REVA UNIVERSITY

BENGALURU, INDIA

DIGITAL LOGIC DESIGN LAB

B22EH0307

3rd Sem B.Tech 2023

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

Name	
Srn	
Branch	
Semester	
Section	
Academic Year	

CONTENTS

Experiment No 1a.

Realization of basic gates using Universal gates

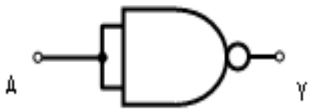
Aim: To verify the behavior of the basic gates which are realized using Universal gates.

Components/Apparatus required:

1) NAND gate IC 7400 -01 No.
2) NOR gate IC 7402-01 No.
3) Digital trainer kit.
4) Patch cards.

CIRCUIT DIAGRAM:

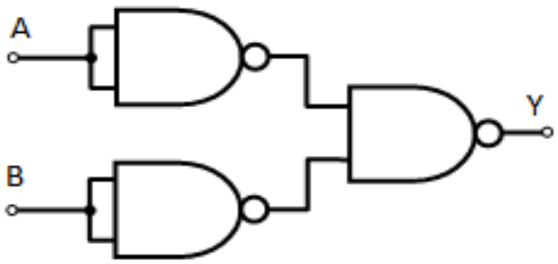
A) NOT using NAND



Truth Table

INPUT A	OUTPUT Y
0	1
1	0

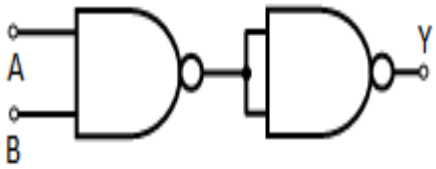
B) **OR using NAND**



Truth Table

INPUT A	INPUT B	OUTPUT Y
0	0	0
0	1	1
1	0	1
1	1	1

C) **AND using NAND**



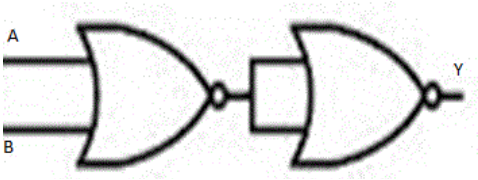
Truth Table

INPUT A	INPUT B	OUTPUT Y
0	0	0
0	1	0
1	0	0
1	1	1

D) **NOT using NOR** Truth Table

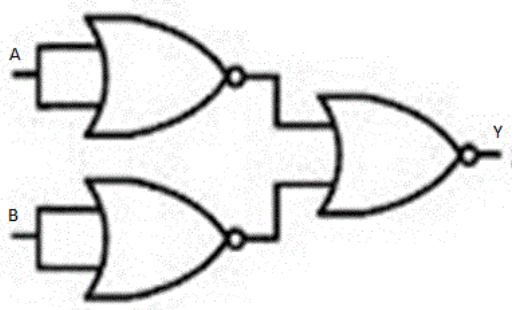
INPUT A	OUTPUT Y
0	1
1	0

E) **OR using NOR** Truth Table



INPUT A	INPUT B	OUTPUT Y
0	0	0
0	1	1
1	0	1
1	1	1

F) **AND using NOR** Truth Table



INPUT A	INPUT B	OUTPUT Y
0	0	0
0	1	0
1	0	0
1	1	1

Procedure:

1) Make connections as shown in the circuit diagram.
2) Connect input to +5v for logic 1 and to ground for logic 0
3) Verify the truth table.

Experiment No 1b.

Design and develop VHDL code to realize basic gates using Universal gates

Aim: To verify the behavior of basic gate function using universal gates through simulation.

Steps in Using the Xilinx software for writing VHDL Code:

1. Double click on "**Xlinx ISE 9.1i**" icon on the desktop.
2. Click on File- New project.
3. Give an name (preferably same name as the experiment name) and click on next.
4. In the new project wizard select Family as "**Automotive Spartan3**" and simulator "**ISE simulator (VHDL/Verilog)**" and prefer language"**VHDL**". Click on next.
5. Click on "**New source**" and next.
6. Enter the same file name as given in step 3 and select "**VHDL module**" click on next.
7. Enter the port name, direction and bus information given in the VHDL program under entity. Click on next.
8. After ensuring the port definition clicks on finish if not go back to re- enter the values for step 7 and then click on next.
9. Click on finish to enter the editor window and enter the VHDL code.
10. Save the file and check for syntax (by expanding codes "**synthesize-XST**")
11. Double click on "**Create new source"** and enter the file name. Select "**Test Bench waveform**" and click on next - next – finish.
12. Select single clock/ combinational under "**clock information**" and click on finish.
13. Select the inputs as high and low by clicking the mouse on the requiredinputs and save.
14. Select "**Behavioral simulation**" under sources and select **".tbw file**"
15. Click on process and expand "**Xlinx ISE simulator**" and double click on simulate behavioral module to observe the output waveforms.
16. Use cursor to verify truth table / result.

Note: The above steps have to be used for all VHDL simulations.

VHDL code for NOT gate Using NAND gates:

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

entity notgate is
        port(  A: in std_logic;
               Y: out std_logic
                );
end notgate;

architecture Behavioral of notgate is

begin
                Y<=  A nand A;
```

```
End Behavioral;
```

VHDL code for ORgate Using NAND gates:

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

entity orgate is
        port(   A: in std_logic;
                B: in std_logic;
                Y: out std_logic
                );
end orgate;

architecture Behavioral of orgate is

begin
                Y<=  (A nand A) nand (B nand B);

End Behavioral;
```

VHDL code for AND gate Using NAND gates:

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

entity andgate is
        port(   A: in std_logic;
                B: in std_logic;
                Y: out std_logic
                );
end andgate;

architecture Behavioral of andgate is

begin
                Y<=  (A nand B) nand (A nand B);

End Behavioral;
```

VHDL code for NOT gate Using NOR gates:

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

entity notgate is
        port(  A: in std_logic;
                Y: out std_logic
                );
end notgate;

architecture Behavioral of notgate is

begin
                Y<=  A nor A;

End Behavioral;
```

VHDL code for AND gate Using NAND gates:

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

entity orgate is
        port(    A: in std_logic;
                B: in std_logic;
                Y: out std_logic
                );
end orgate;

architecture Behavioral of orgate is

begin
                Y<=  (A nor A) nor (B nor B);

End Behavioral;
```

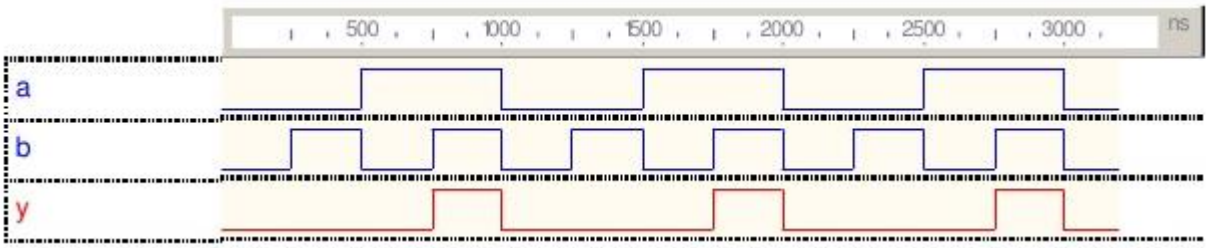
VHDL code for OR gate Using NAND gates:

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

entity andgate is
        port(    A: in std_logic;
                B: in std_logic;
                Y: out std_logic
                );
```

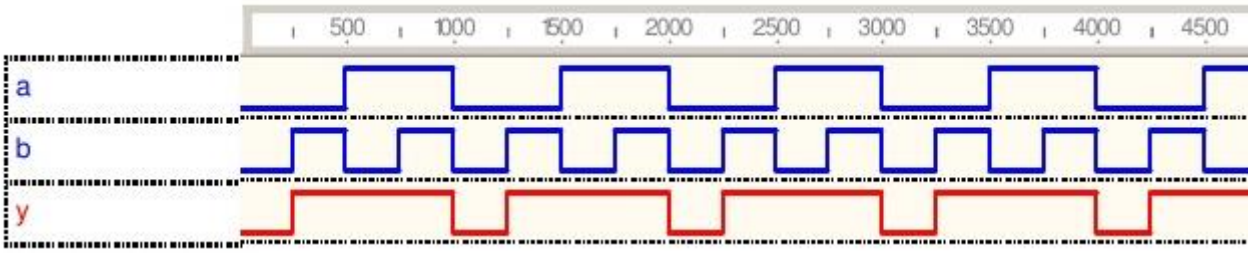
```
end andgate;

architecture Behavioral of andgate is
begin
                Y<=  (A nor B) nor (A nor B);
End Behavioral;
```

Output waveforms:

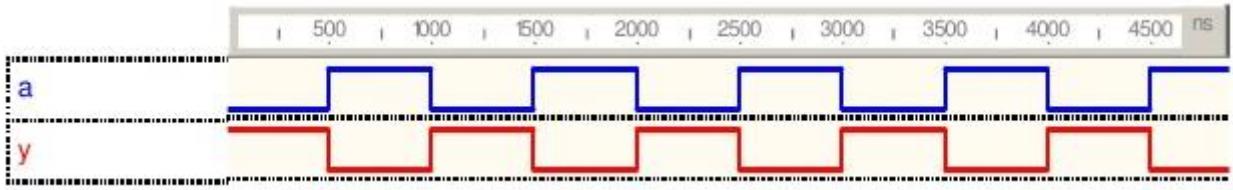
AND Gate:



OR Gate:



NOT Gate:



OBSERVATIONS

1.) What are the errors encountered in conducting the experiment/compilation?

