



TekBrains Pvt Ltd

(Advancing Technology & Research)

www.tekbrains.co.in

VLSI DESIGN MANUAL

Email: tkb@tekbrains.co.in

URL: www.tekbrains.co.in

Head Office/Sales & Marketing Branch

12, II Floor, A - 41, Main Market
Madhu Vihar, IP Ext.
New Delhi - 110092
Contact: +91-11 - 45873557, +91 -8826536536

Regd. Office/R&D Branch Advanced Electronics Lab

Malti Niwas, road no-1, Dhelwan, Near Mithapur
Near Mithapur Bypass road, Kankarbagh
Patna, Bihar, India, Pin: 800020
Contact: +91-8002359537, +91-612-2340844



Content

1. Combinational Logic Design
 - a. Logic Gates
 - b. Binary to Grey Converter
 - c. Mux
 - d. Demux
 - e. Decoder
 - f. BCD to 7 segment decoder
 - g. Encoder
 - h. Priority Encoder
 - i. Parity generator
 - j. Ripple Carry Adder

2. Sequential Logic Design
 - a. DFF
 - b. JK FF
 - c. T FF
 - d. Counter
 - e. Shifter

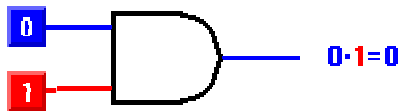
3. State Machine Design



Design Module: Logic Gates

Logic gates process signals which represent true or false. Normally the positive supply voltage +Vs represents true and 0V represents false. Other terms which are used for the true and false states are shown in the table on the right. It is best to be familiar with them all.

Gates are identified by their function: NOT, AND, NAND, OR, NOR, EX-OR and EX-NOR. Capital letters are normally used to make it clear that the term refers to a logic gate.



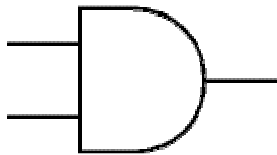
AND Gate

AND Gate

The output Y is true if input A AND input B are both true: **Y = A AND B**

An AND gate can have two or more inputs, its output is true if all inputs are true.

Logic states	
True	False
1	0
High	Low
+Vs	0V
On	Off



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1



VHDL Source 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;

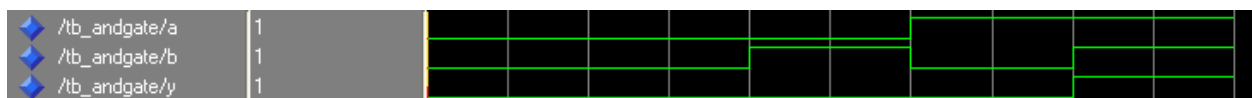
entity andgate is
    Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          y : out STD_LOGIC);
end orgate;

architecture arch_andgate of andgate is
begin
    y<= a and b;
end arch_andgate;
```

User Constraints File (UCF)

```
NET "A" LOC = "p55" ;
NET "B" LOC = "p56" ;
NET "Y" LOC = "p57" ;
```

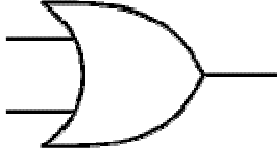
Simulation Waveform:



OR gate: The output Y is true if input A OR input B is true (or both of them are true):
 $Q = A \text{ OR } B$. An OR gate can have two or more inputs, its output is true if at least one input is true.



Traditional symbol



Truth Table

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

VHDL Source Code:

```
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 arch_orgate of orgate is  
begin  
  y<= a and b;  
end arch_orgate;
```

User Constraints File (UCF)

```
NET "A" LOC = "p55" ;  
NET "B" LOC = "p56" ;  
NET "Y" LOC = "p57" ;
```

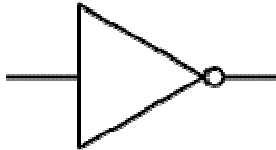
Simulator Waveform:





NOT gate (inverter): The output Q is true when the input A is NOT true, the output is the inverse of the input: $Y = \text{NOT } A$. A NOT gate can only have one input. A NOT gate is also called an inverter.

Traditional symbol



Truth Table

Input A	Output Y
0	1
1	0

VHDL Source Code:

```
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 arch_notgate of notgate is
begin
```

```
    y<= not a;
end arch_notgate;
```

User Constraints File (UCF)

```
NET "A" LOC = "p55" ;
NET "Y" LOC = "p57" ;
```

Simulation waveform

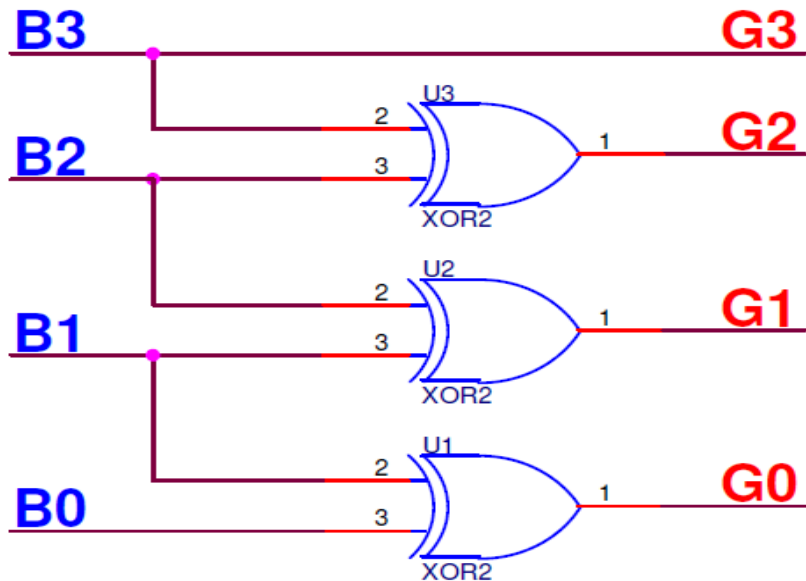




Design Module: BINARY TO GRAY CONVERTER

Binary –It is a number system, which has only two states ‘0’ (high) and ‘1’ (low)

Gray- In Gray code “ Every new code differs from the previous in terms of single bit” only one bit changes between successive numbers.



	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000



VHDL Source Code

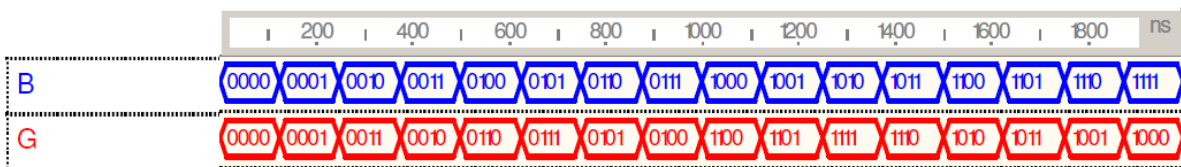
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity BTOG is
port (B3,B2,B1,B0: in STD_LOGIC;
G3,G2,G1,G0: out STD_LOGIC);
end BTOG;

architecture Behavioral of BTOG is
begin
G3<=B3;
G2<=B2 xor B3;
G1<=B1 xor B2;
G0<=B0 xor B1;
end Behavioral;
```

