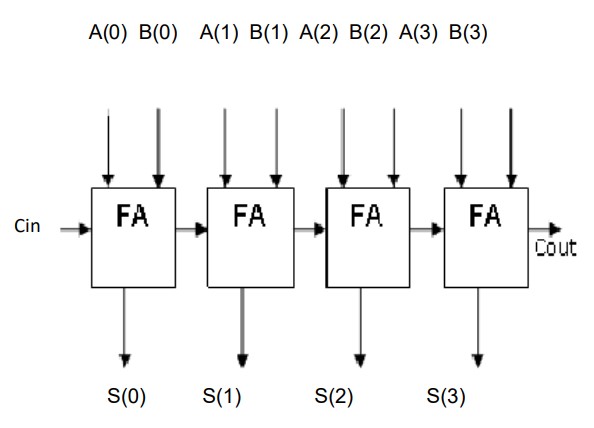
library IEEE;

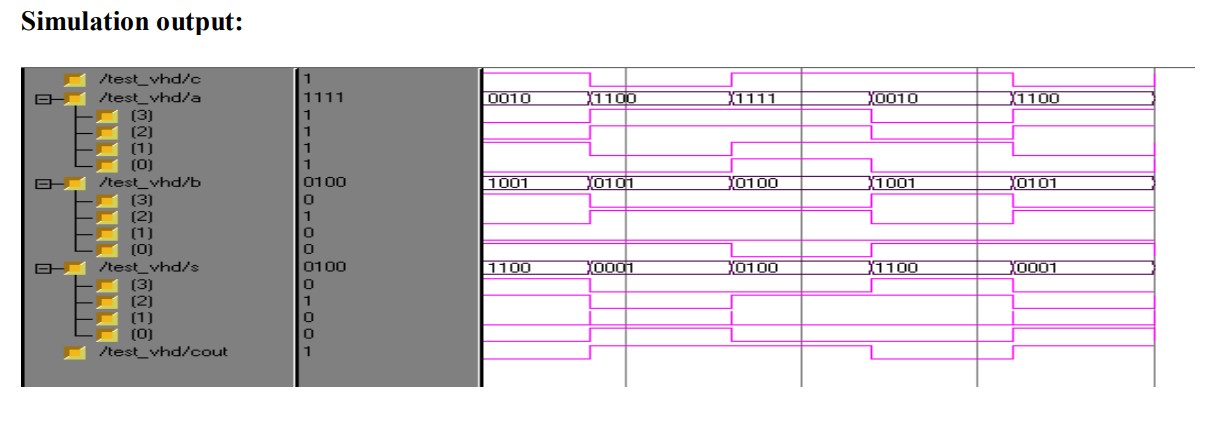
use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL; use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity fadd\_behv is Port ( a : in std\_logic; b : in std\_logic; c : in std\_logic; sum : out std\_logic; carry : out std\_logic); end fadd\_behv; architecture Behavioral of fadd\_behv is begin p1:process(a,b,c) variable r,s,t:std\_logic; begin r:= a and b; s:= b and c; t:= c and a; sum<= a xor b xor c; carry<= r or s or t; end process p1; end Behavioral;

**Result:** Full Adder was implemented and the result was simulated in FPGA ISE Design Suite.

# Experiment 3 Ripple Carry Adder Logic Implementation and simulation in FPGA





# Experiment 3 Ripple Carry Adder Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for Ripple Carry Adder by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity fadd\_behv is Port ( a : in std\_logic; b : in std\_logic; c : in std\_logic; sum : out std\_logic; carry : out std\_logic); end fadd\_behv;

architecture Behavioral of fadd\_behv is begin p1:process(a,b,c) variable r,s,t:std\_logic; begin r:= a and b; s:= b and c; t:= c and a; sum<= a xor b xor c; carry<= r or s or t; end process p1; end Behavioral;

**Structural Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL; entity rca is

Port ( a : in std\_logic\_vector(3 downto 0);

A(0) B(0) A(1) B(1) A(2) B(2) A(3) B(3)

S(0) S(1) S(2) S(3)

Cin

b : in std\_logic\_vector(3 downto 0); c : in std\_logic;

s : out std\_logic\_vector(3 downto 0);

cout : out std\_logic);

end rca;

architecture structural of rca is component fadd\_behv

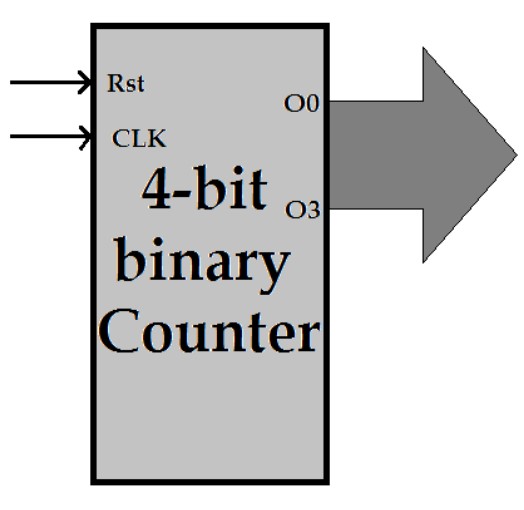
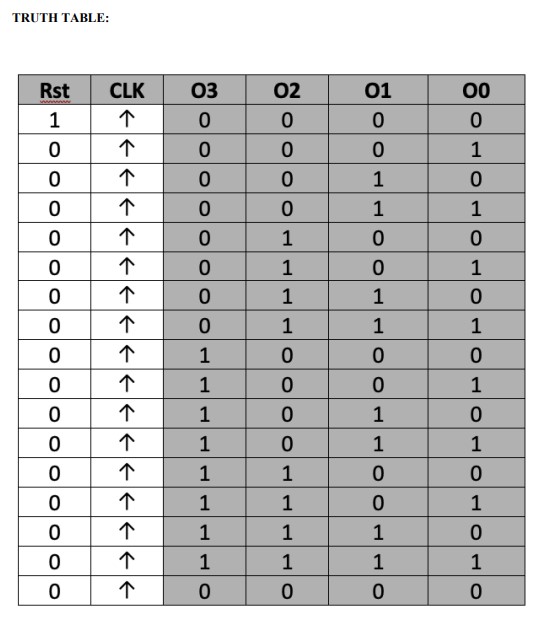
port(a,b,c:in std\_logic; sum,carry:out std\_logic); end component; signal c0,c1,c2:std\_logic;