2.) Measures taken to fix the problems identified in step 1 (circuit debug/logical errors)

3.) Results

ASSIGNMENT

1) Realize the EX-OR function using NAND gates only and verify the behavior.
a. Components Required: b. Circuit Diagram: c. Truth Table:

2) Realize the EX-OR function using NOR gates only and verify the behavior.
a. Components Required: b. Circuit Diagram: c. Truth Table:

VIVA QUESTIONS

1.) List all the basic gates and universal gates.

2.) Draw the truth table of 3 input EX-OR gate.

3.) Realize the function Y= AB+CD using basic gates.

4.) Simplify the expression Y = (A+A') +BC and realize the expression.

5.) Draw the timing diagram of a 2 input NOR gate for all input combination.

6.) Which of the two input gate can be used to implement an inverter circuit?

7.) State De- Morgan's Theorem

8.) What are primary uses of using Boolean algebra to simplify logic gates?

9.) What are the various methods of solving Boolean function?

10.) What is combinational circuit?

11.) What is sequential circuit?

Experiment No 2a.

Realization of Half adder and Full Adder using logic gates

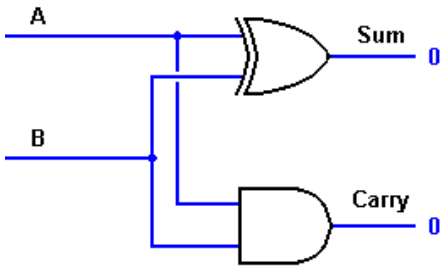
Aim: To build a half adder and Full adder using basic gates and verify its truth table.

Components/Apparatus required:

1) AND gates -01 Nos.
2) OR gates -01 Nos.
3) Digital trainer kit.
4) Patch cards.

1) Half adder

Circuit diagram:

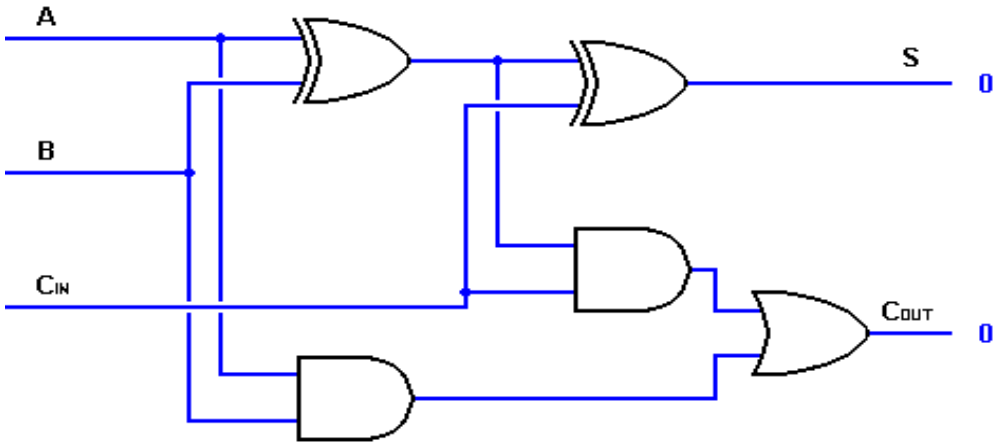


Truth Table

INPUT A	INPUT B	OUTPUT Sum	OUTPUT Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2) Full adder

Circuitdiagram:



Truth Table

INPUT A	INPUT B	INPUT C_{IN}	OUTPUT S	OUTPUT C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

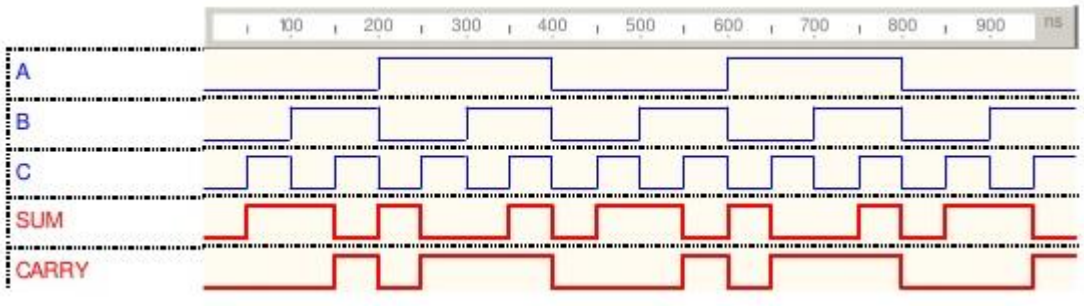
Experiment No 2b.

Design and develop VHDL code to realize Full adder and Full Subtractors

1) VHDL code for Full adder:

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entityfull_adder  is
     Port (   a : in std_logic;
              b : in std_logic;
              c : in std_logic;
              sum : out std_logic;
              carry : out std_logic);
end full_adder;

architecture behavioural of full_adder  is

begin
sum<= a xor b xor c;
carry<=((a xor b )and c) or (a and b) ;

end behavioural;
```

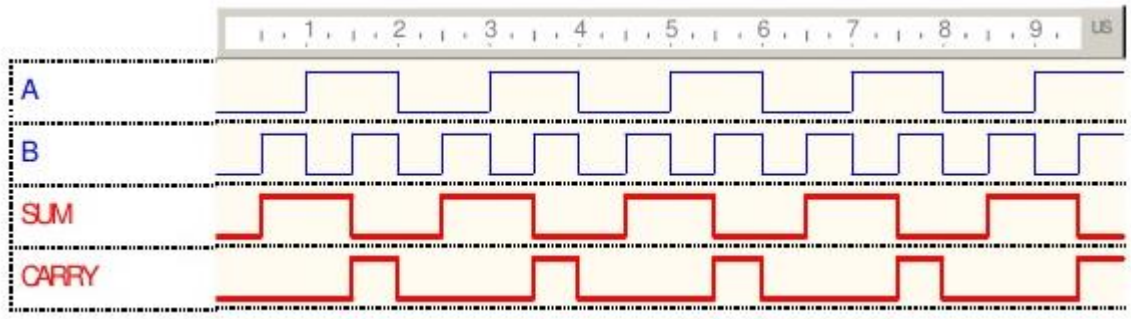
Output waveform:



2) VHDL code for Half Adder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity half_adder  is
     Port (   a : in std_logic;
              b : in std_logic;
              sum : out std_logic;
              carry : out std_logic);
end half_adder;
architecture behavioural of half_adder  is
begin
sum<=a xor b;
carry<= a and b;

end behavioural;
```

Output Waveform:



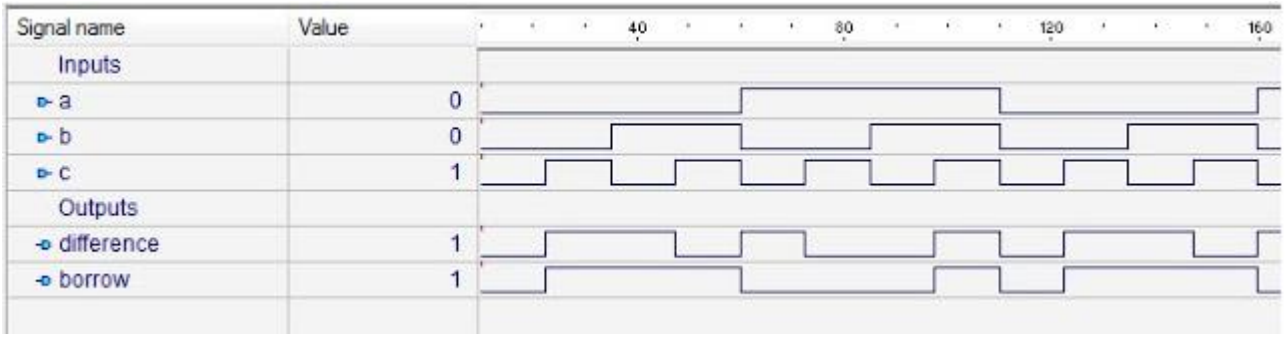
---

3)VHDL code for Full Subtractor:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_subtractor is
    port(
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       c : in STD_LOGIC;
```