USER CONSTRAINTS

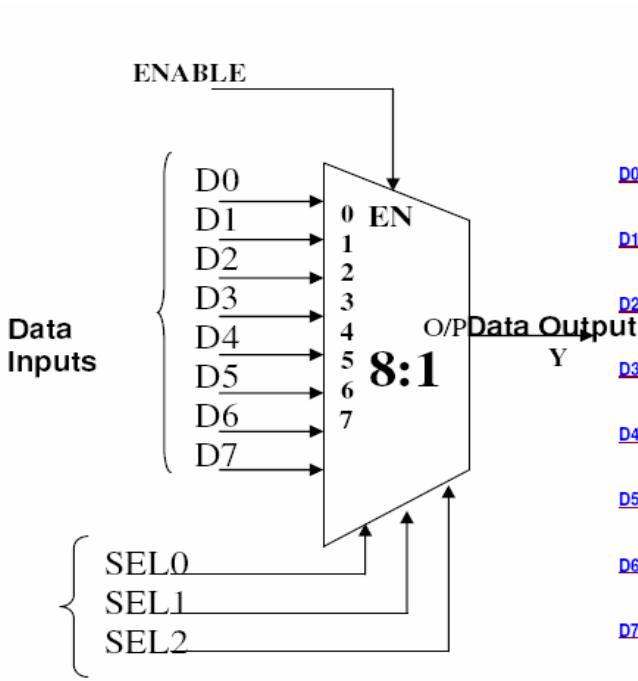
```
NET "B0" LOC = "p55" ;
NET "B1" LOC = "p56" ;
NET "B2" LOC = "p57" ;
NET "B3" LOC = "p58" ;
NET "G0" LOC = "p36" ;
NET "G1" LOC = "p35" ;
NET "G2" LOC = "p34" ;
NET "G3" LOC = "p33" ;
```

Simulator Waveforms for 4-Bit Binary to Gray Conversion:

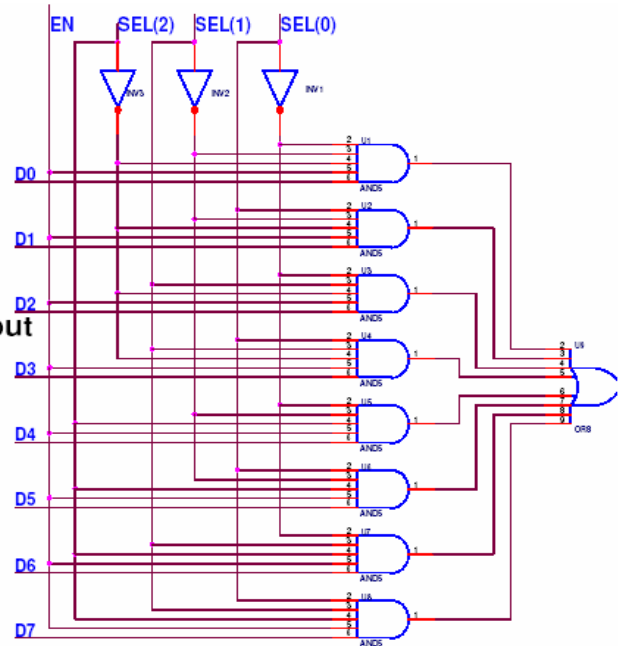


Design Module: 8:1 Multiplexer:

Multiplexer accepts several data inputs and allows only one output from any of the inputs at a time .



Control Inputs
Block Diagram of 8:1 Mux



Logic Diagram

VHDL Source 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 ( D: in STD_LOGIC_VECTOR (7 downto 0);
      EN: in STD_LOGIC;
      SEL: in STD_LOGIC_VECTOR (2 downto 0);
      Y: out STD_LOGIC );
end mux;
architecture Behavioral of mux is
begin
process(EN,SEL,D)
begin
if(EN='1')then
y<='0';
else
case SEL is
when "000" => y <= D(0);
when "001" => y <= D(1);
when "010" => y <= D(2);
when "011" => y <= D(3);
```

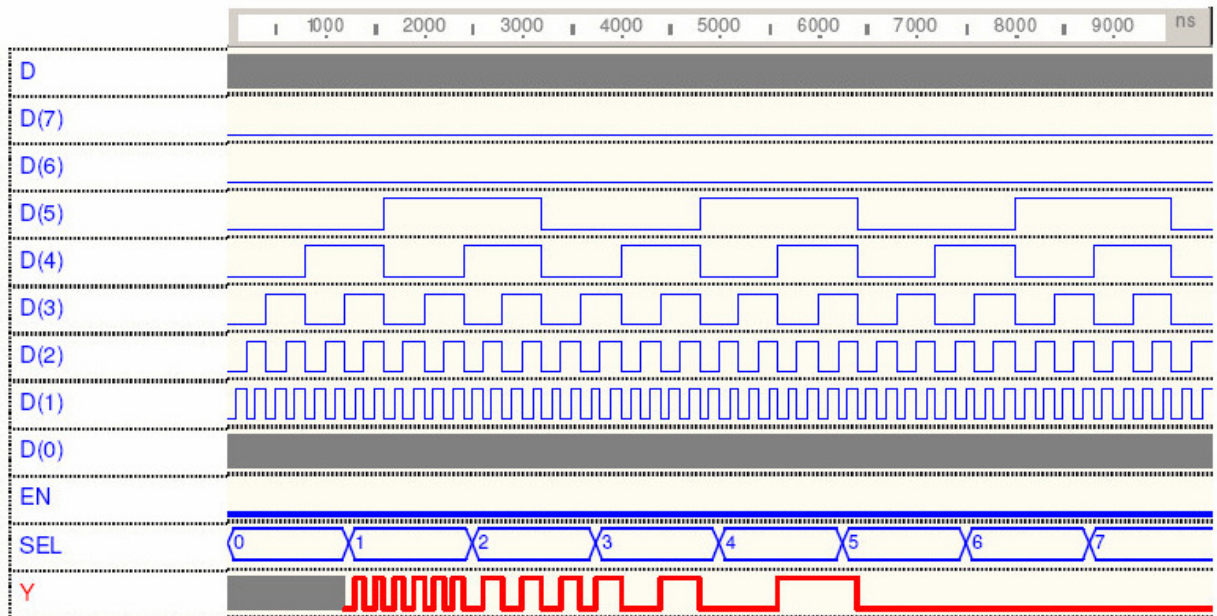


```
when "100" => y <= D(4);  
when "101" => y <= D(5);  
when "110" => y <= D(6);  
when others=> y <= D(7);  
end case;  
end if;  
end process;  
end Behavioral;
```

USER CONSTRAINTS:

```
net "sel<0>" loc=P55;  
net "sel<1>" loc=P56;  
net "sel<2>" loc=P57;  
net "d<0>" loc=P95;  
net "d<1>" loc=P96;  
net "d<2>" loc=P97;  
net "d<3>" loc=P98;  
net "d<4>" loc=P99;  
net "d<5>" loc=P100;  
net "d<6>" loc=P101;  
net "d<7>" loc=P102;  
net en loc=P58;  
net y loc=P36;
```

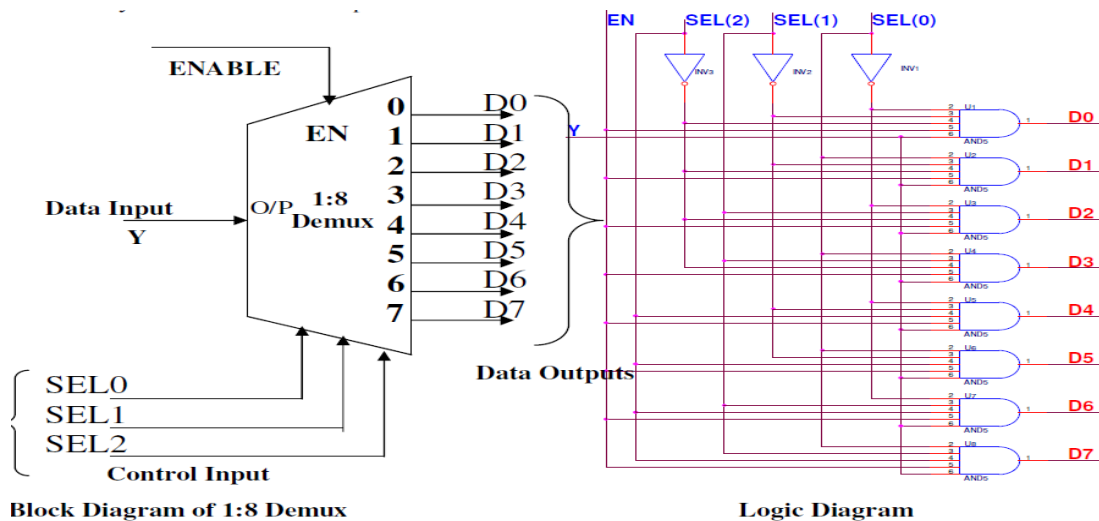
Simulator Waveform





Design Module: Demultiplexer

It gives several output from one input at a time thru Select lines.



Block Diagram of 1:8 Demux

Logic Diagram

VHDL Source 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 (Y : in STD_LOGIC;
EN : in STD_LOGIC;
SEL : in STD_LOGIC_VECTOR (2 downto 0);
D : out STD_LOGIC_VECTOR (7 downto 0));
end DEMUX;
architecture Behavioral of DEMUX is
begin
process(EN,SEL,Y)
begin
if(EN='1')then
D<=(others=>'0');
else
case SEL is
when "000" => D(0)<=Y;
when "001" => D(1)<=Y;
when "010" => D(2)<=Y;
when "011" => D(3)<=Y;
when "100" => D(4)<=Y;
when "101" => D(5)<=Y;
when "110" => D(6)<=Y;
```

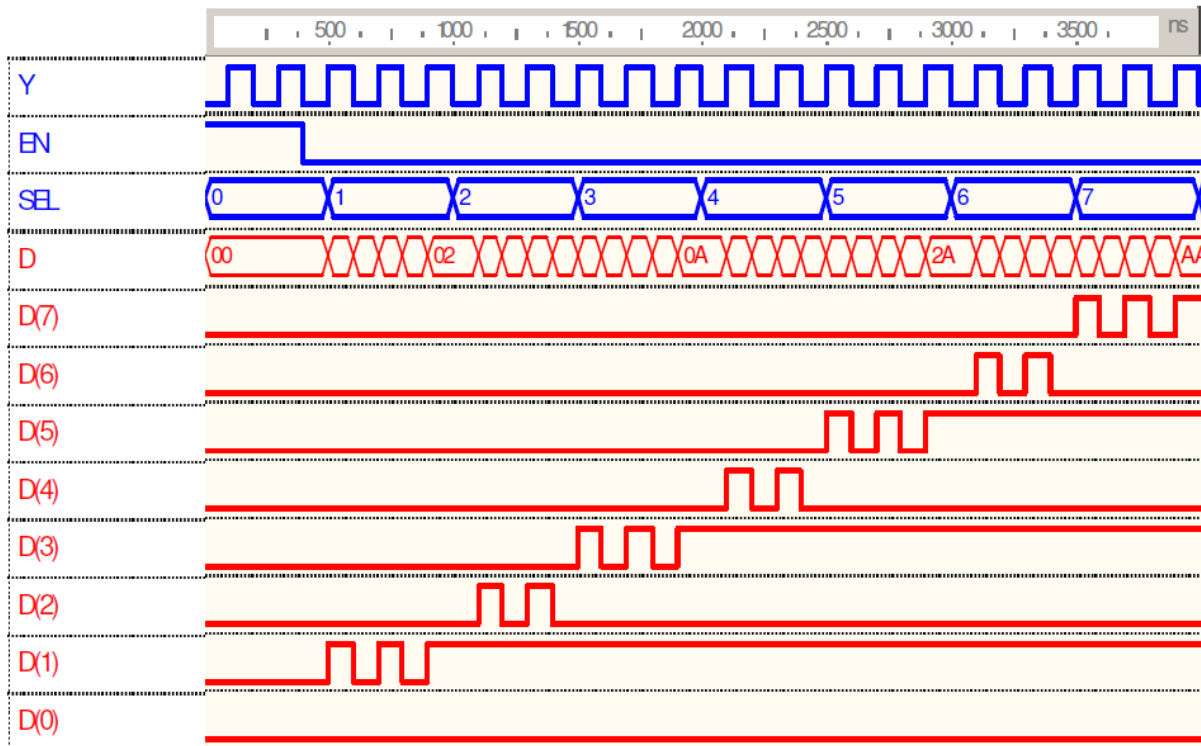


```
when others=> D(7)<=Y;  
end case;  
end if;  
end process;  
end Behavioral;
```

USER CONSTRAINTS

```
NET "D<0>" LOC = "P49" ;  
NET "D<1>" LOC = "P48" ;  
NET "D<2>" LOC = "P47" ;  
NET "D<3>" LOC = "P46" ;  
NET "D<4>" LOC = "P45" ;  
NET "D<5>" LOC = "P44" ;  
NET "D<6>" LOC = "P43" ;  
NET "D<7>" LOC = "P42" ;  
NET "EN" LOC = "P55" ;  
NET "SEL<0>" LOC = "P56" ;  
NET "SEL<1>" LOC = "P57" ;  
NET "SEL<2>" LOC = "P58" ;  
NET "Y" LOC = "P95" ;
```

Simulation Waveform:

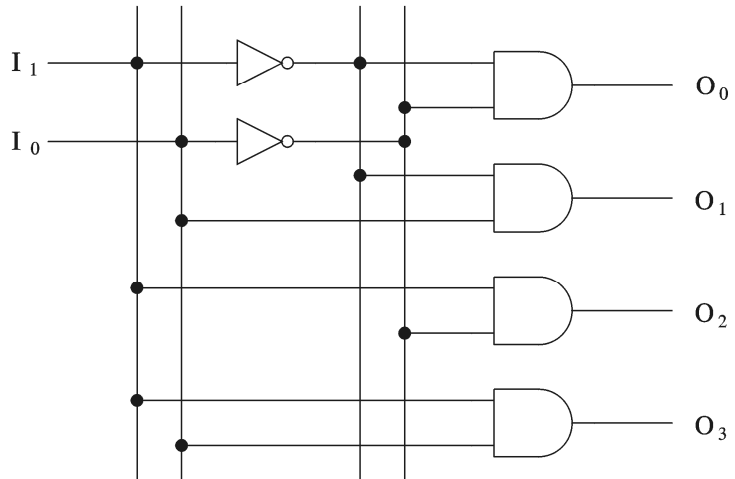
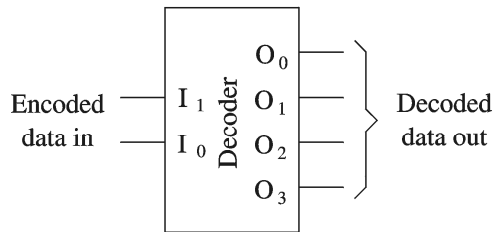