begin f1:fadd\_behv port map (a(0),b(0),c,s(0),c0); f2:fadd\_behv port map (a(1),b(1),c0,s(1),c1); f3:fadd\_behv port map (a(2),b(2),c1,s(2),c2); f4:fadd\_behv port map (a(3),b(3),c2,s(3),cout); end structural;

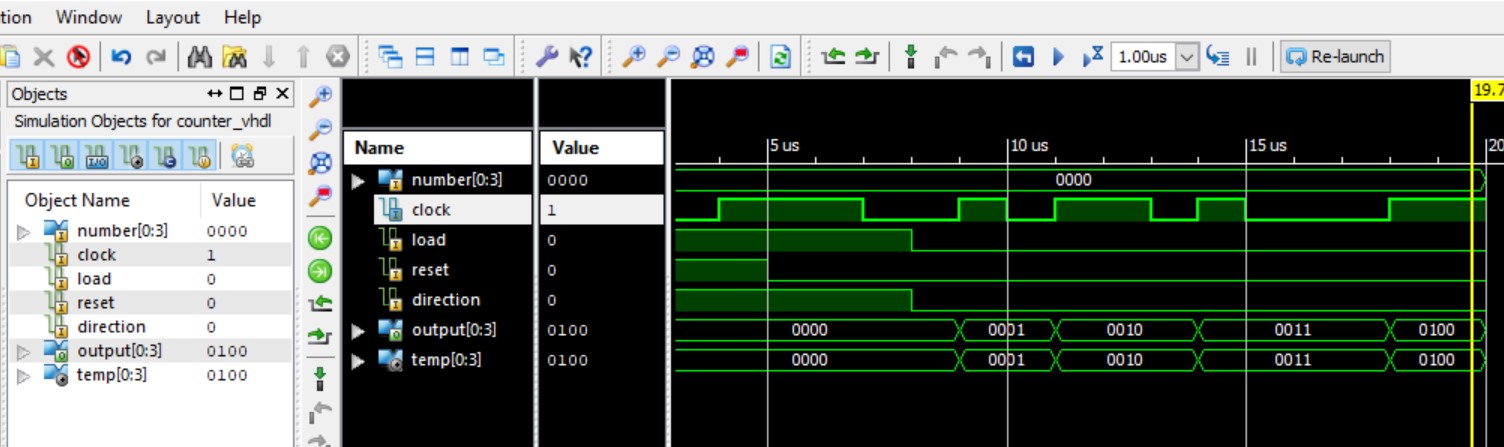
Result: Ripple Carry Adder was implemented and the result was simulated in FPGA ISE Design Suite.

# Experiment 4

4 Bit Up Counter Logic Implementation and simulation in FPGA

Up Counter



# Experiment 4 4 Bit Up Counter Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for 4 Bit Up Counter by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL; use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Counter\_VHDL is

port( Number: in std\_logic\_vector(0 to 3);

Clock: in std\_logic;

Load: in std\_logic;

Reset: in std\_logic;

Direction: in std\_logic;

Output: out std\_logic\_vector(0 to 3) ); end Counter\_VHDL;

architecture Behavioral of Counter\_VHDL is signal temp: std\_logic\_vector(0 to 3); begin

process(Clock,Reset) begin if Reset='1' then temp <= "0000";

elsif ( Clock'event and Clock='1') then if Load='1' then

temp <= Number;

elsif (Load='0' and Direction='0') then temp <= temp + 1;

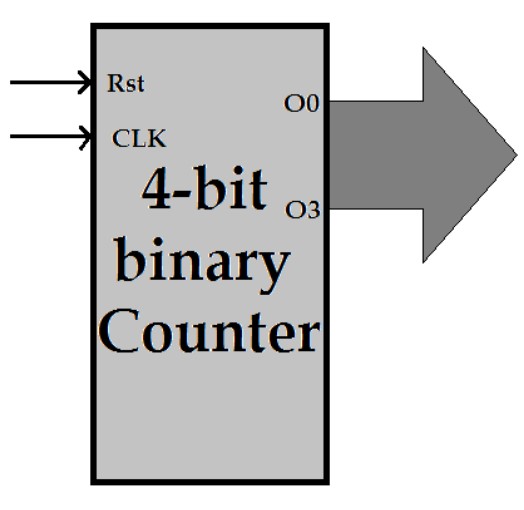
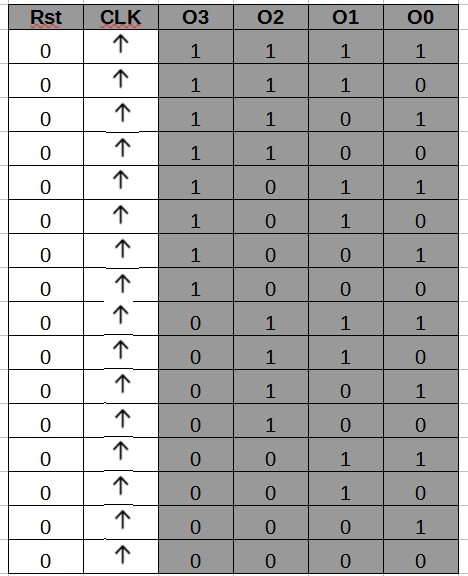
elsif (Load='0' and Direction='1') then

temp <= temp - 1;