```
        difference : out STD_LOGIC;
        borrow : out STD_LOGIC);
end full_subtractor;

architecture behavioural of full_subtractor is
begin

    difference <= a xor b xor c;
    borrow <= ((not a) and b) or ((not(a xor b)) and c));

end behavioural;
```

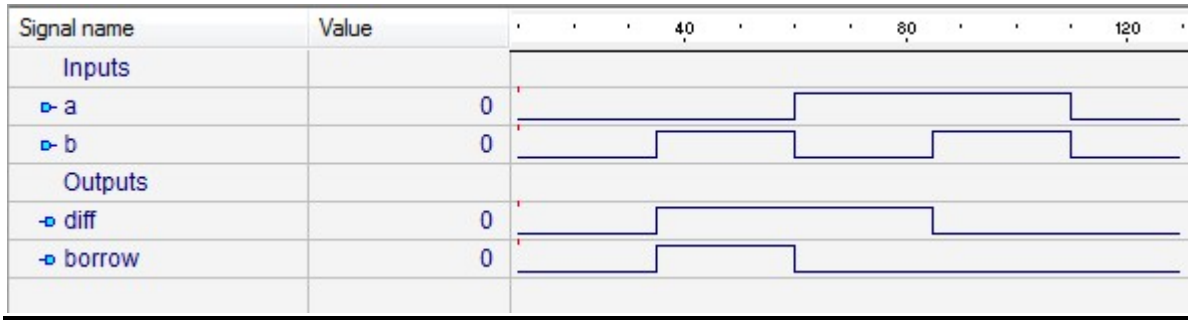
Output Waveform:



4)VHDL code for Half Subtractor:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity half_subtractor is
      port(   a : in STD_LOGIC;
              b : in STD_LOGIC;
              diff : out STD_LOGIC;
              borrow : out STD_LOGIC);
end half_subtractor;

architecture behavioural of half_subtractor is
begin

    diff <= a xor b;
    borrow <= (not a) and b;

end behavioural;
```

Output Waveform:



OBSERVATIONS

1.) What are the errors encountered in conducting the experiment/compilation?

2.) Measures taken to fix the problem (circuit debug/logical errors)?

3.) Results

ASSIGNMENT

1.) Realize the full adder using 3:8 decoder and verify the behavior.

a. Components Required: b. Circuit Diagram: c. Truth Table:

VIVA QUESTIONS

1.) The full adder realized in this experiment is

a.) 4-bit adder b) 2-bit adder c) 3-bit adder d) 1-bit adder

2.) Draw the block diagram for 3-bit parallel adder.

3.) Write the 2's compliment of the following numbers

a.) 00001111

b.) 01011010

c.) 10111110

4.) Subtract 64 from 89 using 2's compliment addition.

5.) What is parallel adder?

6.) What is the difference between adder & parallel adder?

7.) How many full adders are required to construct m bit parallel adder?

8.) Write the Boolean expression for Sum and Carry for full adder.

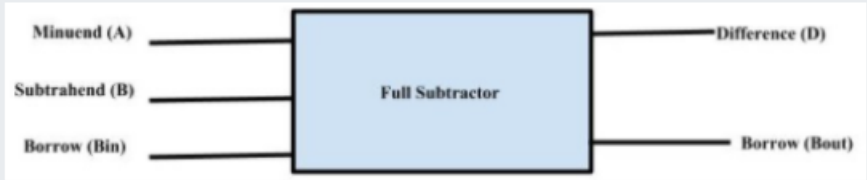
9.) Write the Boolean expression for Difference and Borrow for full subtractor.

10.) Write the difference between combinational & sequential circuits.

Experiment No 3a.

Realization of Full Subtractor using 3:8 decoders IC 74138.

Full subtractor contains 3 inputs and 2 outputs (Difference and Borrow) as shown-



The truth table for full subtractor:

Inputs			Outputs	
A	B	B_{in}	B_{out} (Borrow)	D (Difference)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The truth table for full subtractor:

From the above truth table,

For the different functions in the truth table, the minterms can be written as 1,2,4,7, and similarly, for the borrow, the minterms can be written as 1,2,3,7.

Since there are three inputs and a total of eight minterms. So we need 3-to-8 line decoder. The decoder generates the eight minterms for A, B & B_{in}.

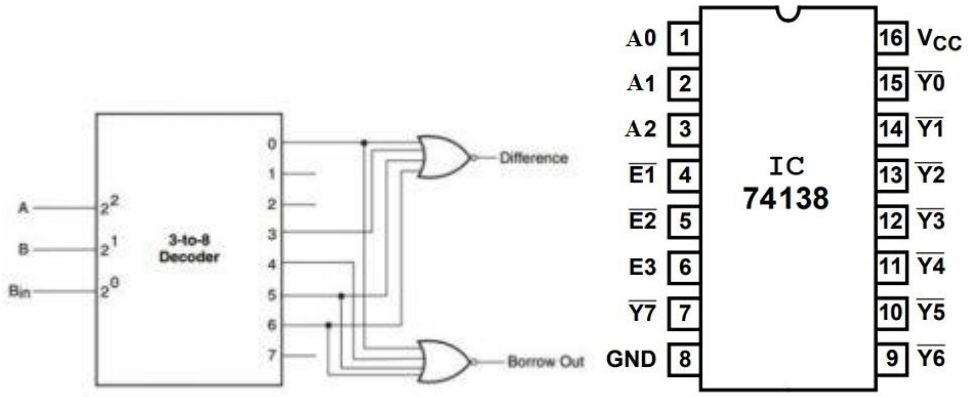
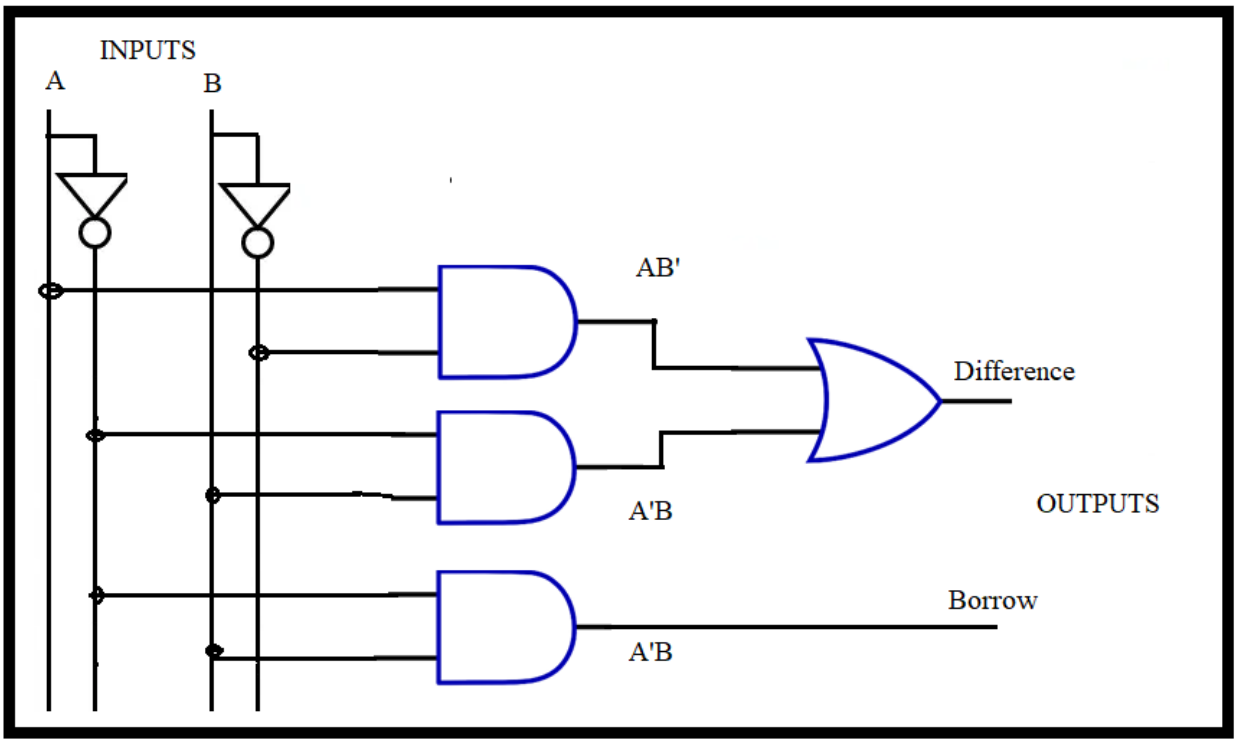


Fig: Full subtractor with 3-to-8 Decoder and NOR gates

Experiment No 3b.