Design Module: 3:8 Decoder

Decoder selects one out of N Inputs

I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



VHDL Source Code

```

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

entity decoder is
  port (sel: in std_logic_vector (2 downto 0);
        res: out std_logic_vector (7 downto 0));
end decoder;
architecture arch_decoder of decoder is
  begin
    res <= "00000001" when sel = "000" else
           "00000010" when sel = "001" else
           "00000100" when sel = "010" else
           "00001000" when sel = "011" else
           "00010000" when sel = "100" else
           "00100000" when sel = "101" else
           "01000000" when sel = "110" else
           "10000000";
  end arch_decoder;

```

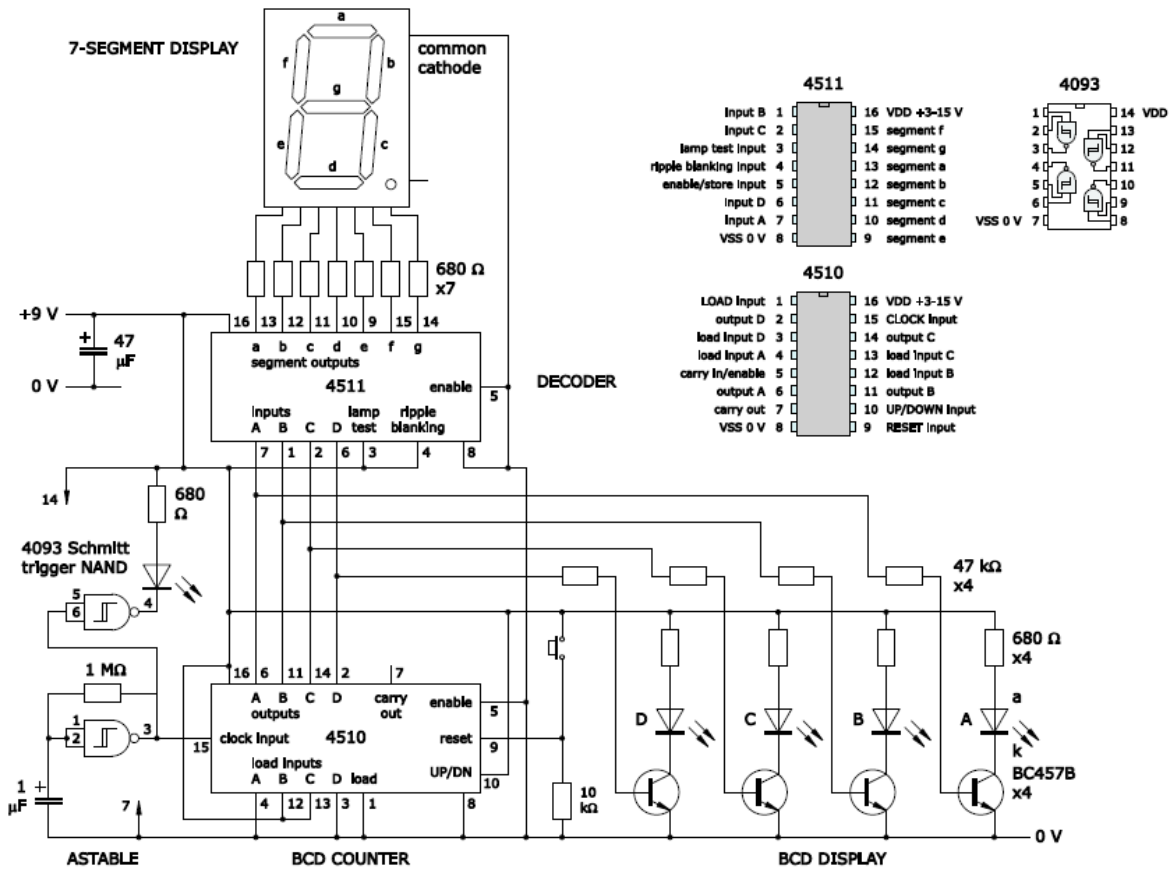


Design Module: BCD TO SEVEN SEGMENT DECODER

There are two important types of 7-segment LED display. In a common cathode display, the cathodes of all the LEDs are joined together and the individual segments are illuminated by HIGH voltages. In a common anode display, the anodes of all the LEDs are joined together and the individual segments are illuminated by connecting to a LOW voltage.

<i>BCD inputs</i>				<i>segment outputs</i>							<i>display</i>
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9

In normal operation, the lamp test and ripple blanking inputs are connected HIGH, and the enable (store) input is connected LOW. The circuit diagram shows the 4511 and a 7-segment common cathode display connected to the outputs of a 4510 BCD counter:



VHDL Source Code

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity bcd is
Port ( bcd : in std_logic_vector(3 downto 0);
leds : out std_logic_vector(7 downto 0);
q:out std_logic_vector(3 downto 0));
end bcd;
architecture Behavioral of bcd is
begin
q<="1111";
process(bcd)
begin
case bcd is --abcdefgdp
when "0000"=> leds<="00000011";
when "0001"=> leds<="10011111";
when "0010"=> leds<="00100101";

```



```
when "0011"=> leds<="00001101";
when "0100"=> leds<="10011 001";
when "0101"=> leds<="01001001";
when "0110"=> leds<="01000001";
when "0111"=> leds<="00011111";
when "1000"=> leds<="00000001";
when "1001"=> leds<="00001001";
when others=> leds<="11111111";
end case;
end process;
end Behavioral;
```

User Constraint File

```
NET "bcd<0>" LOC = "P55" ;
NET "bcd<1>" LOC = "P56" ;
NET "bcd<2>" LOC = "P57" ;
NET "bcd<3>" LOC = "P58" ;
NET "leds<0>" LOC = "P167" ;
NET "leds<1>" LOC = "P166" ;
NET "leds<2>" LOC = "P165" ;
NET "leds<3>" LOC = "P164" ;
NET "leds<4>" LOC = "P163" ;
NET "leds<5>" LOC = "P162" ;
NET "leds<6>" LOC = "P161" ;
NET "leds<7>" LOC = "P160" ;
NET "q<0>" LOC = "P175" ;
NET "q<1>" LOC = "P176" ;
NET "q<2>" LOC = "P174" ;
NET "q<3>" LOC = "P173" ;
```

Design Module: 7 SEGMENT DISPLAY

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- Title "7 segment display driver circuit";
-- File: seg7.vhd
ENTITY seg7display IS
PORT (D      : IN  STD_LOGIC_VECTOR (3 DOWNTO 0); -- BCD input
      S      : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)); -- 7 segment outputs
END seg7;
ARCHITECTURE arch_seg7display OF seg7display IS
BEGIN
s <=      "1000000" WHEN d = "0000" ELSE -- H"40"
          "1111001" WHEN d = "0001" ELSE -- H"79"
          "0100100" WHEN d = "0010" ELSE -- H"24"
          "0110000" WHEN d = "0011" ELSE -- H"30"
          "0011001" WHEN d = "0100" ELSE -- H"19"
```



```

"0010010" WHEN d = "0101" ELSE -- H"12"
"0000010" WHEN d = "0110" ELSE -- H"02"
"1111000" WHEN d = "0111" ELSE -- H"78"
"0000000" WHEN d = "1000" ELSE -- H"00"
"0010000" WHEN d = "1001" ELSE -- H"10"
"0001000" WHEN d = "1010" ELSE -- H"08"
"0000011" WHEN d = "1011" ELSE -- H"03"
"1000110" WHEN d = "1100" ELSE -- H"46"
"0100001" WHEN d = "1101" ELSE -- H"21"
"0000110" WHEN d = "1110" ELSE -- H"06"
"0001110";           -- H"0E"