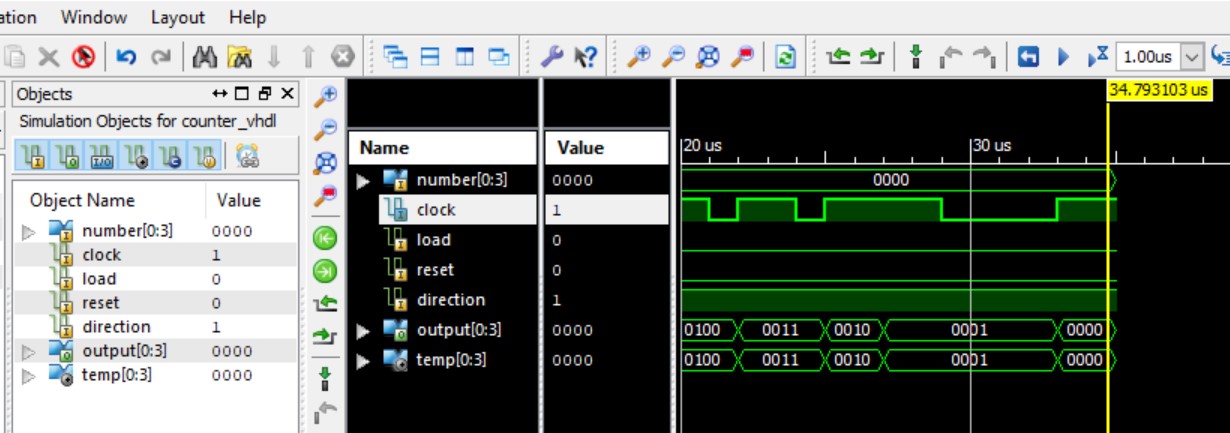
end if; end if; end process; Output <= temp; end Behavioral;

RESULT: Thus the OUTPUT’s of 4 Bit up counter was verified by synthesizing and simulating the VHDL code **Experiment 5**

4 Bit Down Counter Logic Implementation and simulation in FPGA



Down Counter



# Experiment 5 4 Bit Down Counter Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for 4 Bit Down Counter by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL; use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Counter\_VHDL is

port( Number: in std\_logic\_vector(0 to 3);

Clock: in std\_logic;

Load: in std\_logic;

Reset: in std\_logic;

Direction: in std\_logic;

Output: out std\_logic\_vector(0 to 3) ); end Counter\_VHDL;

architecture Behavioral of Counter\_VHDL is signal temp: std\_logic\_vector(0 to 3); begin

process(Clock,Reset) begin if Reset='1' then temp <= "0000";

elsif ( Clock'event and Clock='1') then if Load='1' then

temp <= Number;

elsif (Load='0' and Direction='0') then temp <= temp + 1;