Realization of Half Subtractor using basic gates.



Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Experiment No 4a.

Given a 4-variable logic expression, simplify it using Entered Variable Map and realize the simplified logic expression using 8:1 multiplexer IC.

Aim: To realize the Boolean Expression Y= Σm(1,2,3,6,8,9,10,12,13,14) using 8:1 Multiplexer IC

Components/Apparatus required:

1) Multiplexer IC74151.
2) Digital IC Trainer Kit.
3) Patch Chords.

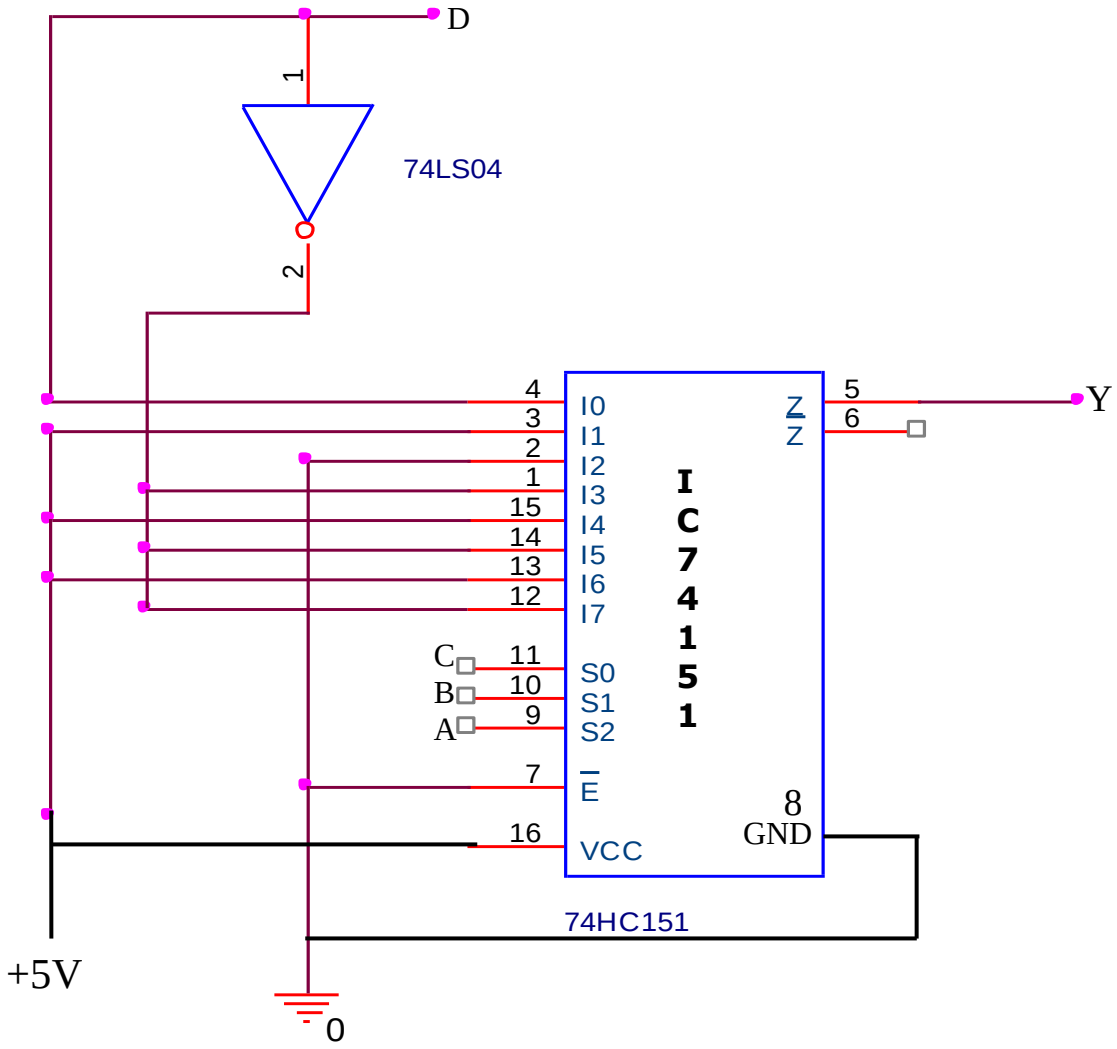
TRUTH TABLE:

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

IMPLEMENTATION TABLE:

A B C	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
D = 0	Y = 0	Y = 1	Y = 0	Y = 1	Y = 1	Y = 1	Y = 1	Y = 1
D = 1	Y = 1	Y = 1	Y = 0	Y = 0	Y = 1	Y = 0	Y = 1	Y = 0
Mux I/P	$I_0 = D$	$I_1 = 1$	$I_2 = 0$	$I_3 = \overline{D}$	$I_4 = 1$	$I_5 = \overline{D}$	$I_6 = 1$	$I_7 = \overline{D}$

CIRCUIT DIAGRAM:



PROCEDURE:

1. Convert any given Boolean Expression into Σm( ) notation.
2. Write the Implementation table for the given expression.
3. Find the inputs to be applied to the inputs of the Multiplexer.

4. Make connections as per the implementation table.
5. Apply the inputs using Switches on the Trainer kit and verify the output LEDs on the Trainer kit.
6. Verify the truth table

Experiment No 4b.

Design and develop the VHDL code for an 8:1 multiplexer. Simulate and verify it's working.

Aim: Simulate verify the working of 8:1 multiplexer by writing VHDL code.

Steps in Using the Xilinx software for writing VHDL Code:

1. Double click on "**Xlinx ISE 9.1i**" icon on the desktop.
2. Click on File- New project.
3. Give a name (preferably same name as the experiment name) and click on next.
4. In the new project wizard select Family as "**Automotive Spartan3**" and simulator "**ISE simulator (VHDL/Verilog)**" and prefer language"**VHDL**". Click on next.
5. Click on "**New source**" and next.
6. Enter the same file name as given in step 3 and select "**VHDL module**" click on next.
7. Enter the port name, direction and bus information given in the VHDLprogram under Entity. Click on next.
8. After ensuring the port definition clicks on finish if not go back to re-enter the values for step 7 and then click on next.
9. Click on finish to enter the editor window and enter the VHDL code.
10. Save the file and check for syntax (by expanding codes "**synthesize-XST**")
11. Double click on "**Create new source"** and enter the file name. Select"**Test Bench Waveform**" and click on next - next – finish.
12. Select single clock/ combinational under "**clock information**" and clickon finish.
13. Select the inputs as high and low by clicking the mouse on the required inputs and save.
14. Select "**Behavioral simulation**" under sources and select **".tbw file**"
15. Click on process and expand "**Xlinx ISE simulator**" and double click on simulate behavioral module to observe the output waveforms.
16. Use cursor to verify truth table / result

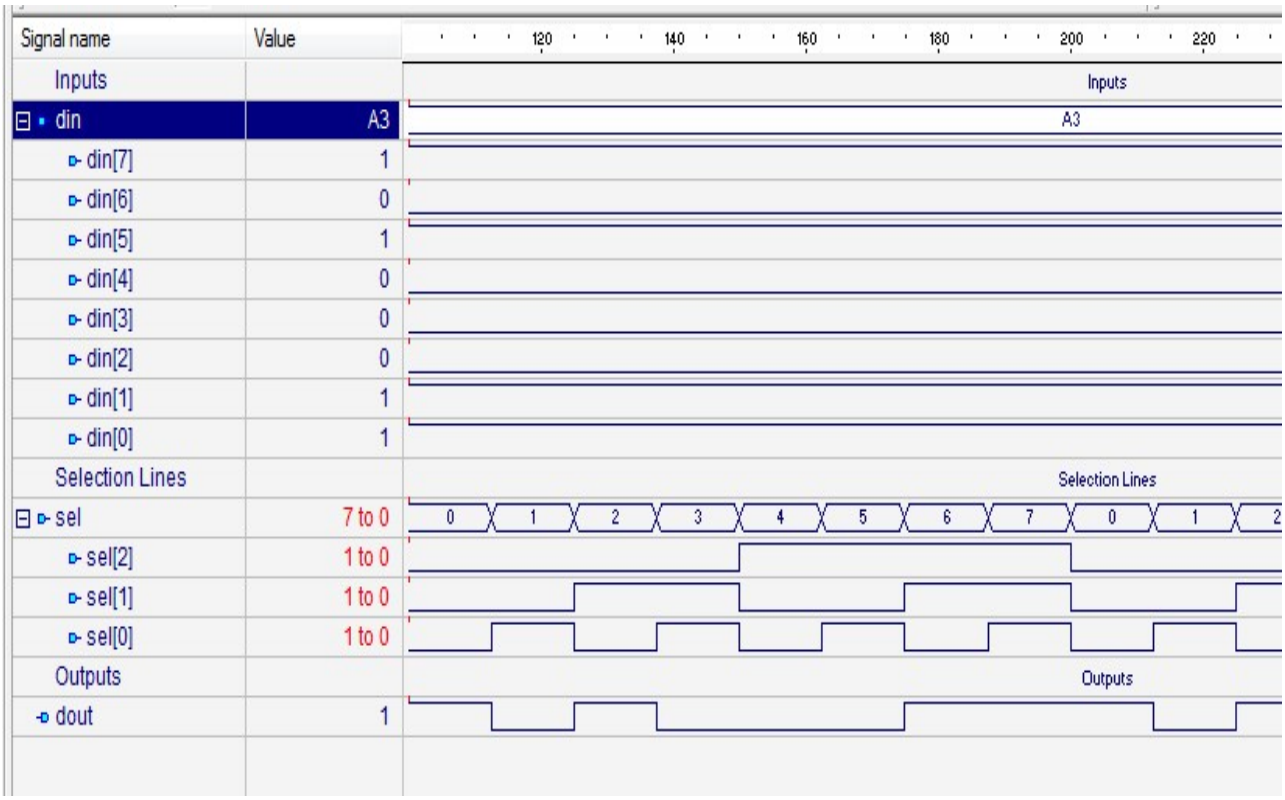
Note: The above steps have to be used for all VHDL simulations.