```

END arch_seg7display;

Design Module : Encoder

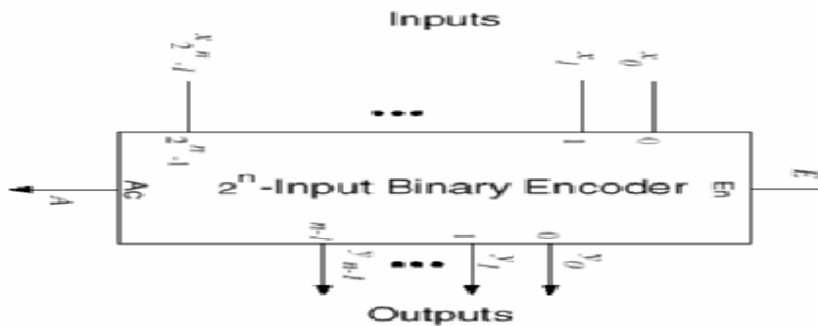
A logic circuit that produces coded binary outputs from uncoded inputs.

Priority encoder:

Whenever two or more inputs are applied at a time, internal hardware will check this condition and if the priority is set such that higher numbered input should be taken into account and remaining are considered as don't care then

Truth table for 8-input priority encoder

EN	DIN (7:0)	EOUT
0	X X X X X X X X	0
1	X X X X X X X 0	0
1	X X X X X X 0 1	1
1	X X X X X 0 1 1	2
1	X X X X 0 1 1 1	3
1	X X X 0 1 1 1 1	4
1	X X 0 1 1 1 1 1	5
1	X 0 1 1 1 1 1 1	6
1	0 1 1 1 1 1 1 1	7
1	1 1 1 1 1 1 1 1	0



Block Diagram of priority encoder



VHDL Source 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 ENCODER is
Port ( a : in unsigned (7 downto 0);
y : out unsigned(2 downto 0));
end ENCODER;
```

```
architecture Behavioral of ENCODER is
begin
process(a)
begin
case a is
when"00000001"=> y <="000";
when"00000010"=> y <="001";
when"00000100"=> y <="010";
when"00001000"=> y <="011";
when"00010000"=> y <="100";
when"00100000"=> y <="101";
when"01000000"=> y <="110";
when "10000000" => y<="111";
when others => y<="000";
end case;
end process;
end Behavioral;
```

USER CONSTRAINTS

```
net "a<0>" loc=P55;
net "a<1>" loc=P56;
net "a<2>" loc=P57;
net "a<3>" loc=p58;
net "a<4>" loc=P59;
net "a<5>" loc =P60;
net "a<6>" loc=P61;
net "a<7>" loc=p62;
net "y<0>" loc=p36;
net "y<1>" loc=P35;
net "y<2>" loc=P34;
```

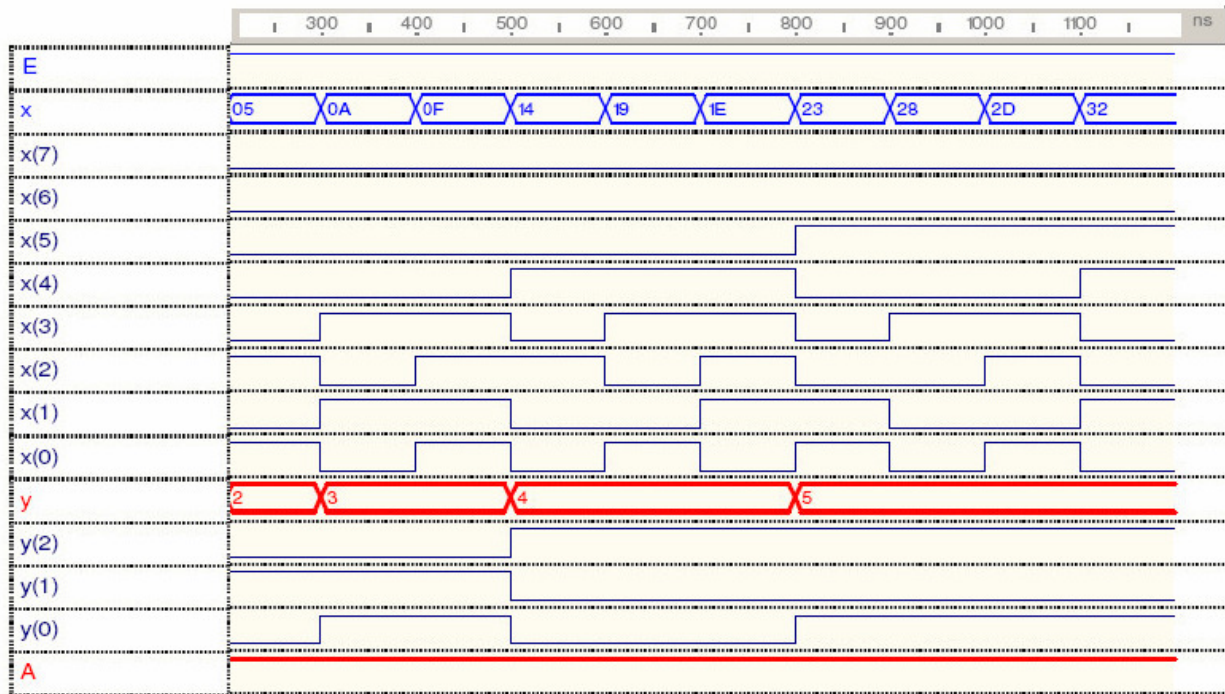


VHDL Source code of Priority Encoder

```
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 PRIOENCODER is
Port ( a : in unsigned(7 downto 0);
y : out unsigned(2 downto 0));
end PRIOENCODER;
architecture Behavioral of PRIOENCODER is
begin
process(a)
begin
if (a(7)='1') then y<="111";
elsif(a(6)='1') then y<="110";
elsif(a(5)='1') then y<="101";
elsif(a(4)='1') then y<="100";
elsif(a(3)='1') then y<="011";
elsif(a(2)='1') then y<="010";
elsif(a(1)='1') then y<="001";
elsif(a(0)='1') then y<="000";
else y<="000";
end if;
end process;
end Behavioral;
USER CONSTRAINTS
net "a<7>" loc=p62;
net "a<6>" loc=P61;
net "a<5>" loc =P60;
net "a<4>" loc=P59;
net "a<3>" loc=p58;
net "a<2>" loc=P57;
net "a<1>" loc=P56;
net "a<0>" loc=P55;
net "y<0>" loc=p36;
net "y<1>" loc=P35;
net "y<2>" loc=P34;
```



Simulation Waveform:



Design module: IC 7485

