MULTIPLEXER AND DEMULTIPLEXER

AIM:

To design and implement the multiplexer and demultiplexer using logic gates and study of IC 74150 and IC 74154.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	3 I/P AND GATE	IC 7411	2
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	32

THEORY:

MULTIPLEXER:

Multiplexer means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally there are 2^n input line and n selection lines whose bit combination determine which input is selected.

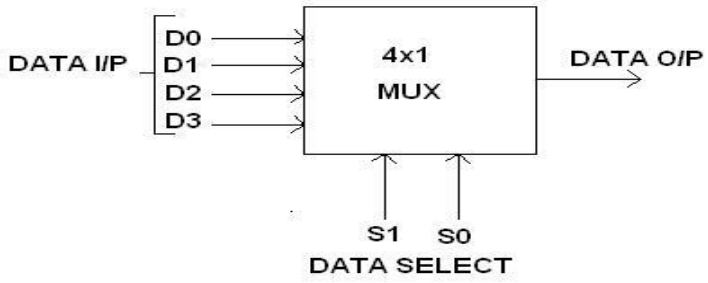
DEMULTIPLEXER:

The function of Demultiplexer is in contrast to multiplexer function. It takes information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. Decoder can also be used as demultiplexer.

In the 1: 4 demultiplexer circuit, the data input line goes to all of the AND gates. The data select lines enable only one gate at a time and the data on the data input line will pass through the selected gate to the associated data output line.

4:1 MULTIPLEXER

BLOCK DIAGRAM FOR 4:1 MULTIPLEXER:



FUNCTION TABLE:

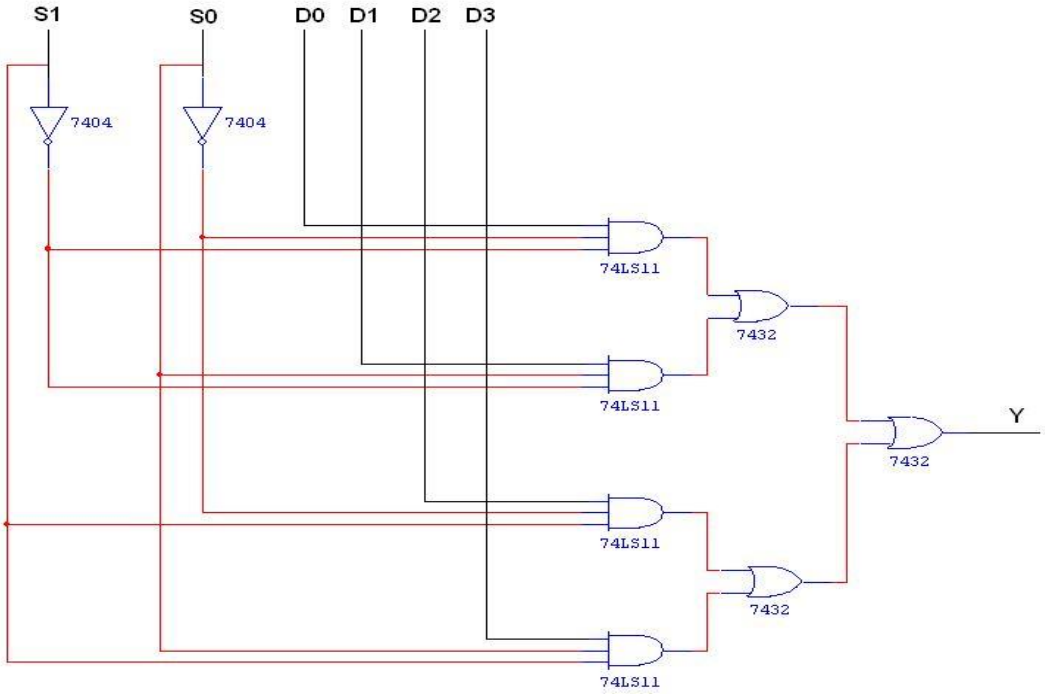
S1	S0	INPUTS Y
0	0	D0 → D0 S1' S0'
0	1	D1 → D1 S1' S0
1	0	D2 → D2 S1 S0'
1	1	D3 → D3 S1 S0

Y = D0 S1' S0' + D1 S1' S0 + D2 S1 S0' + D3 S1 S0

TRUTH TABLE:

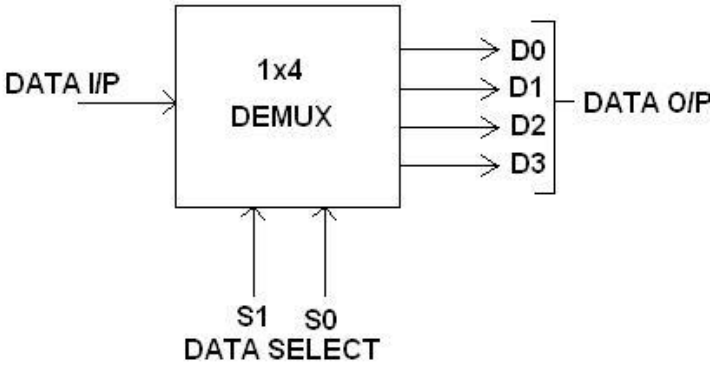
S1	S0	Y = OUTPUT
0	0	D0
0	1	D1
1	0	D2
1	1	D3

CIRCUIT DIAGRAM FOR MULTIPLEXER:



1:4 DEMULTIPLEXER

BLOCK DIAGRAM FOR 1:4 DEMULTIPLEXER:



FUNCTION TABLE:

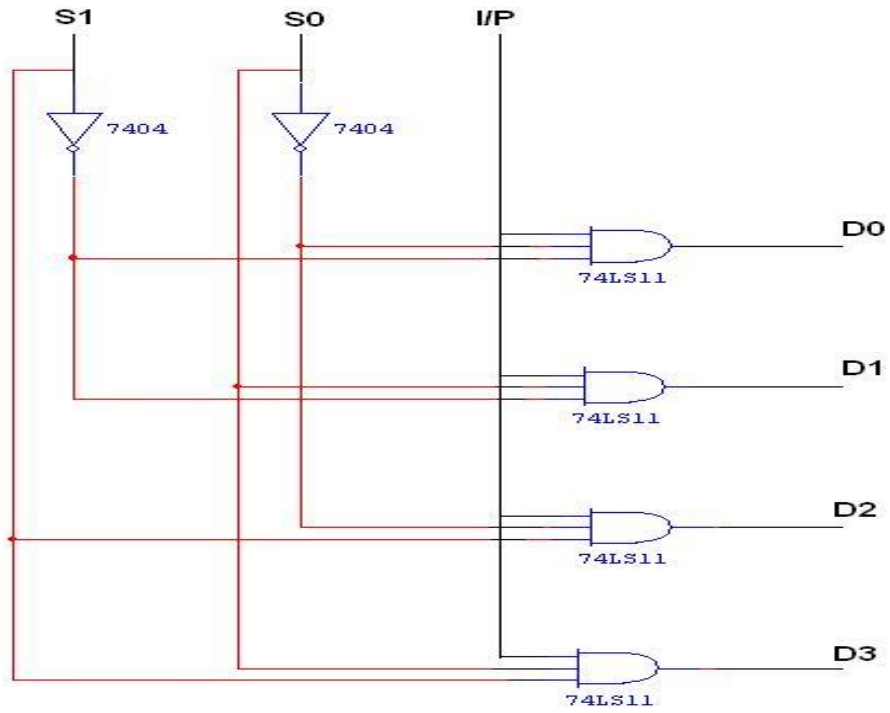
S1	S0	INPUT
0	0	X → D0 = X S1' S0'
0	1	X → D1 = X S1' S0
1	0	X → D2 = X S1 S0'
1	1	X → D3 = X S1 S0

Y = X S1' S0' + X S1' S0 + X S1 S0' + X S1 S0

TRUTH TABLE:

INPUT			OUTPUT			
S1	**S0**	**I/P**	**D0**	**D1**	**D2**	**D3**
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

LOGIC DIAGRAM FOR DEMULTIPLEXER:



PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

Thus the multiplexer and demultiplexer using logic gates are designed and implemented.