VHDL code for 8:1 MUX

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux8to1 is
                Port ( sel : in  STD_LOGIC_vector(2 downto 0);
                din : in  STD_LOGIC_vector(0 downto 7);
                dout : out  STD_LOGIC);
end mux8to1;

architecture Behavioral of mux8to1 is
```

```
begin
        process(sel,din(7), din(6), din(5), din(4), din(3), din(2), din(1), din(0))

begin
        case sel is

        when "000"=>dout<=din(7);
        when "001"=>dout <= din(6);
        when "010"=>dout <= din(5);
        when "011"=>dout <= din(4);
        when "100"=>dout <= din(3);
        when "101"=>dout <= din(2);
        when "110"=>dout <= din(1);
        when "111"=>dout <= din(0);
        when others=>null;

end case;

end process;

end Behavioral;
```

Output waveform:



OBSERVATIONS

1.) What are the errors encountered in conducting the experiment/compilation?

2.) Measures taken to fix the problem(circuit debug/logical errors)

3.) Results

ASSIGNMENT

1.) Realize the Boolean Expression Y= πM(1,2,3,6,8,9,10,12,13,14) using 8:1 Multiplexer IC

a. Components Required: b. Circuit Diagram: c. Truth Table:

VIVA QUESTIONS

1.) Draw the block diagram of a 4:1 multiplexer and a 2:1 multiplexer.

2.) Realize a 8:1 multiplexer using two 4:1 multiplexer and a 2:1 multiplexer.

3.) A circuit with many inputs and only one output is called a _____

4.) Write a VHDL code for 4:1 multiplexer.

5.) What is function of enable input in multiplexer chip?

6.) Why multiplexer called as data selector?

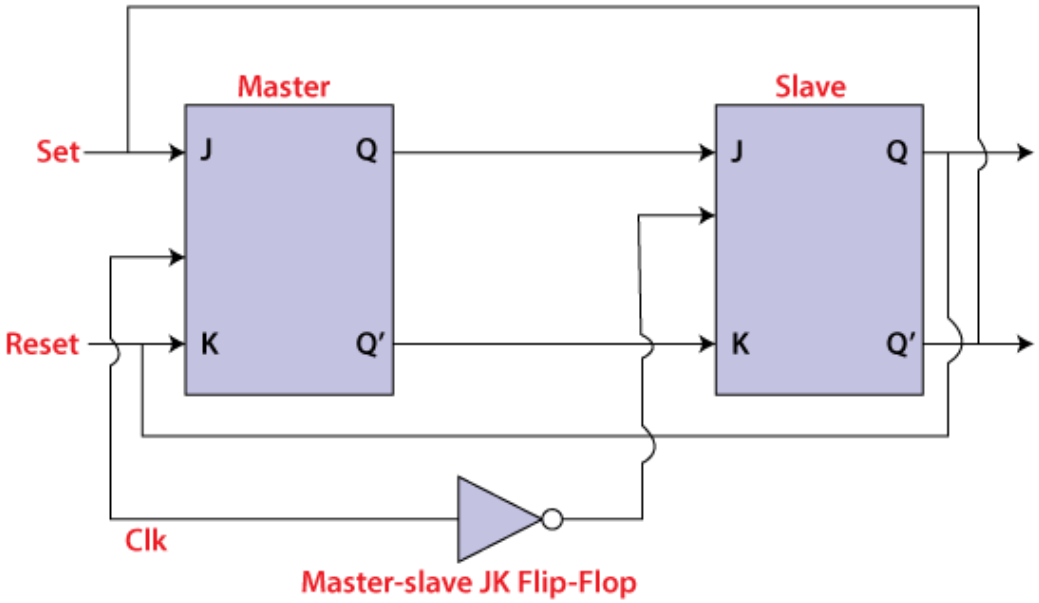
7.) Mention the applications of multiplexer.

8.) How many control inputs are there in 16:1 demultiplexer?

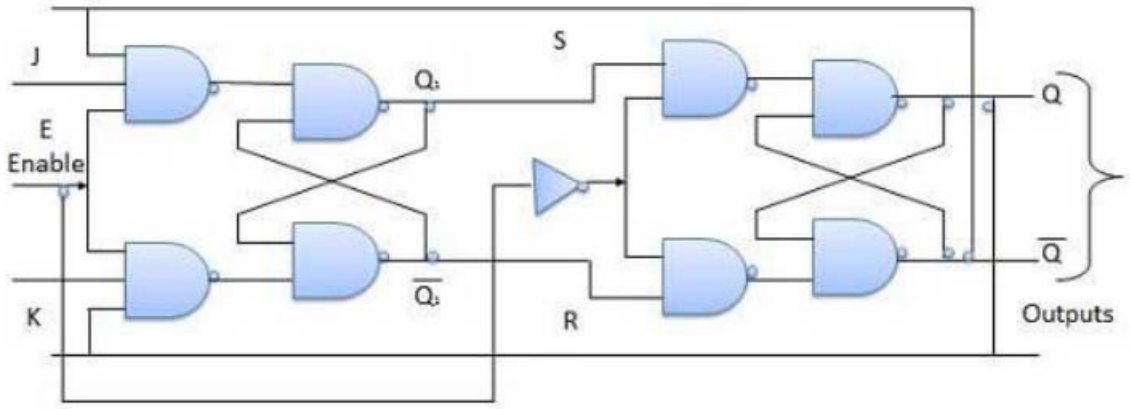
9.) How many select lines will a 32:1 multiplexer will have?

Experiment No. 5a

Realization of Master Slave JK flip flop truth table using IC 7476 and verify Delay and toggle flip flop using MSJK flip flop IC.

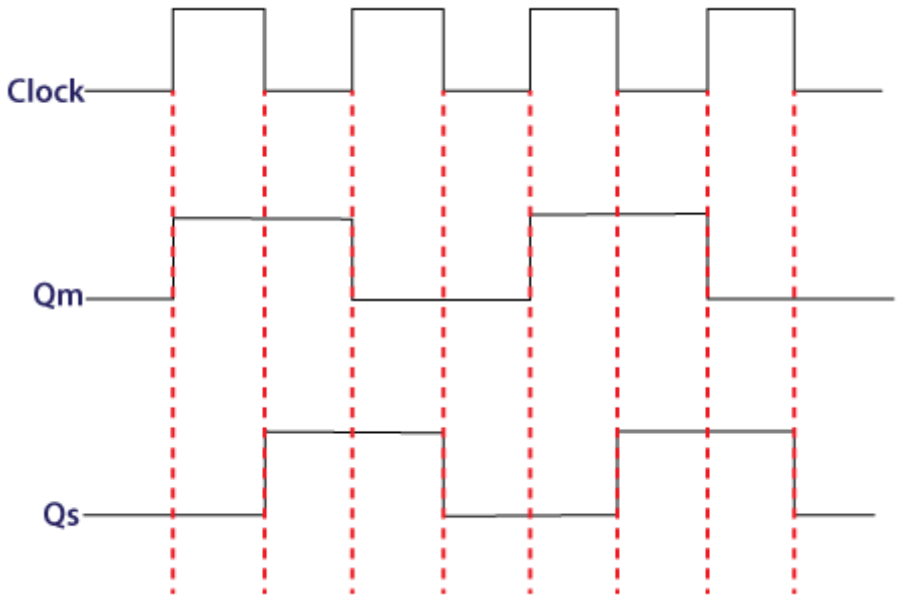


Circuit Diagram



Truth Table

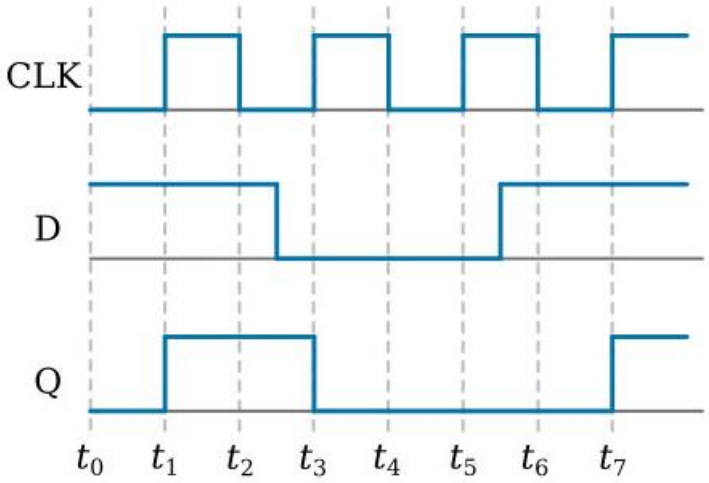
Inputs			Outputs		Comments
E	J	K	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	$\overline{Q}_n$	Q_n	Toggle



5b. Design and develop VHDL code for D Flip-Flop with positive-edge triggering. Simulate and verify it's working.

VHDL Code for D Flip Flop

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity D_FF is
PORT( D,CLOCK: in std_logic;
Q: out std_logic);
end D_FF;

architecture behavioral of D_FF is
begin
process(CLOCK)
begin
if(CLOCK='1' and CLOCK'EVENT) then
Q <= D;
end if;
end process;
end behavioral;
```