```
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 IC7485 is
Port ( a,b : in std_logic_vector(3 downto 0);
gt,eq,lt : in std_logic;
a_gt_b,a_eq_b,a_lt_b : out std_logic);
end IC7485;
architecture Behavioral of IC7485 is
begin
process(a,b,gt,lt,eq)
begin
if a>b then
a_gt_b<='1';
a_eq_b<='0';
```



```
a_lt_b<='0';
elsif a<b then
a_gt_b<='0';
a_eq_b<='0';
a_lt_b<='1';
elsif a=b then
a_gt_b<='0';
a_lt_b<='0';
a_eq_b<='1';
ELSE
a_gt_b<=GT;
a_lt_b<=LT;
a_eq_b<=EQ;
end if;
end process;
end Behavioral;
```

USER CONSTRAINTS

```
NET "a<0>" LOC = "p55" ;
NET "a<1>" LOC = "p56" ;
NET "a<2>" LOC = "p57" ;
NET "a<3>" LOC = "p58" ;
NET "a_eq_b" LOC = "p36" ;
NET "a_gt_b" LOC = "p35" ;
```

```
NET "a_lt_b" LOC = "p34" ;
NET "b<0>" LOC = "p59" ;
NET "b<1>" LOC = "p60" ;
NET "b<2>" LOC = "p61" ;
NET "b<3>" LOC = "p62" ;
```

VHDL Source Code for IC74318

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

entity IC74138 is
port (
S: in STD_LOGIC_VECTOR(2 DOWNTO 0);
OE1: in STD_LOGIC;--G1
OE2: in STD_LOGIC;--G2
Y: out STD_LOGIC_VECTOR (7 downto 0) );
end IC74138;
architecture Behavioral of IC74138 is
SIGNAL T: STD_LOGIC_VECTOR(7 DOWNTO 0);
```



```
begin
PROCESS(OE1,OE2,S)
BEGIN
IF OE2='1' OR OE1='0' THEN
T <= (OTHERS=>'1');
ELSIF OE1='1' AND OE2='0' THEN
CASE S IS
WHEN "000" =>
T <= "11111110";
WHEN "001" =>
T <= "11111101";
WHEN "010" =>
T <= "11111011";
WHEN "011" =>
T <= "11110111";
WHEN "100" =>
T <= "11101111";
WHEN "101" =>
T <= "11011111";
WHEN "110" =>
T <= "10111111";
WHEN OTHERS =>
T <= "01111111";
END CASE;
ELSE
T<= "11111111";

END IF;
END PROCESS ;
Y<=T;
end Behavioral;
```

USER CONSTRAINTS

```
NET "S<0>" LOC = "p55" ;
NET "S<1>" LOC = "p56" ;
NET "S<2>" LOC = "p57" ;
NET "OE1" LOC = "p95" ;
NET "OE2" LOC = "p96" ;
NET "Y<0>" LOC = "p36" ;
NET "Y<1>" LOC = "p35" ;
NET "Y<2>" LOC = "p34" ;
NET "Y<3>" LOC = "p33" ;
NET "Y<4>" LOC = "p30" ;
NET "Y<5>" LOC = "p29" ;
NET "Y<6>" LOC = "p23" ;
net "y<7>" loc="p22";
```



Design Module: Parity Generator

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

entity paritygen is
PORT(INPUT:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
OUTPUT: OUT STD_LOGIC_VECTOR(8 DOWNTO 0));
end paritygen;
architecture Behavioral of paritygen is
begin
process(input)
VARIABLE TEMP1:STD_LOGIC;
VARIABLE TEMP2:STD_LOGIC_VECTOR(8 DOWNTO 0);
Begin
TEMP1:='0';
FOR I IN 0 TO 7 LOOP
TEMP1:=TEMP1 XOR INPUT(I);
TEMP2(I):=INPUT(I);
END LOOP;
TEMP2(OUTPUT'HIGH):=TEMP1;
OUTPUT<=TEMP2;
END PROCESS;
end Behavioral;
USER CONSTRAINTS
NET "INPUT<0>" LOC = "P55" ;
NET "INPUT<1>" LOC = "P56" ;
NET "INPUT<2>" LOC = "P57" ;
NET "INPUT<3>" LOC = "P58" ;
NET "INPUT<4>" LOC = "P59" ;
NET "INPUT<5>" LOC = "P60" ;
NET "INPUT<6>" LOC = "P61" ;
NET "INPUT<7>" LOC = "P62" ;
NET "OUTPUT<0>" LOC = "P36" ;
NET "OUTPUT<1>" LOC = "P35" ;
NET "OUTPUT<2>" LOC = "P34" ;
NET "OUTPUT<3>" LOC = "P33" ;
NET "OUTPUT<4>" LOC = "P30" ;
NET "OUTPUT<5>" LOC = "P29" ;
NET "OUTPUT<6>" LOC = "P23" ;
NET "OUTPUT<7>" LOC = "P22" ;
NET "OUTPUT<8>" LOC = "P21" ;
```



Design Module: Ripple Carry Adder

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

entity ripplecarryadder is
Port ( a,b : in std_logic_vector (3 downto 0);
cin : in std_logic;
s : out std_logic_vector (3 downto 0);
cout : out std_logic);
end ripplecarryadder;
architecture Behavioral of ripplecarryadder is
begin
process(a,b,cin)
variable carry:std_logic_vector (4 downto 0);
begin
carry(0):=cin;
for i in 0 to 3 loop
s(i)<=a(i) xor b(i) xor carry(i);
carry(i+1):=(a(i) and b(i)) or (a(i) and carry(i))
or (b(i) and carry(i));
end loop;
cout<=carry(4);
end process;
end Behavioral;
USER CONSTRAINTS
NET "a<0>" LOC = "p55" ;
NET "a<1>" LOC = "p56" ;
NET "a<2>" LOC = "p57" ;
NET "a<3>" LOC = "p58" ;
NET "b<0>" LOC = "p59" ;
NET "b<1>" LOC = "p60" ;
NET "b<2>" LOC = "p61" ;
NET "b<3>" LOC = "p62" ;
NET "cin" LOC = "p95" ;
NET "cout" LOC = "p36" ;
NET "s<0>" LOC = "p20" ;
NET "s<1>" LOC = "p21" ;
NET "s<2>" LOC = "p22" ;
NET "s<3>" LOC = "p23" ;
```



Design Module: DFF

In DFF, the transfer of data from input to the output is delayed and hence the name D-Flip flop. The D-Flip flop is either used as delay device or as a latch to store '1' bit of binary information. D input transferred to Q output when clock is asserted.

D-F/F Truth table

D	Q ⁺	Action
0	0	Reset
1	1	Set

Output = input when clock is applied.