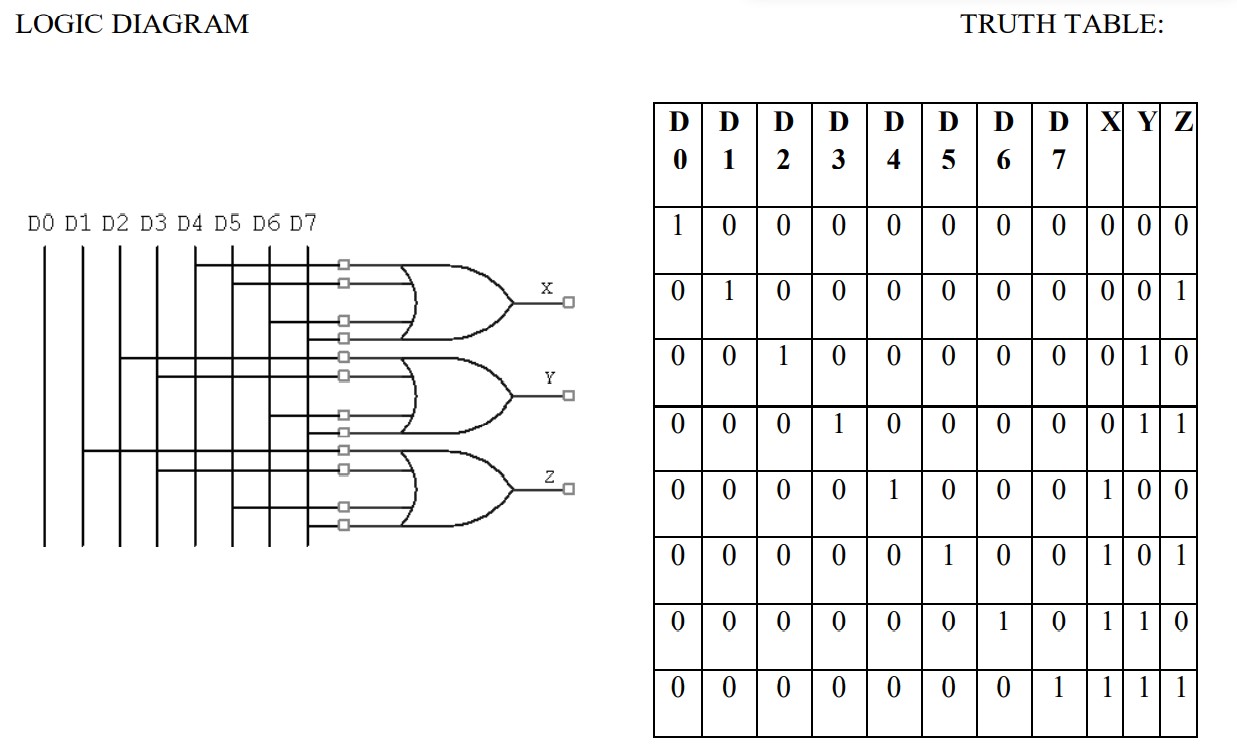
elsif (Load='0' and Direction='1') then

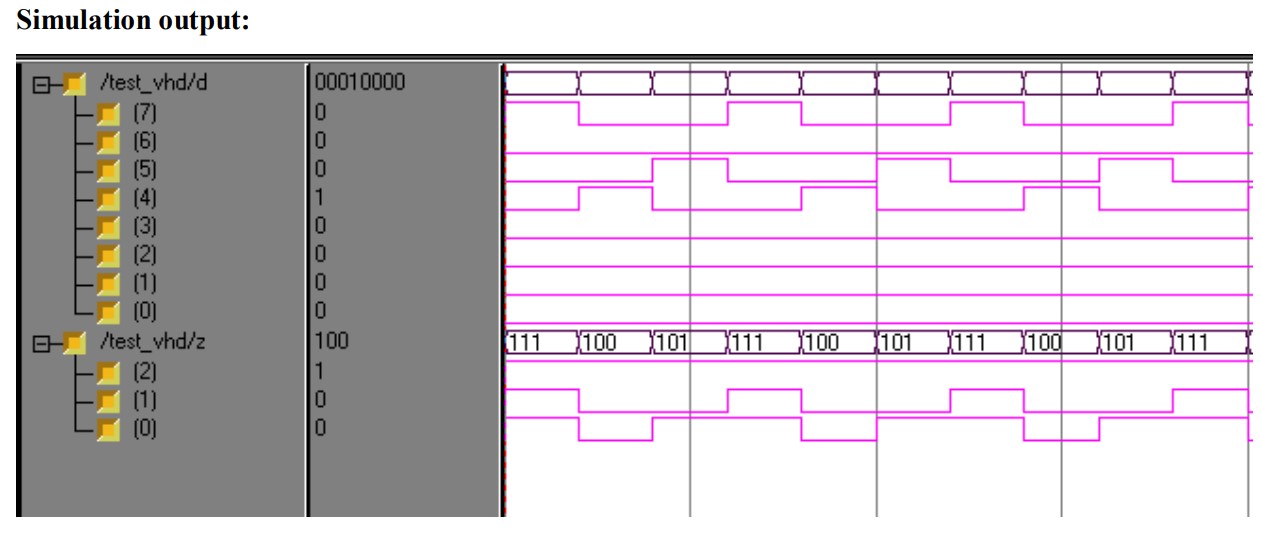
temp <= temp - 1;

end if; end if; end process; Output <= temp; end Behavioral;

RESULT: Thus the OUTPUT’s of 4 Bit down counter was verified by synthesizing and simulating the VHDL code

# Experiment 6 Encoder Logic Implementation and simulation in FPGA





# Experiment 6 Encoder Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for Encoder by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Dataflow Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity encoder\_dataflow is

Port ( d : in std\_logic\_vector(7 downto 0); z : out std\_logic\_vector(2 downto 0));

end encoder\_dataflow;

architecture dataflow of encoder\_dataflow is begin z(2)<= d(4) or d(5) or d(6) or d(7); z(1)<= d(2) or d(3) or d(6) or d(7); z(0)<= d(1) or d(3) or d(5) or d(7); end dataflow;

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity encoder\_behv is

Port ( d : in std\_logic\_vector(7 downto 0); e : in std\_logic;

z : out std\_logic\_vector(2 downto 0)); end encoder\_behv;

architecture Behavioral of encoder\_behv is begin p1:process(d,e) begin if (e='1') then

case d is

when "10000000"=>z<="000"; when "01000000"=>z<="001"; when "00100000"=>z<="010"; when "00010000"=>z<="011"; when "00001000"=>z<="100"; when "00000100"=>z<="101"; when "00000010"=>z<="110"; when "00000001"=>z<="111"; when others=>z<="ZZZ";