Experiment No. 6a

Design and implement a mod-n (n<8) synchronous up counter using J-K Flip-Flop ICs and demonstrate its working.

Step 1:

Determine the number of flip flop needed

Flip flop required are

$2^n \geq N$

Mod 5 hence N=5

$\therefore 2^n \geq N$

$\therefore 2^n \geq 5$

N=3 i.e. 3 flip flop are required

Step 2:

Type of flip flop to be used: JK flip flop

Step 3:

1) Excitation table for JK flip flop

Qn	Qn+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

2) Excitation table for counter

Present state			Next state			Flip flop Input					
Q_c	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_a
0	0	0	0	0	1	×	0	0	×	1	×
0	0	1	0	1	0	×	1	1	×	×	1
0	1	0	0	1	1	×	×	×	0	1	×
0	1	1	1	0	0	×	×	×	1	×	1
1	0	0	0	0	0	1	0	0	×	0	×
1	0	1	×	×	×	×	×	×	×	×	×
1	1	0	×	×	×	×	×	×	×	×	×
1	1	1	×	×	×	×	×	×	×	×	×

Step 4

K-map simplification

For J_C

Q_BQ_A / Q_C	00	01	11	10
0	0	0	1	0
1	×	×	×	×

$J_C=Q_BQ_A$

For K_C

Q_BQ_A / Q_C	00	01	11	10
0	×	×	×	×
1	1	×	×	×

$K_C=1$

For J_B

Q_BQ_A / Q_C	00	01	11	10
0	1	×	×	×
1	×	×	×	×

$J_B=QA$

For K_B

Q_BQ_A / Q_C	00	01	11	10
0	×	×	1	0
1	×	×	×	×

$K_B=QA$

For J_A

Q_BQ_A / Q_C	00	01	11	10
0	1	×	×	1
1	0	×	×	×

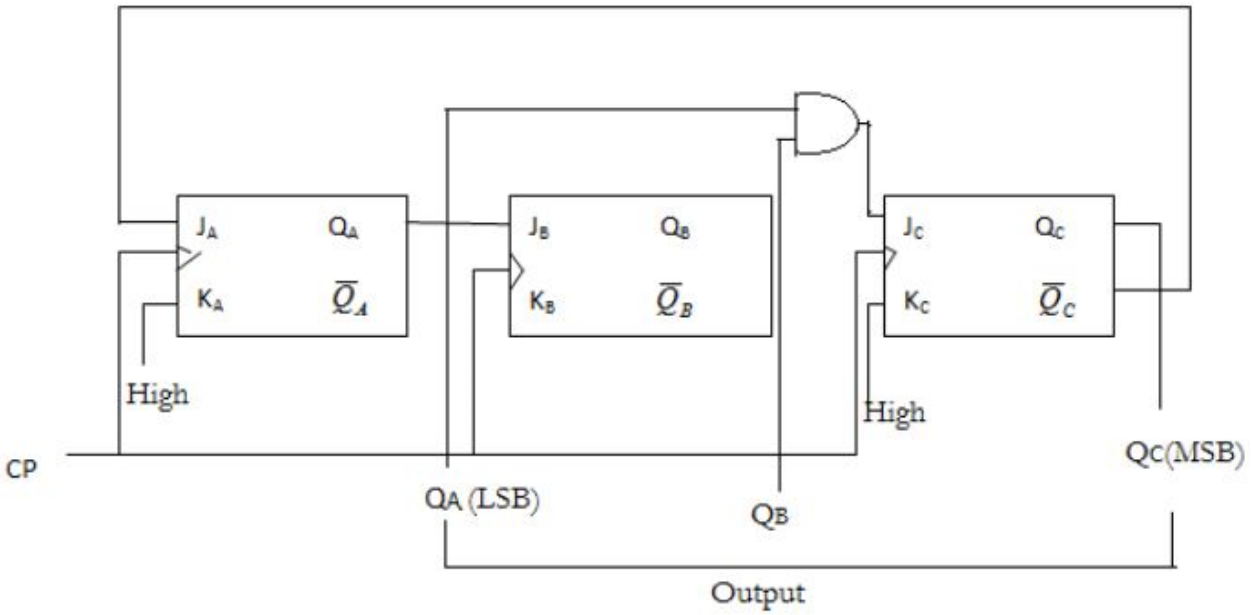
$J_A=Qc'$

For K_A

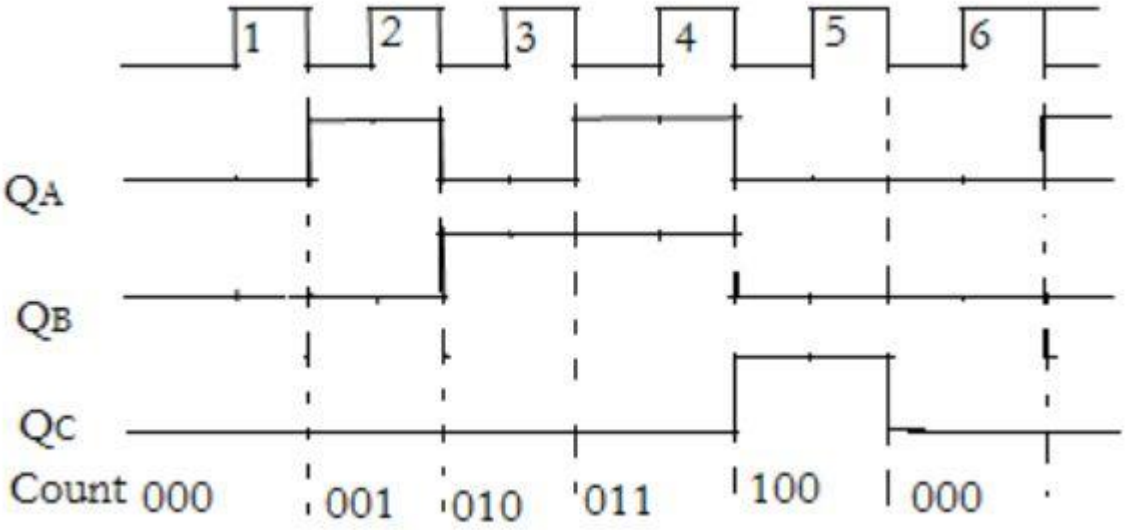
Q_BQ_A / Q_C	00	01	11	10
0	×	1	1	×
1	×	×	×	×