VHDL Source 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(d: in std_logic;
      clk: in std_logic;
      q : out std_logic
);
Architecture arch_dff of dff is
Begin
Process(clk)
Begin
If clk = '1' and clk'event then
Q<=d;
End if;
End process;
End arch_dff;
```



VHDL Source code DFF IC: IC 7474

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

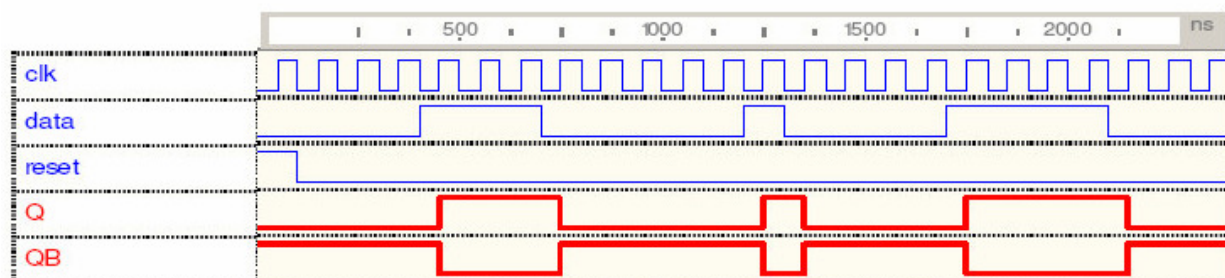
```
entity ic7474 is
port(d,clk,pr_1,clr_1:in std_logic;
q,qn:out std_logic);
end ic7474;
```

```
architecture Behavioral of ic7474 is
signal pr,clr:std_logic;
begin
process(clr_1,clr,pr_1,pr,clk)
begin
pr<=not pr_1;clr<=not clr_1;
if (clr and pr)='1' then q<='0';qn<='1';
elsif clr='1' then q<='0'; qn<='1';
elsif pr='1' then q<='1';qn<='0';
elsif (clk'event and clk='1') then
q<=d;
qn<=not d;
end if;
end process;
end Behavioral;
```

USER CONSTRAINTS

```
NET "clr_1" LOC = "p55" ;
NET "d" LOC = "p56" ;
NET "pr_1" LOC = "p57" ;
NET "q" LOC = "p36" ;
NET "qn" LOC = "p35" ;
net clk loc=p80;
```

Simulator Waveform





Design Module: J.K Flip-flop:

The race conditions in S-R Flip-flop can be eliminated by converting it in to J.K, the data inputs J and K are ANDed with Q' and Q to obtain S & R inputs. Here SR, T, or D depending on inputs

$$S=J.Q'$$

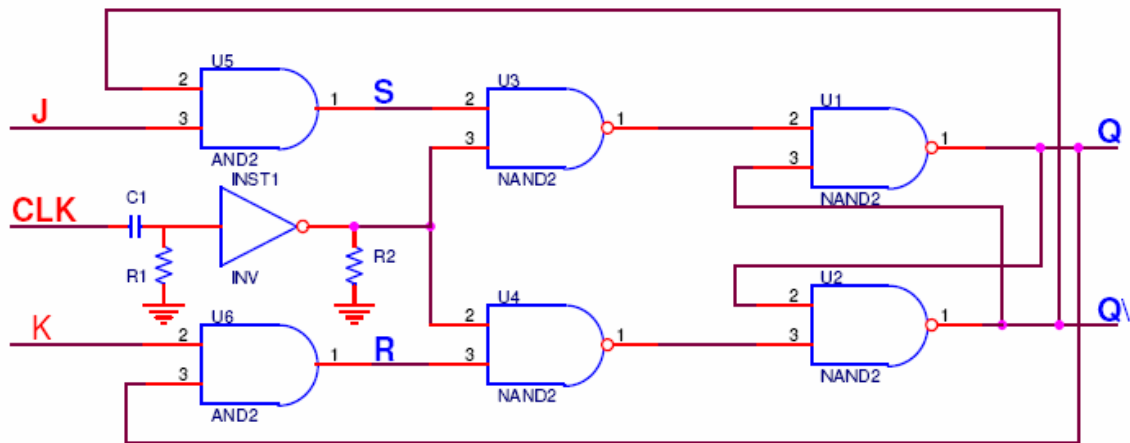
$$R=K.Q$$

Logic diagram:

JK-F/F Truth table

J	K	Q^+	Action
0	0	Q	No Change
0	1	0	Reset
1	0	1	Set
1	1	Q'	Toggle

Gate level Example:



+Ve edge triggered JK Flip-flop



VHDL Source Code:

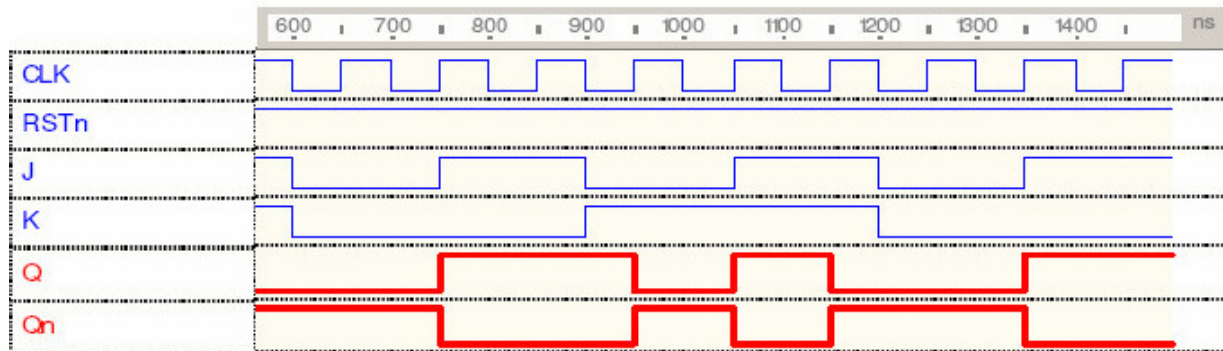
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity jkff is
port (CLK, RST, J, K : in std_logic;
Q, Qn: out std_logic);
end jkff;
architecture Behavioral of jkff is
signal FF : std_logic;
begin
process (CLK, RST)
variable JK : std_logic_vector(1 downto 0);
begin
if (RST = '0') then
FF <= '0';
elsif (CLK'event and CLK = '1') then
JK := J & K;
case JK is
when "01" => FF <= '0';
when "10" => FF <= '1';
when "11" => FF <= not FF;
when others => FF <= FF;
end case;
end if;
end process;
Q <= FF ;
Qn <= not FF ;
end Behavioral;
```

USER CONSTRAINTS

```
NET "J" LOC = "p55" ;
NET "K" LOC = "p56" ;
NET "Q" LOC = "p36" ;
NET "Qn" LOC = "p35" ;
net clk loc =p80;
net rst loc=p57;
```



Simulator Waveform:



Design Module: T-Flip-flop (Toggle Flip-flop):

On every change in clock pulse the output 'Q' changes its state (Toggle). A Flip-flop with one data input which changes state for every clock pulse. (J=K='1' in JQK Flip-flop the resulting output is 'T' Flip-flop).