SHIFT REGISTER

AIM:

To design and implement the following shift registers
(i) Serial in serial out
(ii) Serial in parallel out
(iii) Parallel in serial out
(iv) Parallel in parallel out

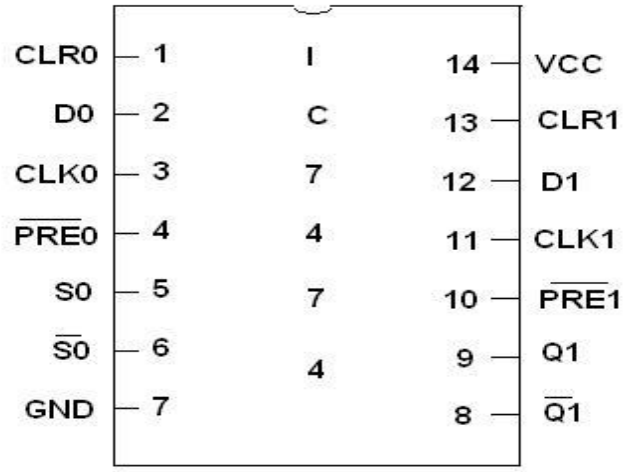
APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	D FLIP FLOP	IC 7474	2
2.	OR GATE	IC 7432	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	35

THEORY:

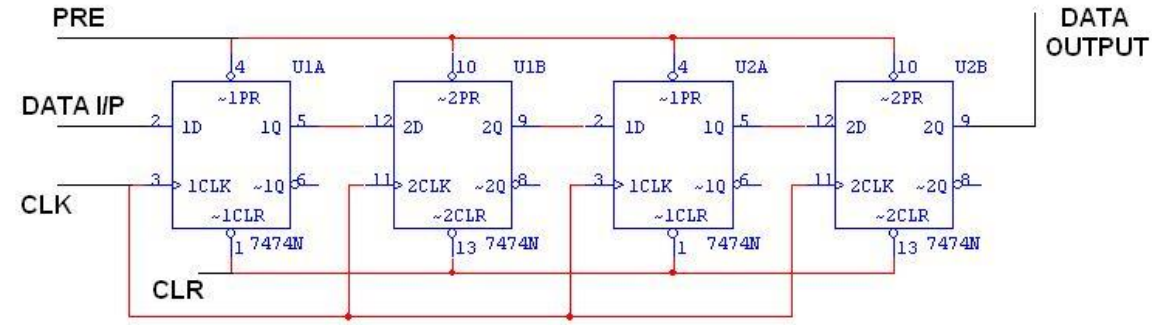
A register is capable of shifting its binary information in one or both directions is known as shift register. The logical configuration of shift register consist of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which causes the shift in the output of the flip flop. The simplest possible shift register is one that uses only flip flop. The output of a given flip flop is connected to the input of next flip flop of the register. Each clock pulse shifts the content of register one bit position to right.

PIN DIAGRAM OF IC 7474:



SERIAL IN SERIAL OUT

LOGIC DIAGRAM:

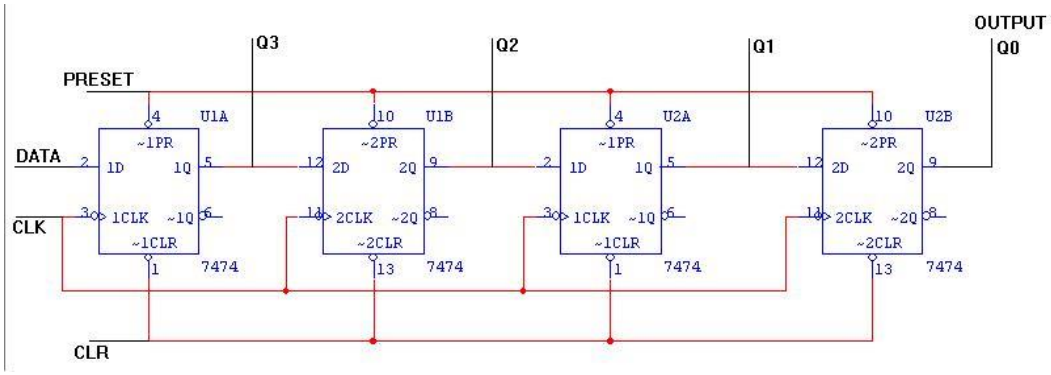


TRUTH TABLE:

CLK	Serial In	Serial Out
1	1	0
2	0	0
3	0	0
4	1	1
5	X	0
6	X	0
7	X	1

SERIAL IN PARALLEL OUT

LOGIC DIAGRAM:

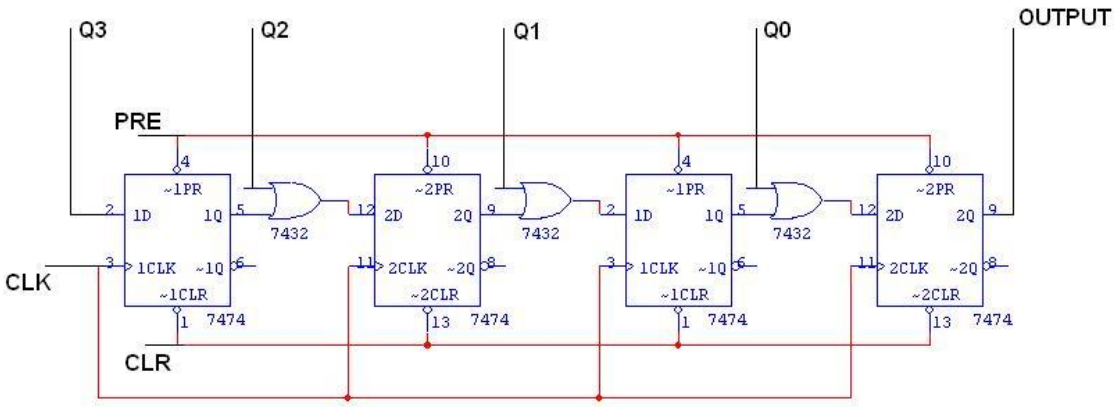


TRUTH TABLE:

CLK	DATA	OUTPUT			
		Q_A	Q_B	Q_C	Q_D
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	1