$K_A=1$

Step 5 Logic Diagram



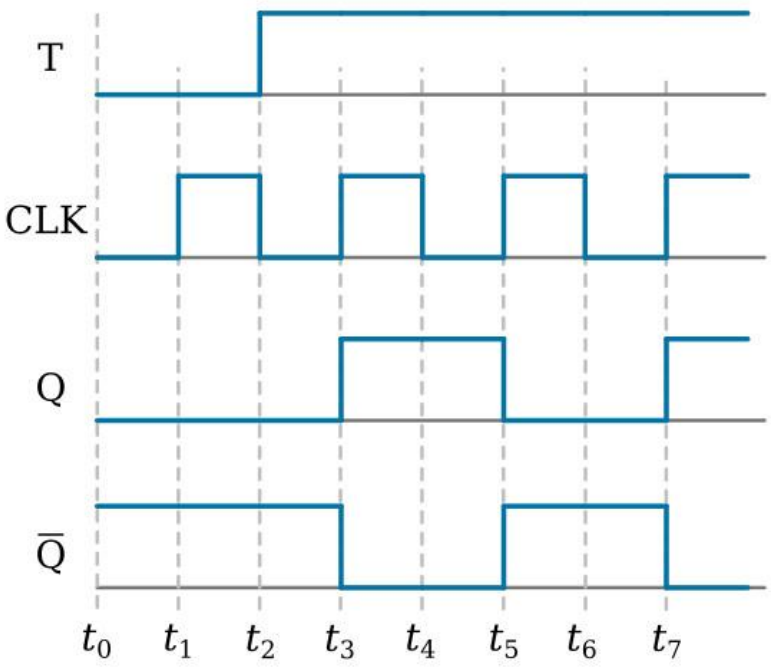
Step 6: Timing Diagram



Design and develop the Verilog / VHDL code for T Flip-Flop with positive-edge triggering. Simulate and verify it's working.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity T_FLIP_FLOP is
   Port ( T, CLK : in STD_LOGIC;
          Q      : out STD_LOGIC);
end T_FLIP_FLOP;

architecture Behavioral of T_FLIP_FLOP is
   signal Q_internal : STD_LOGIC := '0';
begin
   process (CLK)
   begin
      if rising_edge(CLK) then
         if T = '1' then
            Q_internal <= not Q_internal;
         end if;
      end if;
   end process;

   Q <= Q_internal;
end Behavioral;
```

OUT PUT



Experiment No. 7a

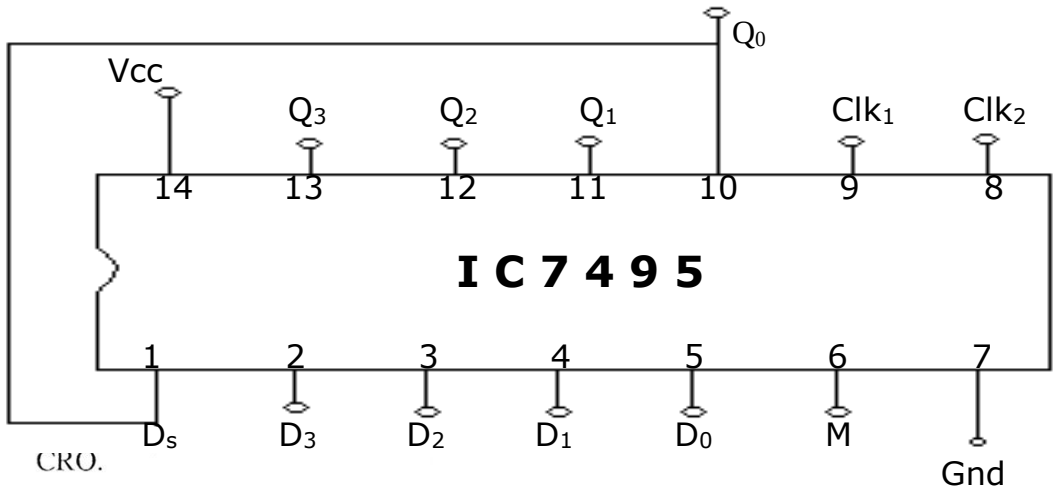
Implement a ring counter using 4-bit shift register and demonstrate its working .

Aim: To implement a Ring Counter using 4-bit Shift Register

Components/Apparatus required:

1) Shift Register IC 7495
2) Digital IC Trainer Kit
3) Patch Chords

CIRCUIT DIAGRAM:



Procedure:

1. Mode control is made 1.
2. Parallel inputs say 0001 are given to D_3 D_2 D_1 D_0 inputs of 7495 respectively
3. Clock 2 is pulsed once. Now D_3 D_2 D_1 D_0 parallel inputs appears on Q_3 Q_2 Q_1 Q_0 lines.
4. Clock 1 of 7495 is connected to the pulser.
5. Now mode control is made '0'.
6. When clock pulses are applied this '1' circulates around the circuit as shown.

Clock	Time	Outputs Q_3	Q_2	Q_1	Q_0
Clock 2	t_0 (Starting state)	0	0	0	1
Clock 1	t_1 (After 1st clock pulse)	1	0	0	0
	t_2 (After 2nd clock pulse)	0	1	0	0
	t_3 (After 3rd clock pulse)	0	0	1	0
	t_4 (After 4th clock pulse)	0	0	0	1

Experiment No. 7b

Design and develop VHDL code for Johnson counter. Simulate and verify it's working.

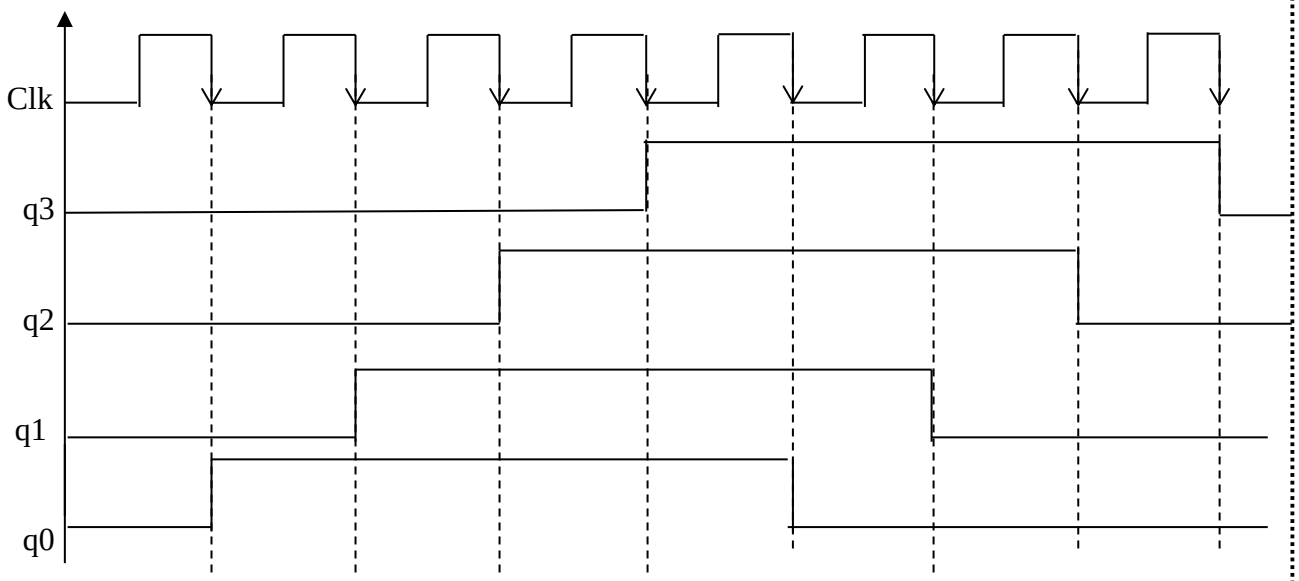
Design and develop VHDL code for Johnson counter. Simulate and verify it's working.

Aim: Simulate verify the working of switched tail counter by writing VHDLcode.

VHDL code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity johnson is
        Port ( clk : in  STD_LOGIC;
                q : inout  STD_LOGIC_VECTOR (3 to 0) :="0000");
end johnson;

architecture Behavioral of johnson is

begin
        process(clk)
begin
        if(clk'event and clk='0') then
        q(3)<= not(q(0));
        for i in 3downto 1 loop
        q(i-1)<= q(i);

        end loop;
        end if;
        end process;

end Behavioral;
```

WAVEFORM:



Experiment No 8a.