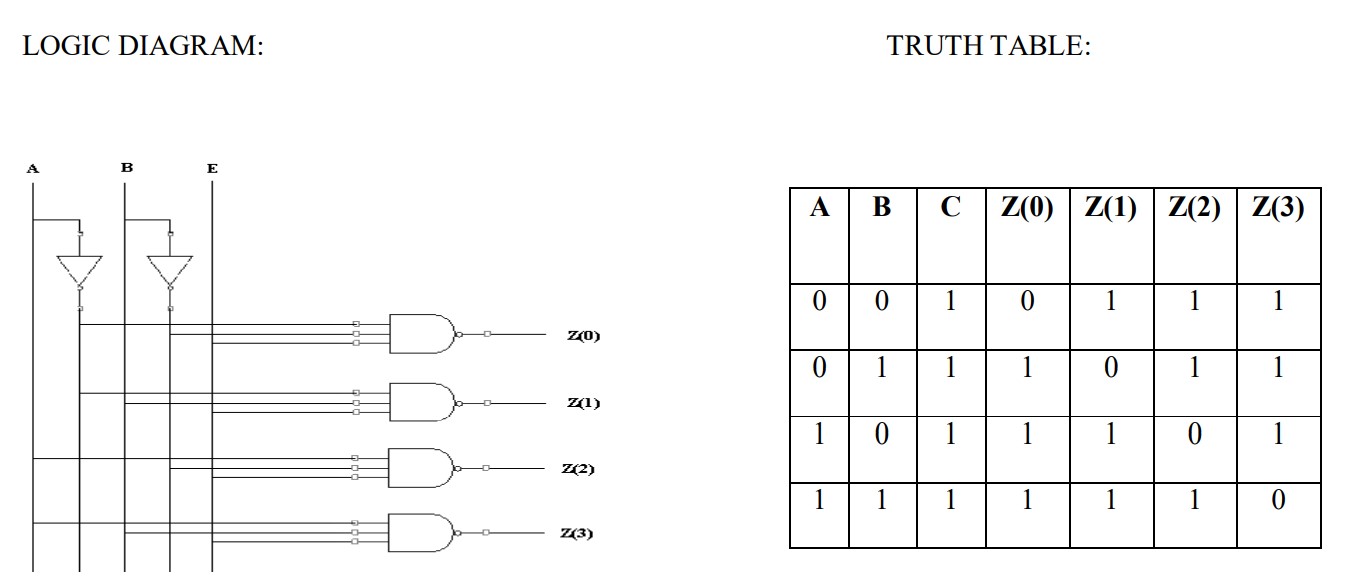
end case; else

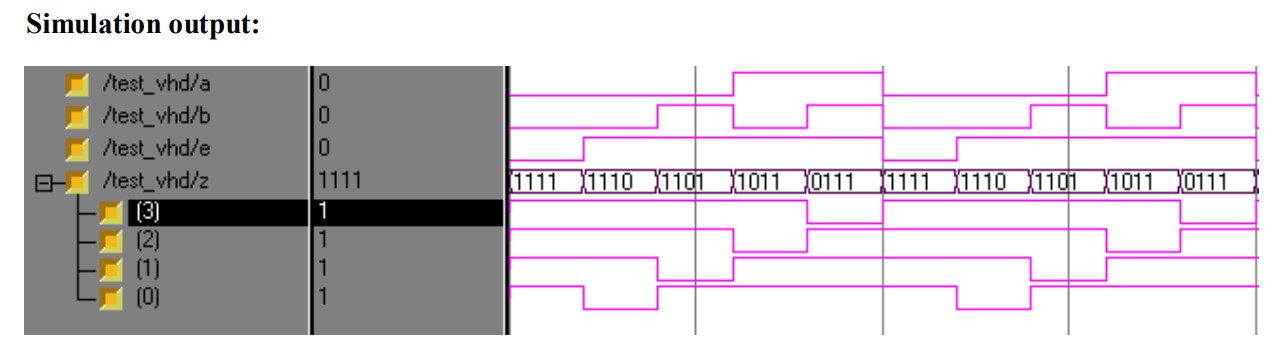
z<="XXX";

end if; end process p1; end Behavioral;

RESULT: Thus the OUTPUT’s of Encoder was verified by synthesizing and simulating the VHDL code

# Experiment 7 Decoder Logic Implementation and simulation in FPGA





# Experiment 7 Decoder Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for Decoder by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Dataflow Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity decoder\_dataflow is Port ( a : in std\_logic; b : in std\_logic; e : in std\_logic;

z : out std\_logic\_vector(3 downto 0));

end decoder\_dataflow;

architecture dataflow of decoder\_dataflow is signal abar,bbar:std\_logic; begin abar<= not a; bbar<= not b; z(0)<= not (abar and bbar and e); z(1)<= not (abar and b and e); z(2)<= not (a and bbar and e); z(3)<= not (a and b and e); end dataflow;

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity decoder\_behv is Port ( a : in std\_logic; b : in std\_logic; e : in std\_logic;