PARALLEL IN SERIAL OUT

LOGIC DIAGRAM:

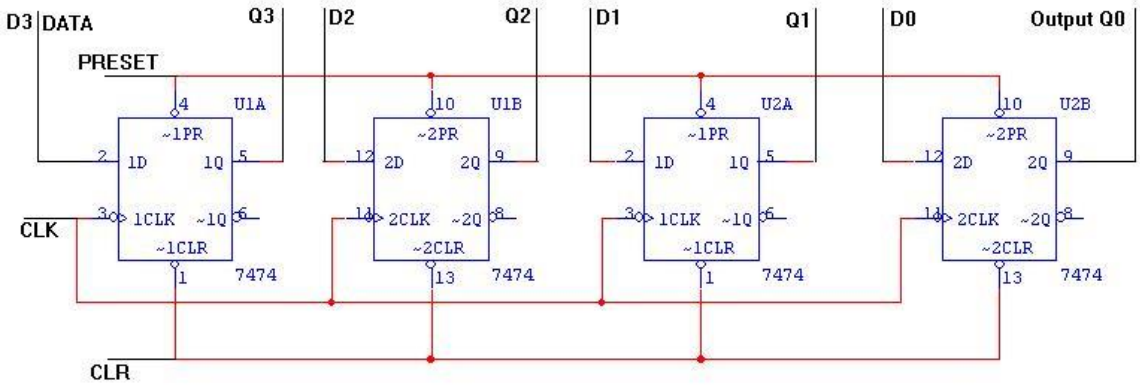


TRUTH TABLE:

CLK	Q3	Q2	Q1	Q0	O/P
0	1	0	0	1	1
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	1

PARALLEL IN PARALLEL OUT

LOGIC DIAGRAM:



TRUTH TABLE:

CLK	DATA INPUT				OUTPUT			
	D_A	D_B	D_C	D_D	Q_A	Q_B	Q_C	Q_D
1	1	0	0	1	1	0	0	1
2	1	0	1	0	1	0	1	0

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

The Serial in serial out, Serial in parallel out, Parallel in serial out and Parallel in parallel out shift registers are designed and implemented.

SYNCHRONOUS AND ASYNCHRONOUS COUNTER

AIM:

To design and implement synchronous and asynchronous counter.

APPARATUS REQUIRED:

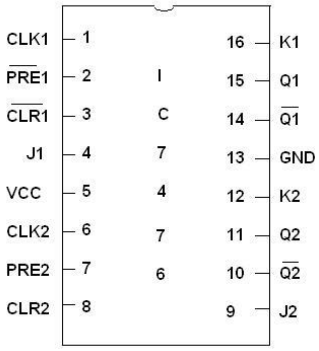
S.NO.	NAME OF THE APPARATUS	RANGE	QUANTITY
1.	Digital IC trainer kit		1
2.	JK Flip Flop	IC 7473	2
3.	D Flip Flop	IC 7473	1
4.	NAND gate	IC 7400	1
5.	Connecting wires		As required

THEORY:

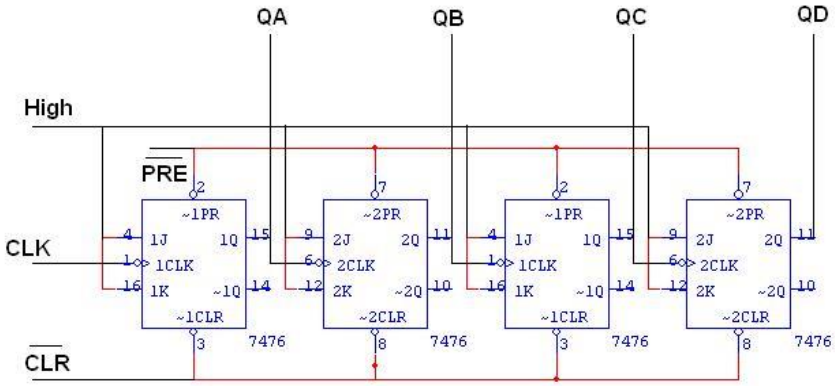
Asynchronous decade counter is also called as ripple counter. In a ripple counter the flip flop output transition serves as a source for triggering other flip flops. In other words the clock pulse inputs of all the flip flops are triggered not by the incoming pulses but rather by the transition that occurs in other flip flops. The term asynchronous refers to the events that do not occur at the same time. With respect to the counter operation, asynchronous means that the flip flop within the counter are not made to change states at exactly the same time, they do not because the clock pulses are not connected directly to the clock input of each flip flop in the counter.

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. A specified sequence of states appears as counter output. This is the main difference between a register and a counter. There are two types of counter, synchronous and asynchronous. In synchronous common clock is given to all flip flop and in asynchronous first flip flop is clocked by external pulse and then each successive flip flop is clocked by Q or Q output of previous stage. A soon the clock of second stage is triggered by output of first stage. Because of inherent propagation delay time all flip flops are not activated at same time which results in asynchronous operation.

PIN DIAGRAM FOR IC 7476:



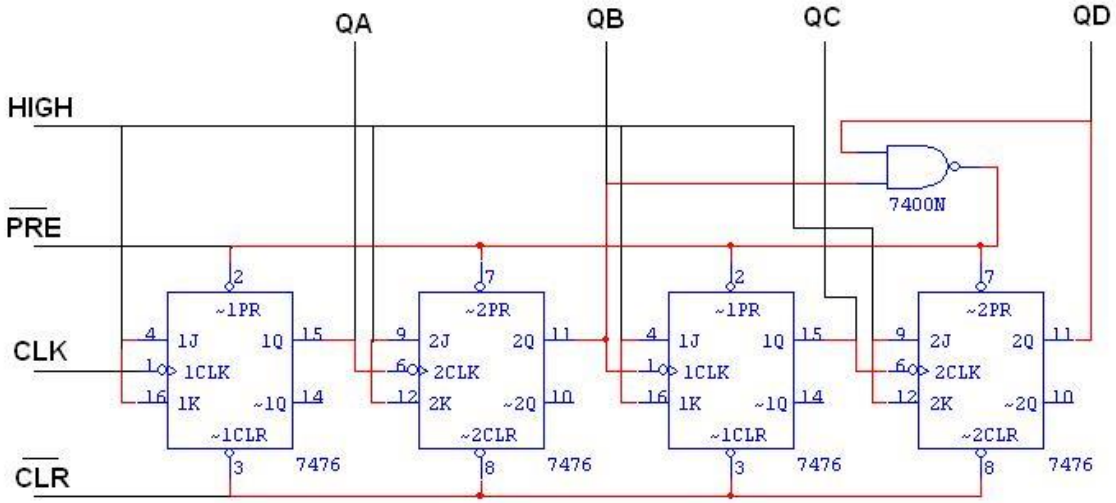
CIRCUIT DIAGRAM:



TRUTH TABLE:

CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

LOGIC DIAGRAM FOR MOD - 10 RIPPLE COUNTER:

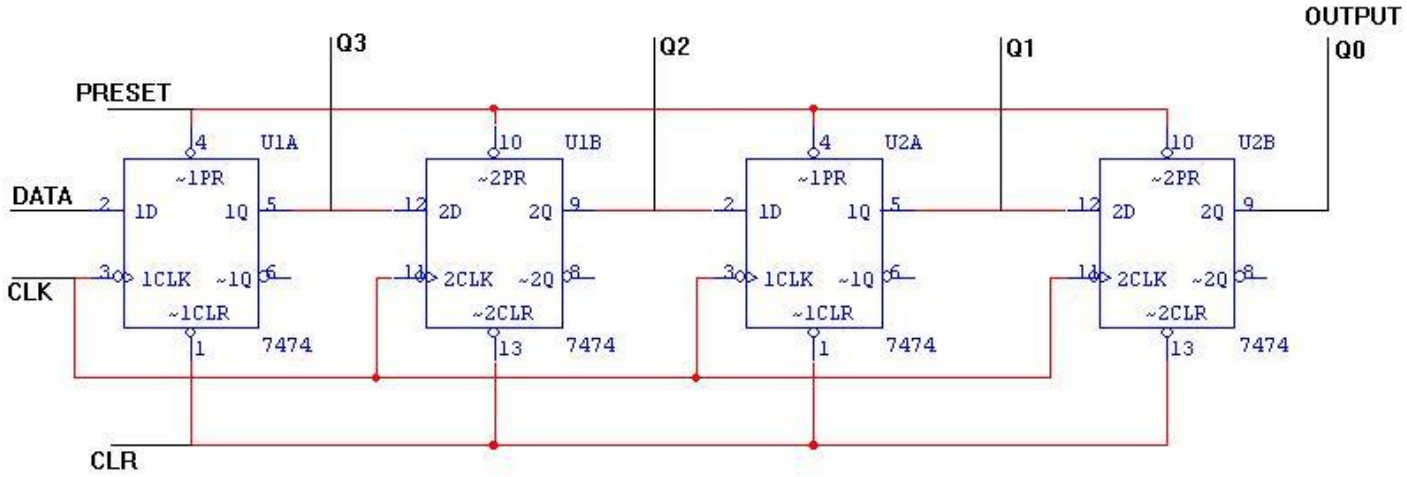


TRUTH TABLE:

CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	0	0	0

SYNCHRONOUS COUNTER

LOGIC DIAGRAM:



TRUTH TABLE:

CLK	DATA	OUTPUT			
		Q_A	Q_B	Q_C	Q_D
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	1

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

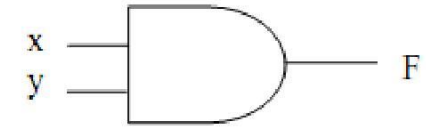
Thus the synchronous and asynchronous counter are designed and implemented.

IMPLEMENTATION OF BASIC LOGIC GATES

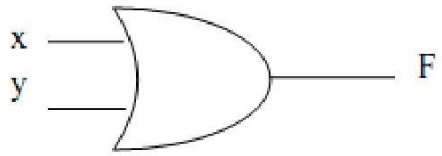
AIM:

To implement all the basic logic gates using Verilog and VHDL simulator.

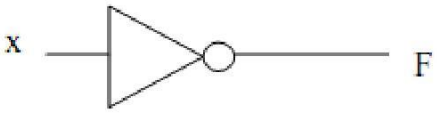
LOGIC GATE SYMBOLS **TRUTH TABLES**



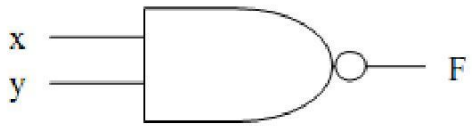
2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1



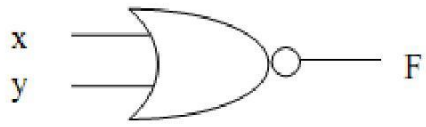
2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



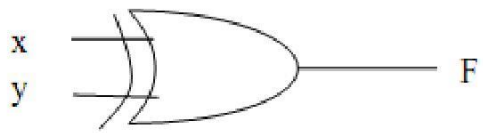
NOT gate	
A	$\overline{A}$
0	1
1	0



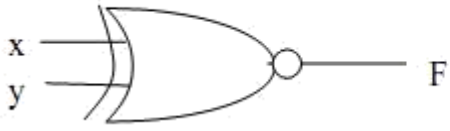
2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0



2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0



2 Input EXOR gate		
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

VERILOG CODE

AND GATE module and12(a,b,c); input a; input b; output c; assign c = a & b; endmodule	**OR GATE** *module or12(a,b,d);* *input a;* *input b;* *output d;* *assign d = a | b;* *endmodule*
NAND GATE *module nand12(a,b,e);* *input a;* *input b;* *output e;* *assign e = ~(a & b);* *endmodule*	**XOR GATE** *module xor12(a,b,h);* *input a;* *input b;* *output h;* *assign h = a ^ b;* *endmodule*
XNOR GATE *module xnor12(a,b,i);* *input a;* *input b;* *output i;* *assign i = ~(a ^ b);* *endmodule*	**NOR GATE** *module nor12(a,b,f);* *input a;* *input b;* *output f;* *assign f = ~(a | b);* *endmodule*
NOT GATE *module not12(a,g);* *input a;* *output g;* *assign g = ~a;* *endmodule*	

AND GATE

VERILOG CODE:

module and12(a,b,c);
input a;
input b;
output c;
assign c = a & b;
endmodule

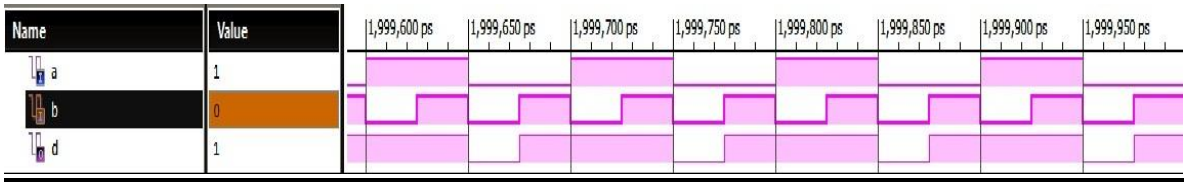
OUTPUT WAVEFORM:

OR GATE

VERILOG CODE:

```
module or12(a,b,d);
   input a;
   input b;
   output d;
   assign d = a | b;
endmodule
```

OUTPUT WAVEFORM:

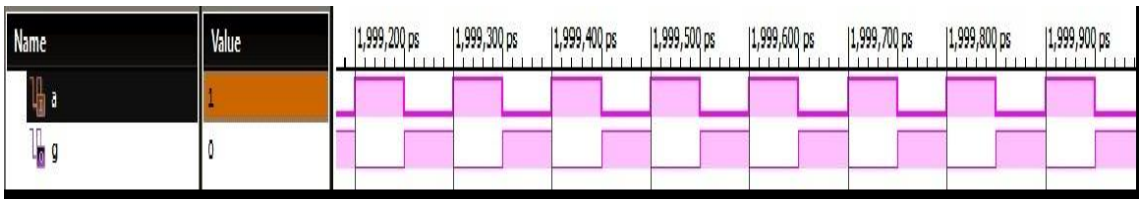


NOT GATE

VERILOG CODE:

```
module not12(a,g);
   input a;
   output g;
   assign g = ~a;
endmodule
```

OUTPUT WAVEFORM:



EX-OR GATE

VERILOG CODE:

```
module xor12(a,b,h);
        input a;
        input b;
        output h;
        assign h = a^ b;
endmodule
```

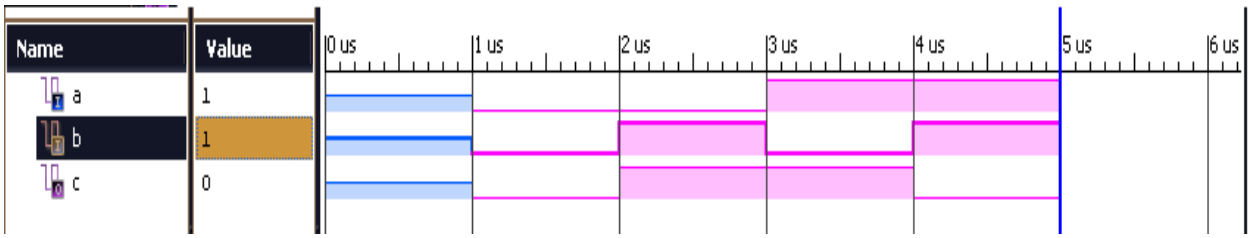
VHDL CODE:

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

entity xor_gate is
port (a,b : in std_logic ;
      c : out std_logic);

end xor_gate;
architecture Behavioral  o f xor_gate  is
begin
c <= a xor b;
end Behavioral;
```

OUTPUT WAVEFORM:



RESULT:

Thus all the basic logic gates are implemented and verified using Verilog and VHDL simulator.

Ex.No.-8 COMBINATIONAL AND SEQUENTIAL CIRCUITS

AIM:

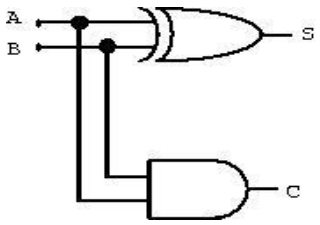
To simulate the sequential and combinational circuits using HDL simulator (Verilog and VHDL).

1. HALF ADDER

Truth Table

Input		Output	
A	B	S(Sum)	C(Carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Circuit Diagram



Graphical Notation

Equations

S (Sum) =A^B
C (Carry) =AB

Verilog Code:

```
module hadd(a,b,s,c); input a;
input b;
output s;
output c;
assign s = a ^ b;
assign c = a & b;
endmodule
```

Output:

VHDL Code:

```
library IEEE;
```

use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; entity halfadder is
port(
a : in std_logic; b : in std_logic;
sum : out std_logic; carry : out std_logic);
end halfadder;
architecture Behavioral of halfadder is begin
sum <= (a xor b); carry <= (a and b); end Behavioral;

Input:

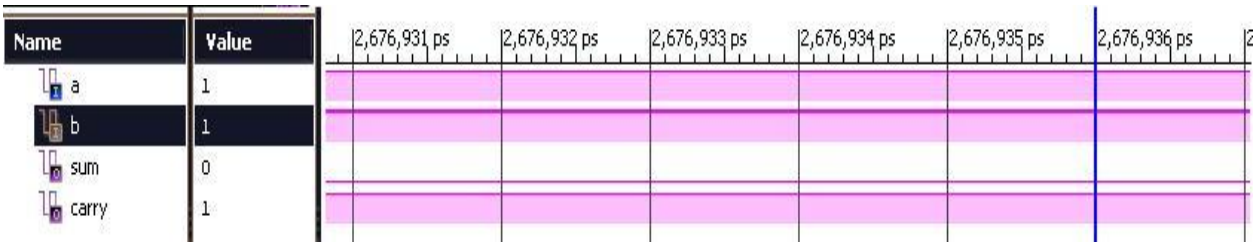
a : 1 ;

b :1;

Sum : 0

Carry : 1

Output:



2. FULL ADDER

Truth Table

Input			Output	
A	**B**	**C**	**SUM**	**Cout**
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

K- Map for sum

A \ BC	B'C'	B'C	BC	BC'
A'	0	(1)	0	(1)
A	(1)	0	(1)	0

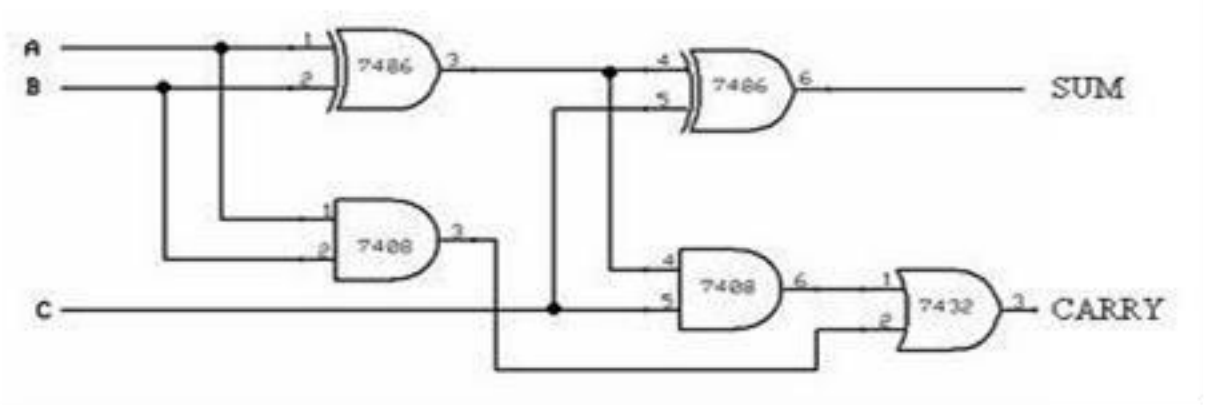
K-map for Carry

A \ BC	B'C'	B'C	BC	BC'
A'	0	0	1	0
A	0	1	1	1

H.ADDER **SUM** = A'B'C + A'BC' + AB'C' + ABC **Cout** = A'BC + AB'C + ABC' +ABC

SUM= A^B^C **Cout=** (A^B)C+AB

Circuit Diagram:



Verilog Code:

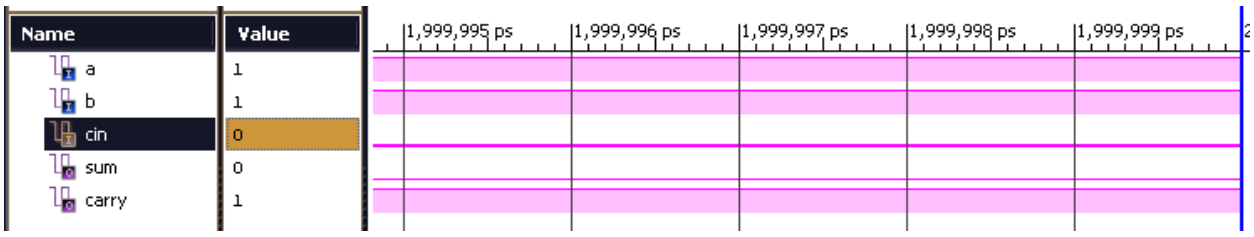
```
module fadd(a,b,c,s,cout);
   input a;
   input b;
   input c; output s;
   output cout;
   assign s = (a ^ b) ^ c;
   assign cout = (a & b)|( b & c)|(c & a);
endmodule
```

Output :

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fulladder is
port(
a : in std_logic;
b : in std_logic;
cin : in std_logic;
sum : out std_logic;
carry : out std_logic
);
end fulladder;
architecture Behavioral of fulladder is
begin
sum <= (a xor b xor cin);
carry <= (a and b) or (b and cin) or (a and cin);
end Behavioral;
```