Design and implement an asynchronous counter using decade counter IC to count up from 0 to n (n<=9) and demonstrate its working

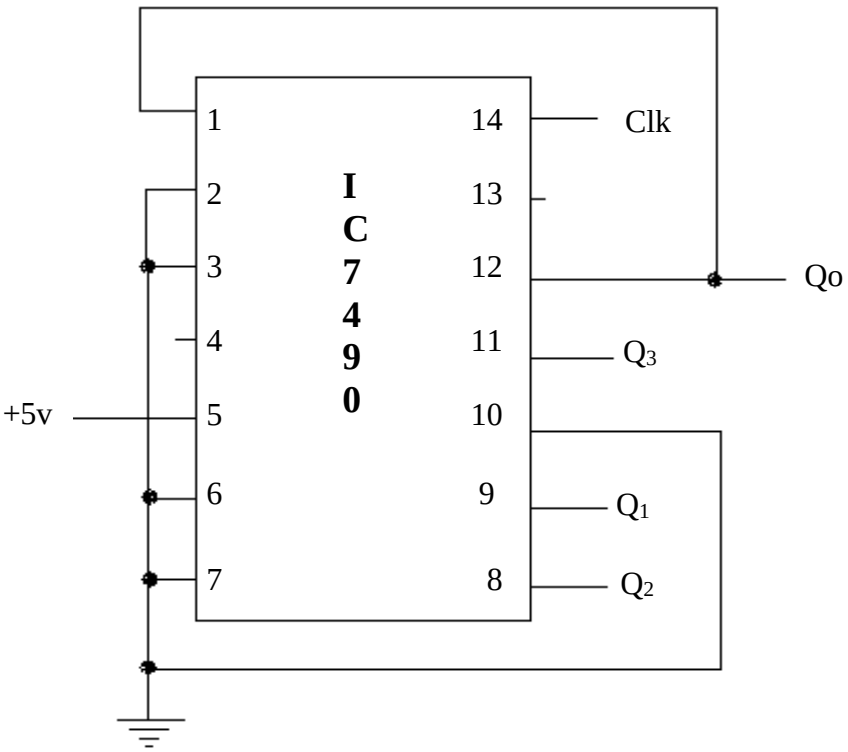
Aim: To design & implement Asynchronous counter using decade counter IC7490.

Components/Apparatus required:

1) Decade counter IC7490
2) Digital IC Trainer Kit
3) Patch Chords

CIRCUIT DIAGRAM:

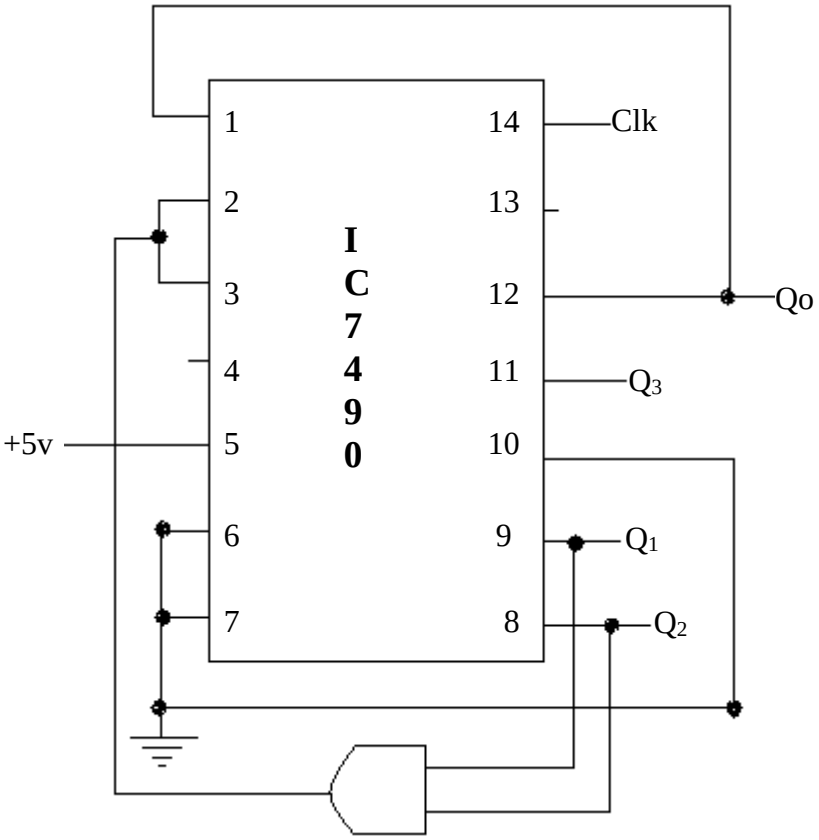
MOD-10 COUNTER:



TRUTH TABLE:

Clk	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

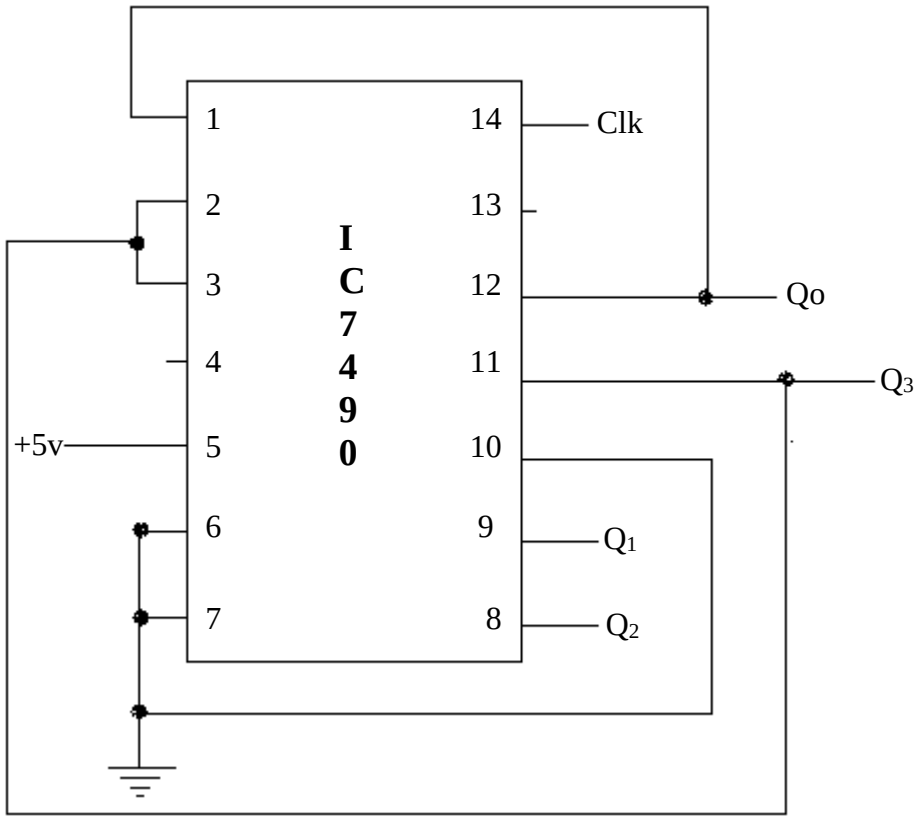
MOD- 6 COUNTER:



TRUTH TABLE:

Clk	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	0	0	0

MOD – 8 COUNTER:



TRUTH TABLE:

Clk	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	0	0	0	0

Procedure:

1. Make connection as shown in the connection diagram.
2. By applying clock pulses verify the count sequence for all the above 3 mod values.

Experiment No 8b.

Design and develop VHDL code for mod-8 up counter, Simulate and verify it's working.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Modulo8Counter is
   Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          count : out STD_LOGIC_VECTOR(2 downto 0));
end Modulo8Counter;

architecture Behavioral of Modulo8Counter is
   signal counter : STD_LOGIC_VECTOR(2 downto 0) := "000";
begin
   process(clk, reset)
   begin
      if reset = '1' then
         counter <= "000";  -- Reset the counter to 0
      elsif rising_edge(clk) then
         if counter = "111" then
            counter <= "000";  -- Wrap around when it reaches 7
         else
            counter <= counter + 1;  -- Increment the counter
         end if;
      end if;
   end process;

   count <= counter;
end Behavioral;
```

OUT PUT

Experiment No 9.

Design and develop VHDL code for 1-bit and 2-bit magnitude comparator, Simulate and verify it's working.

Experiment No 10.

Design and develop VHDL code for 8 to 3 Encoder and 1:4 De-multiplexer, Simulate and verify it's working

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity EIGHT_TO_THREE_ENCODER is
   Port ( I0, I1, I2, I3, I4, I5, I6, I7 : in STD_LOGIC;
          Y0, Y1, Y2                     : out STD_LOGIC);
end EIGHT_TO_THREE_ENCODER;

architecture Behavioral of EIGHT_TO_THREE_ENCODER is
begin
   Y0 <= '1' when (I0 = '1') else '0';
   Y1 <= '1' when (I1 = '1' or I2 = '1' or I3 = '1') else '0';
   Y2 <= '1' when (I4 = '1' or I5 = '1' or I6 = '1' or I7 = '1') else '0';
end Behavioral;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ONE_TO_FOUR_DEMUX is
   Port ( D, S0, S1 : in STD_LOGIC;
          Y0, Y1, Y2, Y3 : out STD_LOGIC);
end ONE_TO_FOUR_DEMUX;

architecture Behavioral of ONE_TO_FOUR_DEMUX is
begin
   Y0 <= D when (S0 = '0' and S1 = '0') else '0';
   Y1 <= D when (S0 = '1' and S1 = '0') else '0';
   Y2 <= D when (S0 = '0' and S1 = '1') else '0';
   Y3 <= D when (S0 = '1' and S1 = '1') else '0';
end Behavioral;
```