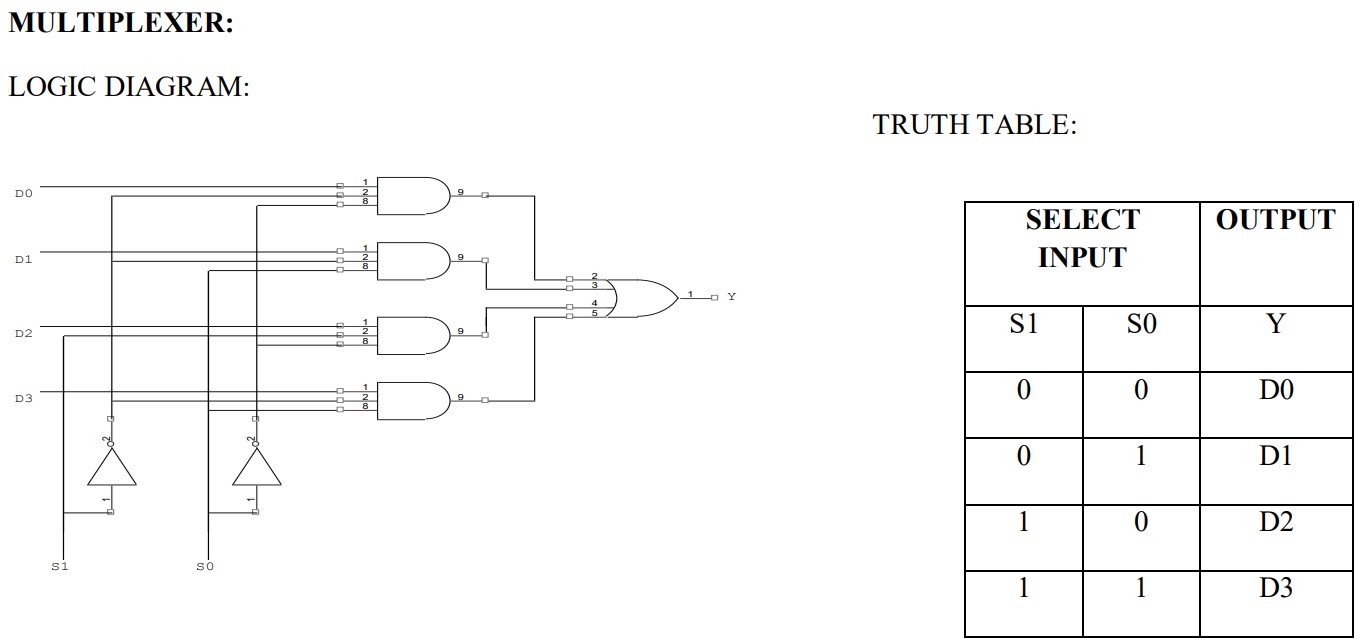
z : out std\_logic\_vector(3 downto 0)); end decoder\_behv;

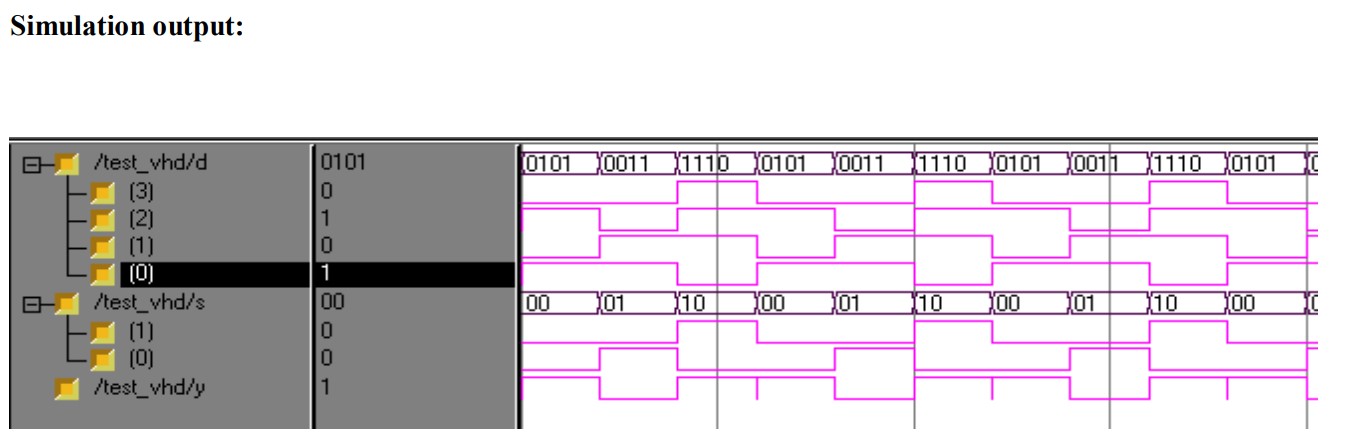
architecture Behavioral of decoder\_behv is begin p1:process(a,b)

begin if (e='1') then z(0)<= not a and not b ; z(1)<= not a and b; z(2)<= a and not b; z(3)<= a and b; else z<="1111"; end if; end process p1; end Behavioral;

RESULT: Thus the OUTPUT’s of Encoder was verified by synthesizing and simulating the VHDL code

# Experiment 8 Multiplexer Logic Implementation and simulation in FPGA





# Experiment 8 Multiplexer Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for Multiplexer by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Dataflow Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity mux\_dataflow is

Port ( d : in std\_logic\_vector(3 downto 0); s : in std\_logic\_vector(1 downto 0);

y : out std\_logic); end mux\_dataflow; architecture dataflow of mux\_dataflow is signal s0bar,s1bar,p,q,r,st:std\_logic; begin p<= d(0) and s0bar and s1bar; q<= d(1) and s0bar and s(1); r<= d(2) and s(0) and s1bar; st<= d(3) and s(0) and s(1); s0bar<= not s(0); s1bar<= not s(1); y<= p or q or r or st; end dataflow;

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity mux\_behv is

Port ( d : in std\_logic\_vector(3 downto 0); s : in std\_logic\_vector(1 downto 0);

y : out std\_logic); end mux\_behv;