Output:



3. HALF SUBTRACTOR

Verilog Code:

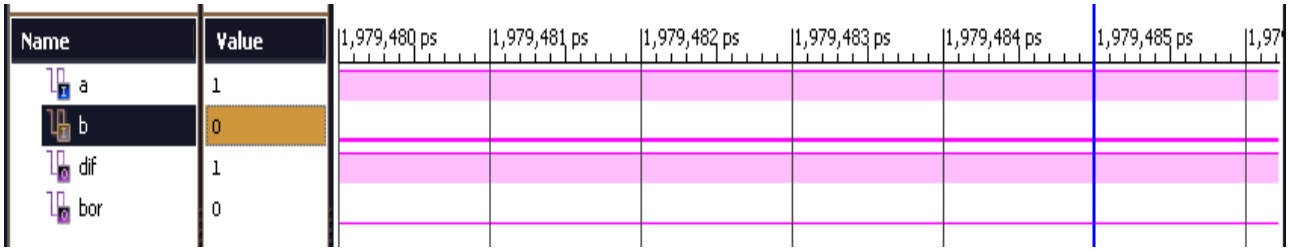
```
module hsub(a,b,d,bor);
Input a;
Input b;
output d;
output bor;
assign d=)a^b);
assign bor = (~a&~b);
end module
```

VHDL Code:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity halfsubtractor is
port(
a : in std_logic;
b : in std_logic;
dif : out std_logic;
bor : out std_logic
);
end halfsubtractor;
architecture Behavioral of halfsubtractor is
begin
dif <= a xor b;
bor <= ((not  a) and b);
end Behavioral;
```

Output:

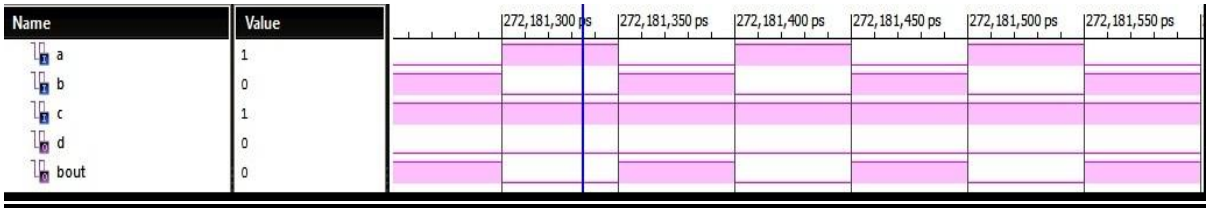


4. FULL SUBTRACTOR

Verilog Code:

```
module sub(a,b,c,d,b out);
    input a;
    input b;
    input c;
    output d;
    output bout;
    assign d = (a ^ b) ^ c;
    assign bout = (~a & b)|( b & c)|(c & ~a);
endmodule
```

Output:



VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fullsubtractor is
port( a : in std_logic;
b : in std_logic;
cin : in std_logic;
dif : out std_logic;
bor : out std_logic );
end fullsubtractor;
architecture Behavioral of fullsubtractor is begin
dif <= a xor b xor cin;
bor <= (((not a) and b) or (( not a) and cin) or (b and cin));
end Behavioral;
```

INPUT:

a : 0 ;

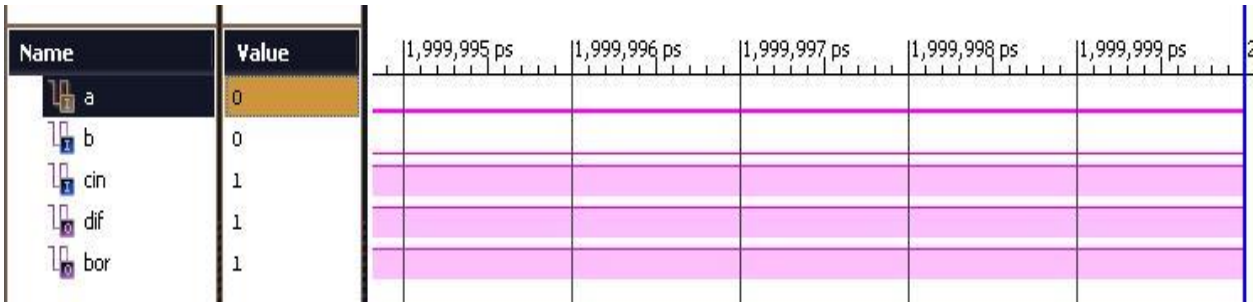
b :0;

Cin : 1

Difference : 1

Borrow : 1

Output:



5. MULTIPLEXER

Verilog Code:

```
module mux4to1(Y, I0,I1,I2,I3, sel);
   output Y;
   input I0,I1,I2,I3;
   input [1:0] sel;
   reg Y;
always @ (sel or I0 or I1 or I2 or I3)
case (sel)
```

```
        2'b00:Y=I0;
        2'b01:Y=I1;
        2'b10: Y=I2;
        2'b11: Y=I3;
endcase
endmodule
```

Output:

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; entity mux is
port(
inp : in std_logic_vector(3 downto 0); sel : in std_logic_vector(1 downto 0); muxout
: out std_logic --mux output line );
end mux;
architecture Behavioral of mux is begin
process(inp,sel) begin
case sel is when "00" =>
muxout <= inp(0); -- mux O/P=1 I/P-- when "01" =>
muxout <= inp(1); -- mux O/P=2 I/P-- when "10" =>
muxout <= inp(2); -- mux O/P=3 I/P-- when "11" =>
muxout <= inp(3); -- mux O/P=4 I/P-- when others =>
end case; end process;
end Behavioral;
```

Truth Table:

INPUTS						OUTPUT
sel1	sel0	inp0	inp1	inp2	inp3	muxout
0	0	I	0	0	0	I
0	1	0	I	0	0	I
1	0	0	0	I	0	I
1	1	0	0	0	I	I
NOTE : I means binary input which is either 0 or 1						

6. DEMULTIPLEXER

Verilog Code:

```
module demux(S,D,Y);
       Input [1:0] S;
       Input D;
       Output [3:0] Y; reg Y;
      always @(S OR )
      case({D,S})
          3'b100: Y=4'b0001;
          3'b101: Y=4'b0010;
          3'b110: Y=4'b0100;
          3'b111: Y=4'b1000;
          default:Y=4'b0000;
     endcase
endmodule
```

VHDL Code:

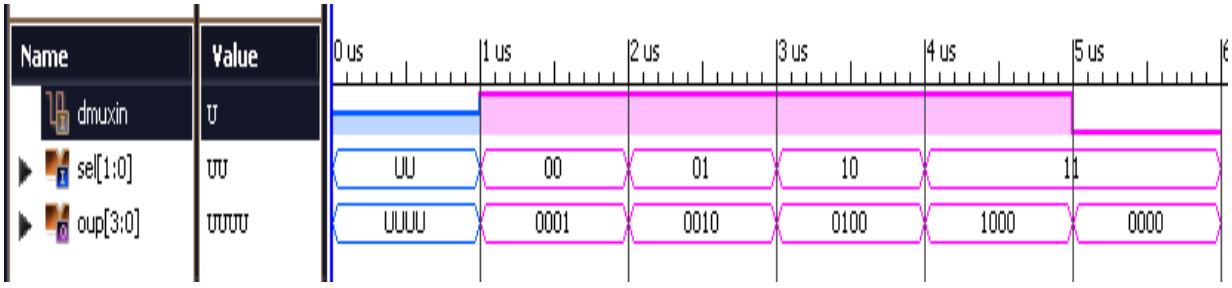
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity demux is
port(
dmuxin : in std_logic;
sel : in std_logic_vector(1 downto 0);
oup : out std_logic_vector(3 downto 0)
);
end demux;
architecture Behavioral of demux is
begin
process(dmuxin,sel)
begin
case sel is
when "00" =>
oup(0) <= dmuxin; --1 dmux o/p = dmux i/p--
oup(1) <= '0';
oup(2) <= '0';
oup(3) <= '0';
when "01" =>
oup(0) <= '0';
oup(1) <= dmuxin; --2 dmux o/p = dmux i/p--
oup(2) <= '0';
oup(3) <= '0';
when "10" =>
```

```
oup(0) <= '0';
oup(1) <= '0';
oup(2) <= dmuxin; --3 dmux o/p = dmux i/p--
oup(3) <= '0';
when "11" =>
oup(0) <= '0';
oup(1) <= '0';
oup(2) <= '0';
oup(3) <= dmuxin; --4 dmux o/p = dmux i/p--
when others =>
end case;
end process;
end Behavioral;
```

Truth Table:

INPUTS			OUTPUTS			
sel1	sel0	dmuxin	oup0	oup1	oup2	oup3
0	0	I	I	0	0	0
0	1	I	0	I	0	0
1	0	I	0	0	I	0
1	1	I	0	0	0	I
NOTE : I means binary input which is either 0 or 1						

Output:



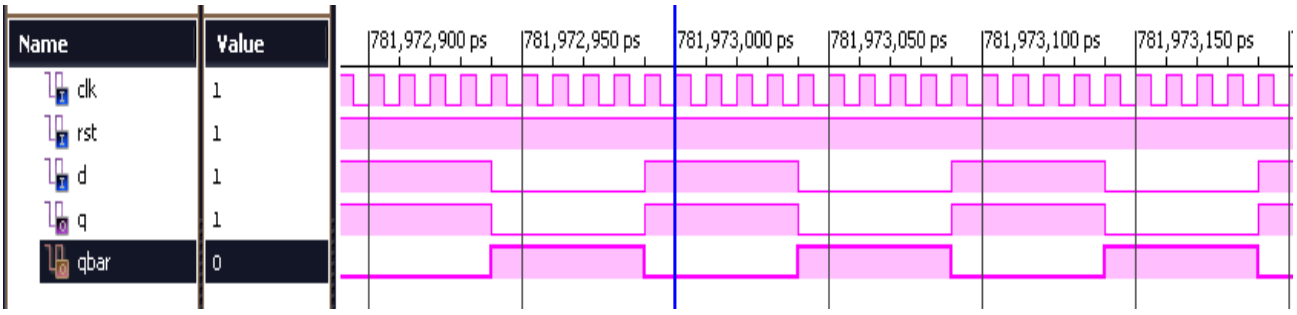
7. D FLIPFLOP

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity dff is
```

```
port(
clk : in std_logic; --clock input
rst : in std_logic; --active low,synchronous reset
d : in std_logic; --d input
q,qbar : out std_logic --flip flop outputs ie,Qn+1 and its complement
);
end dff;
architecture Behavioral of dff is
begin
process(clk,rst)
begin
if rising_edge(clk) then
if (rst = '0') then --active low,synchronous reset
q <= '0';
qbar <= '1';
else
q <= d;
qbar <= not(d);
end if;
end if;
end process;
end Behavioral;
```

Output:



8. T FLIPFLOP

Verilog Code :

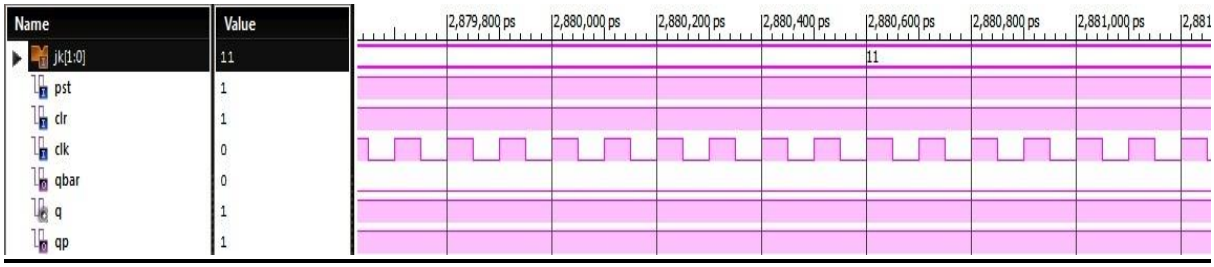
```
module tffeq(t,rst, clk,qp, qbar); input t,rst, clk;
   output qp, qbar; wire q;
   reg qp;
   always @ (posedge clk) if (rst)
            qp=0; else
             qp = q ^ t; assign qbar = ~ qp;
endmodule
```

9. JK FLIPFLOP

Verilog Code:

```
module jkff(jk,pst,clr,clk,qp,qbar);
input [1:0] jk;
input pst,clr,clk;
output qp,qbar;
   reg qp;
   wire q;
   always @ (posedge clk) if (pst)
            qp= 1;
            else
            begin
            if (clr)
            qp= 0;
            else
            begin
            case (jk)
                  2'b00: qp=q;
                  2'b01 : qp = 1'b0;
                  2'b10 : qp =1'b1;
                  2'b11 : qp = ~q;
                  default qp =0;
          endcase
          end
          end
          assign qbar = ~q;
          assign q = qp;
endmodule
```

Output:



10. RIPPLE COUNTER

Verilog Code:

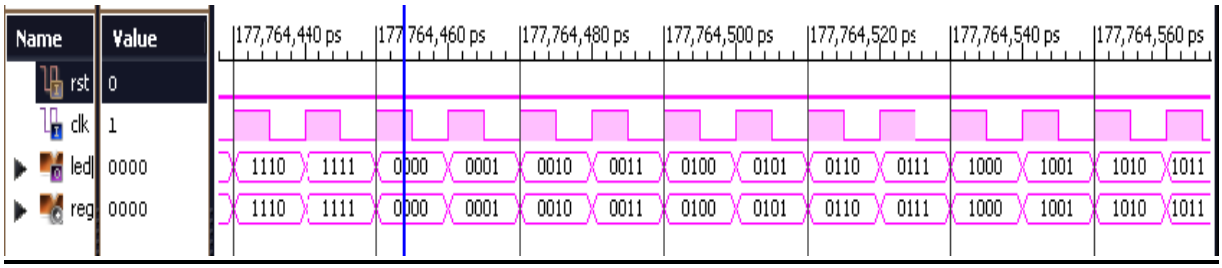
```
module ripple(clkr,st,,t,A,B,C,D);
input clk,rst,t;
output A,B,C,D;
Tff T0(D,clk,rst,t);
Tff T1(C,clk,rst,t);
Tff T2(B,clk,rst,t);
```

```
Tff T3(A,clk,rst,t);
endmodule
module Tff(q,clk,rst,t);
input clk,rst,t;
output q;
reg q;
always @(posedge clk)
begin
if(rst)
q<=1'b0; else
if(t)
q<=~q;
end
endmodule
```

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity counter is
Port ( rst : in STD_LOGIC;
clk : in STD_LOGIC;
led : out std_logic_vector(3 downto 0)
);
end counter;
architecture Behavioral of counter is
signal reg :std_logic_vector(3 downto 0);
begin
process(rst,clk)
begin
if rst = '1' then
reg <= "0000";
elsif rising_edge(clk) then
reg <= reg + 1;
end if;
end process;
led(3 downto 0) <= reg(3 downto 0);
end Behavioral;
```

Output:

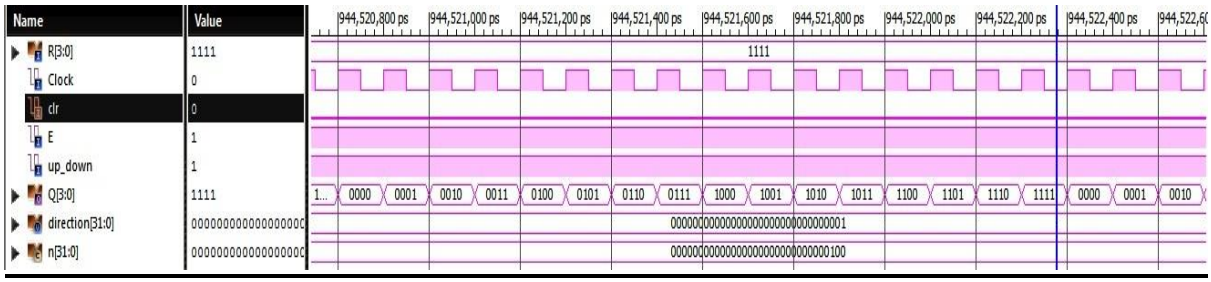


11. UPDOWN COUNTER

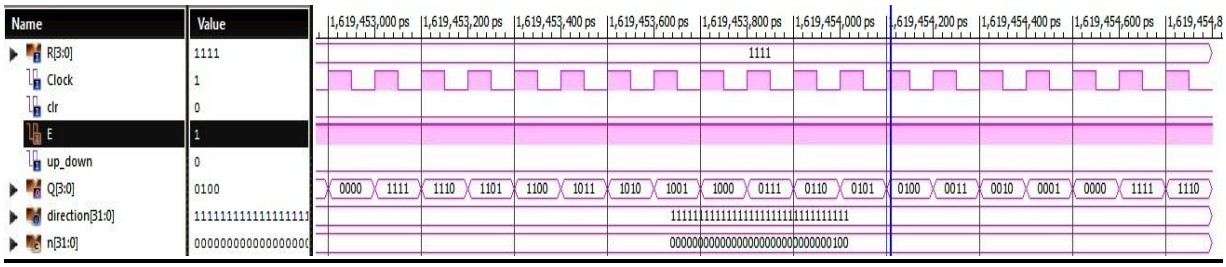
Verilog Code:

```
module updowncount (R, Clock, clr, E, up_down, Q);
parameter n = 4;
input [n-1:0] R;
input Clock, clr, E, up_down;
output [n-1:0] Q;
reg [n-1:0] Q;
integer direction;
always @(posedge Clock)
begin
if (up_down)  direction = 1;
else direction = -1;
if (clr)  Q <= R;
else if (E) Q <= Q + direction;
end
endmodule
```

UP Counter:



DOWN Counter:



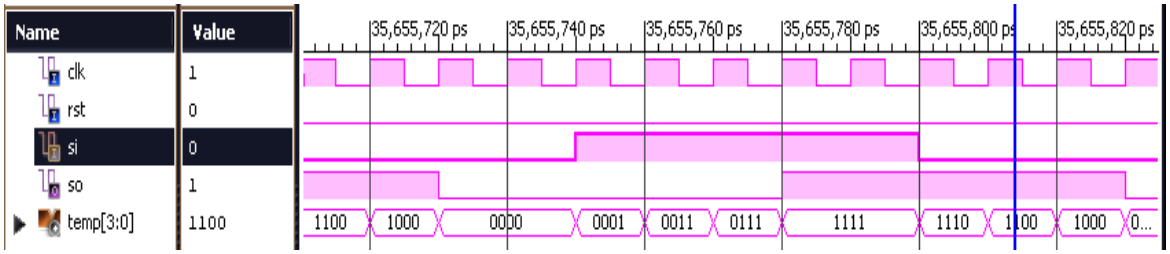
12. SHIFT REGISTER

a. Serial In Serial Out

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--library UNISIM;
--use UNISIM.VComponents.all;
entity hj is
port(
clk : in std_logic;
rst : in std_logic;
si: in std_logic;
so: out std_logic
);
end hj;
architecture Behavioral of hj is
signal temp : std_logic_vector(3 downto 0);
begin
process(clk,rst)
begin
if rising_edge(clk) then
if rst = '1' then
temp <= (others=>'0');
else
temp <= temp(2 downto 0) & si;
end if;
end if;
end process;
so <= temp(3);
end Behavioral;
```

Output:

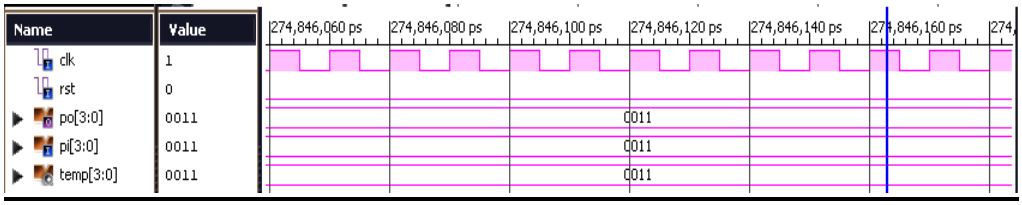


b. Parallel In Parallel Out

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--library UNISIM;
--use UNISIM.VComponents.all;
entity hj is
port(
clk : in std_logic;
rst : in std_logic;
po: out std_logic_vector(3 downto 0);
pi: in std_logic_vector(3 downto 0)
);
end hj;
architecture Behavioral of hj is
signal temp : std_logic_vector(3 downto 0);
begin
process(clk,rst)
begin
if rising_edge(clk) then
if rst = '1' then
temp <= (others=>'0');
else
temp <= pi(3 downto 0);
end if;
end if;
end process;
po <= temp(3 downto 0);
end Behavioral;
```

Output:



RESULT:

Thus the sequential and combinational circuits are designed and implemented using HDL simulator (Verilog and VHDL).