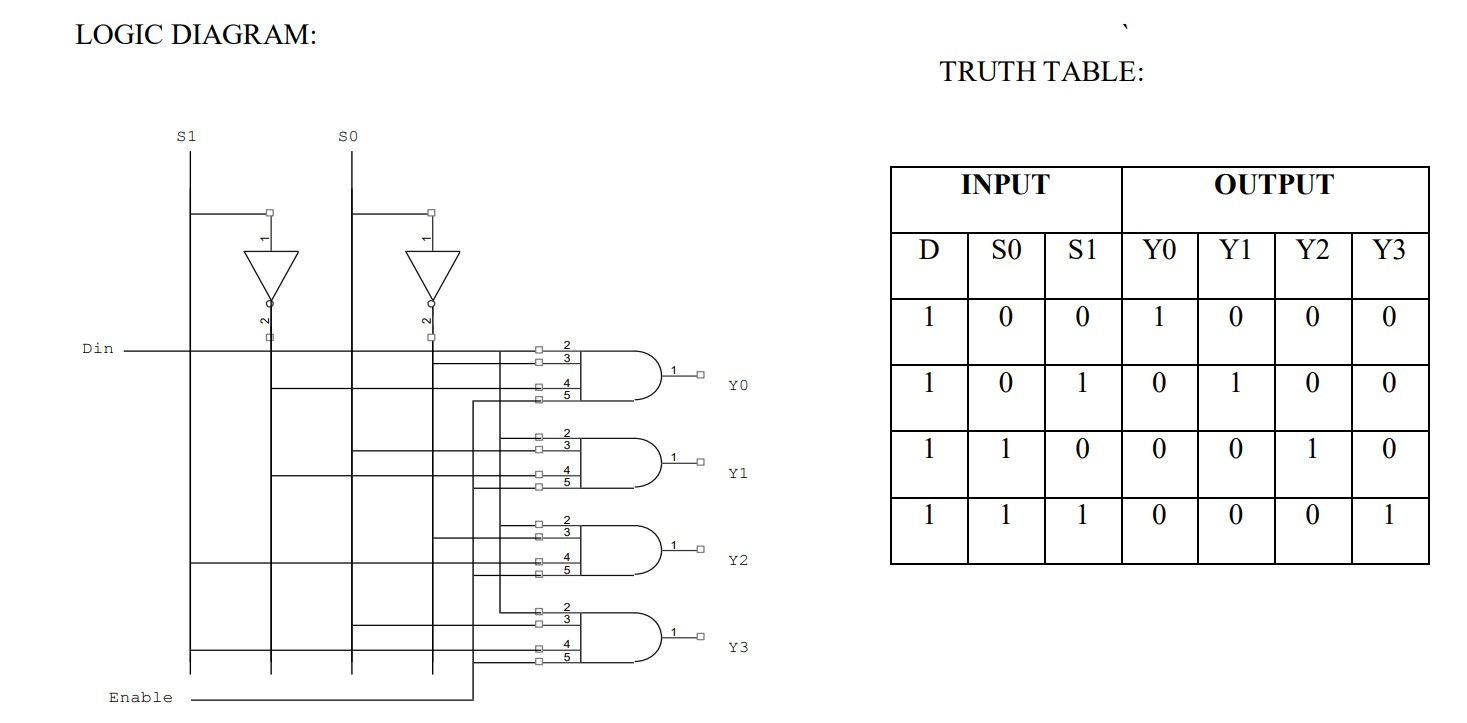
architecture Behavioral of mux\_behv is

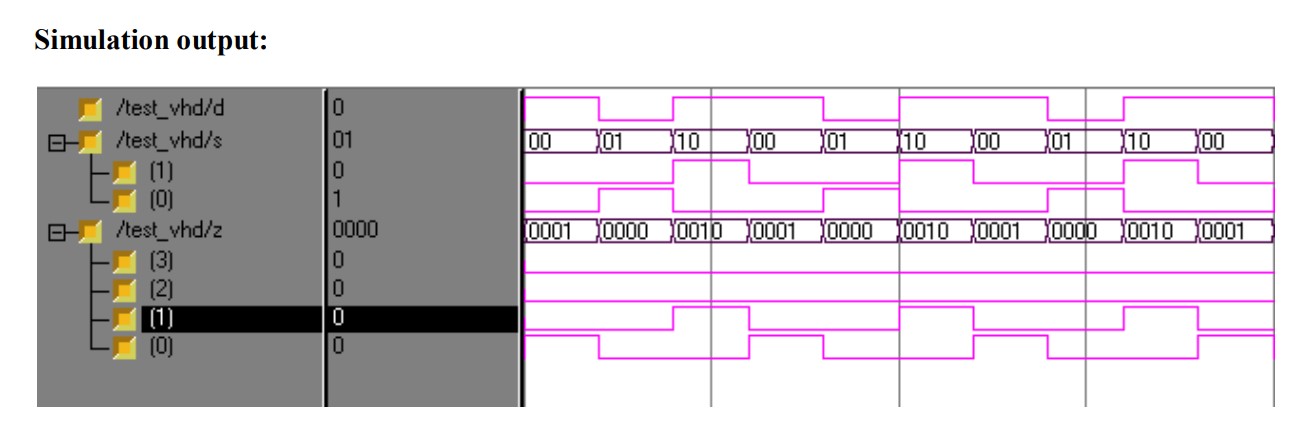
begin p1:process(d,s) begin if (s(0)<='0' and s(1)<='0') then y<=d(0); elsif (s(0)<='0' and s(1)<='1') then y<=d(1);

elsif (s(0)<='1' and s(1)<='0') then y<=d(2); else y<=d(3); end if; end process p1; end Behavioral;

RESULT: Thus the OUTPUT’s of Multiplexer was verified by synthesizing and simulating the VHDL code.

# Experiment 9 DeMultiplexer Logic Implementation and simulation in FPGA





# Experiment 9 DeMultiplexer Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for DeMultiplexer by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Dataflow Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity demux\_dataflow is Port ( d : in std\_logic; s : in std\_logic\_vector(1 downto 0); z : out std\_logic\_vector(3 downto 0));

end demux\_dataflow;

architecture dataflow of demux\_dataflow is signal s0bar,s1bar:std\_logic; begin s0bar<= not s(0); s1bar<= not s(1); z(0)<=d and s0bar and s1bar; z(1)<=d and s0bar and s(1); z(2)<=d and s(0) and s1bar; z(3)<=d and s(0) and s(1); end dataflow;

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity demux\_behv is

Port ( d : in std\_logic; s : in std\_logic\_vector(1 downto 0); z : out std\_logic\_vector(3 downto 0));

end demux\_behv;

architecture Behavioral of demux\_behv is begin p1:process(d,s)

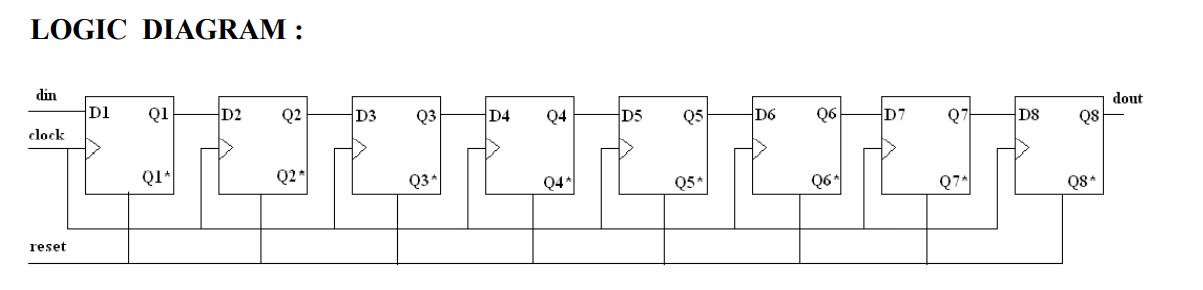
begin if (s(0)<='0' and s(1)<='0') then z(0)<=d; z(1)<='Z'; z(2)<='Z'; z(3)<='Z';

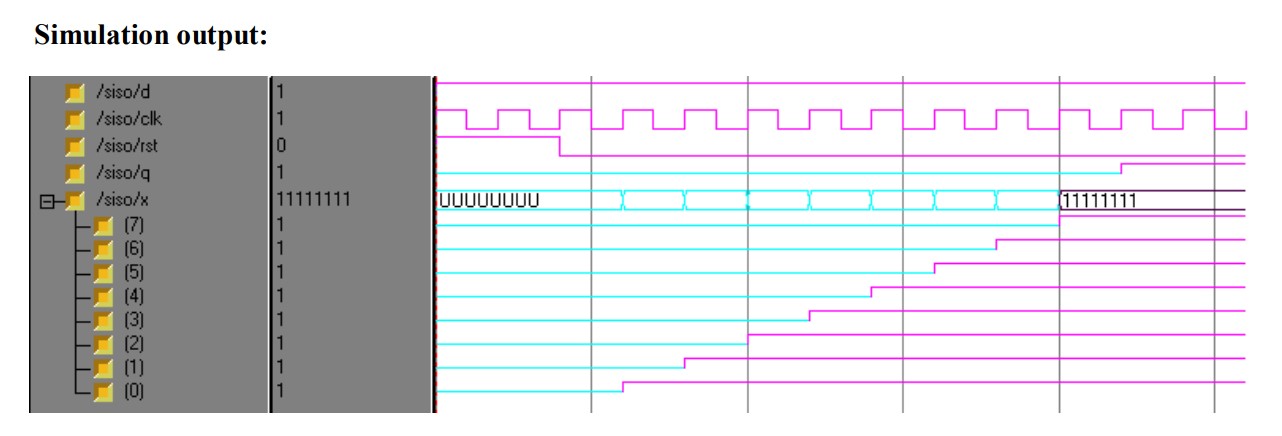
elsif (s(0)<='0' and s(1)<='1') then z(0)<='Z'; z(1)<=d; z(2)<='Z'; z(3)<='Z';

elsif (s(0)<='1' and s(1)<='0') then z(0)<='Z'; z(1)<='Z'; z(2)<=d; z(3)<='Z'; else z(0)<='Z'; z(1)<='Z'; z(2)<='Z'; z(3)<=d; end if; end process p1; end Behavioral;

RESULT: Thus the OUTPUT’s of Demultiplexers was verified by synthesizing and simulating the VHDL code.

# Experiment 10 Shift register Serial In Serial Out Logic Implementation and simulation in FPGA





# Experiment 10 Shift register Serial In Serial Out Logic Implementation and simulation in FPGA

**Aim:**

To develop the source code for Shift register SISO by using VHDL/VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

**Tools Required:**

1. Desktop computer 2. FPGA ISE Design Suite

**Algorithm:**

Step1: Specifications defined and design was initialized.

Step2: Name of the entity and architecture was declared by using VHDL source code.

Step3: Syntax was checked , debugged the errors and obtained the synthesis report. Step4: Output was verified by simulating the source code.

**Behavioral Modeling:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL; entity siso is

Port ( d : in std\_logic; clk : in std\_logic; rst : in std\_logic; q : out std\_logic);

end siso; architecture Behavioral of siso is signal x:std\_logic\_vector(7 downto 0); begin

process(d,clk,rst) begin

if (rst='1') then q<='X';

elsif (clk='1' and clk'event) then

x(0)<=d; x(1)<=x(0); x(2)<=x(1); x(3)<=x(2); x(4)<=x(3); x(5)<=x(4); x(6)<=x(5); x(7)<=x(6); q<=x(7); end if;

end process; end Behavioral;

RESULT: Thus the OUTPUT’s of 8-bit shift register was verified by synthesizing and simulating the VHDL code.