T-F/F Truth table

T	Q ⁺	Action
0	Q	No Change
1	Q	Toggle

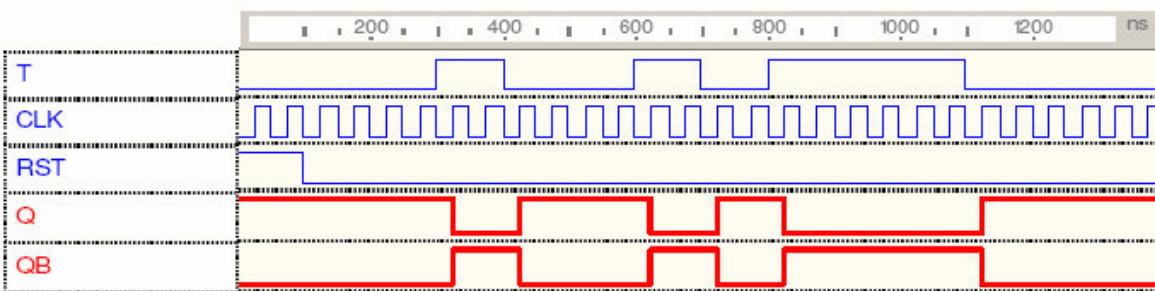
VHDL Source Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tff is
port (T, CLK, clr : in std_logic;
Q,QB : out std_logic);
end tff;
architecture Behavioral of tff is
begin
process (clk, clr) begin
if (clr = '1') then
Q <= '0';
QB <= '1';
elsif (clk'event and clk = '1') then
Q <= not T;
QB <= T;
end if;
```



```
end process;  
end Behavioral;  
USER CONSTRAINTS  
NET "clr" LOC = "p55" ;  
NET "Q" LOC = "p36" ;  
NET "QB" LOC = "p35" ;  
NET "T" LOC = "p56" ;  
net clk loc=P80'
```

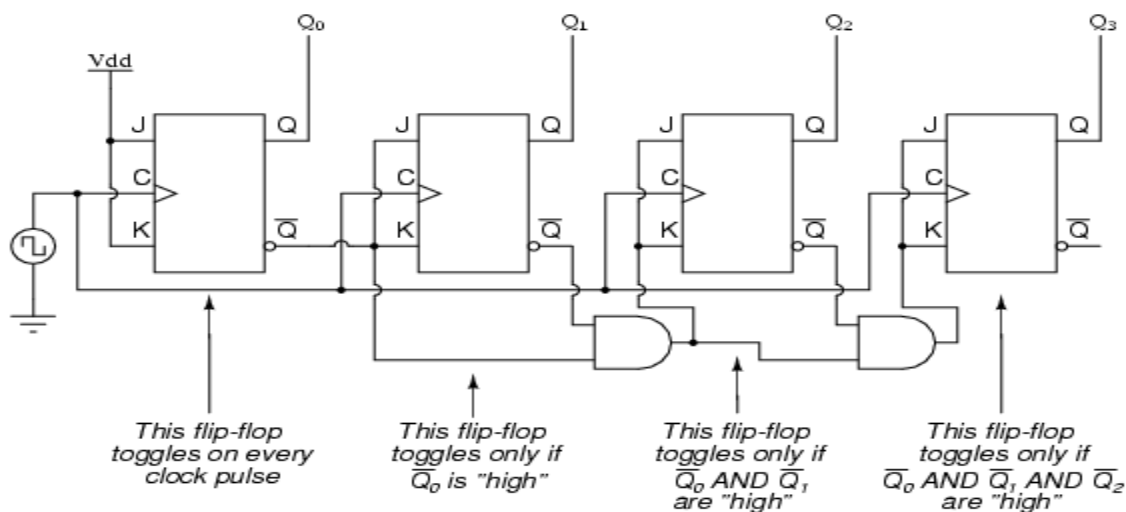
Simulator Waveform



Design Module: Counter

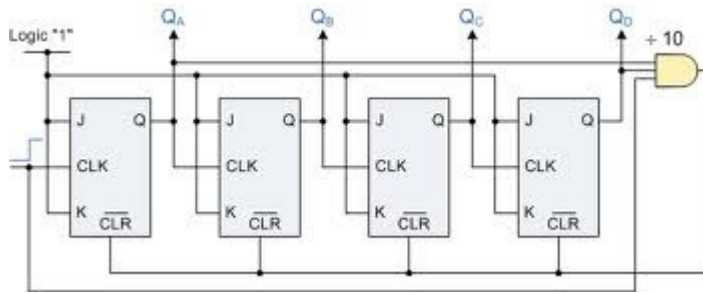
Counter is a digital circuit that can counts the number of pulses. For building the counters, Flip-flop are used. In Synchronous Counter, Clk is triggered by clk.

A four-bit synchronous "down" counter





Asynchronous Counter: In asynchronous counter clk is triggered by last output.



VHDL Source Code:

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

entity modn is
Port (CLK_IN : IN STD_LOGIC;
RST_IN : IN STD_LOGIC;
BCD_OUT : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
end modn;

architecture Behavioral of modn is
SIGNAL CLK_SIGNAL : STD_LOGIC_VECTOR(24 DOWNTO 0);
SIGNAL BCD_OUT_SIGNAL : STD_LOGIC_VECTOR(3 DOWNTO 0);
begin
PROCESS(CLK_IN,RST_IN)
BEGIN
IF(RST_IN='1')THEN
CLK_SIGNAL<=(OTHERS=>'0');
ELSIF(CLK_IN='1' AND CLK_IN'EVENT)THEN
CLK_SIGNAL<=CLK_SIGNAL+1;
END IF;
END PROCESS;
PROCESS(CLK_SIGNAL(24),RST_IN)
BEGIN
IF(RST_IN='1')THEN
BCD_OUT_SIGNAL<=(OTHERS=>'0');
ELSIF(CLK_SIGNAL(24)='1' AND CLK_SIGNAL(24)'EVENT)THEN
```

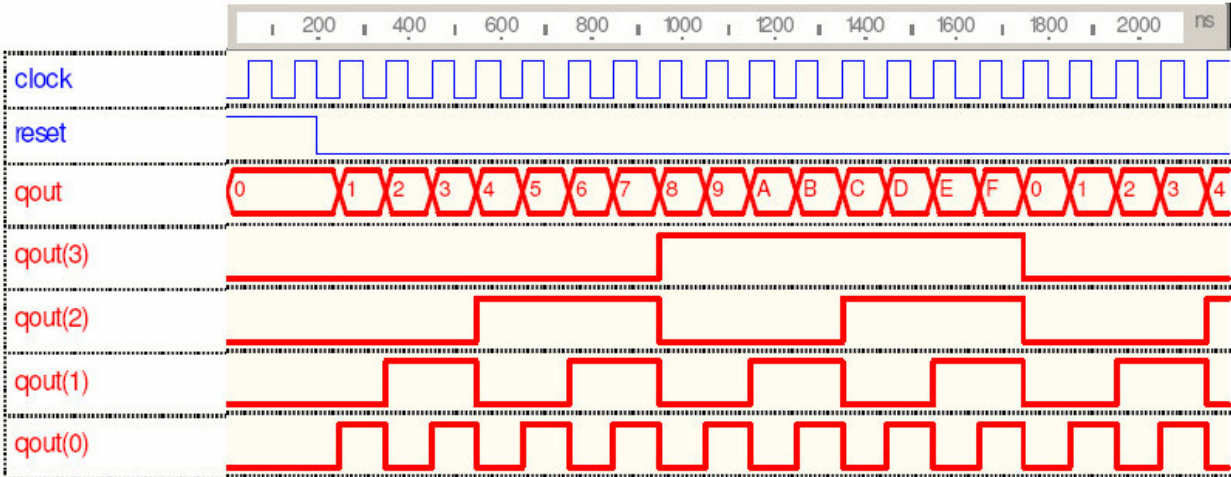


```
IF(BCD_OUT_SIGNAL="0000")THEN  
BCD_OUT_SIGNAL<="1111";  
ELSE  
BCD_OUT_SIGNAL<=BCD_OUT_SIGNAL-1;  
END IF;  
END IF;  
END PROCESS;  
BCD_OUT<=BCD_OUT_SIGNAL;  
end Behavioral;
```

USER CONSTRAINTS

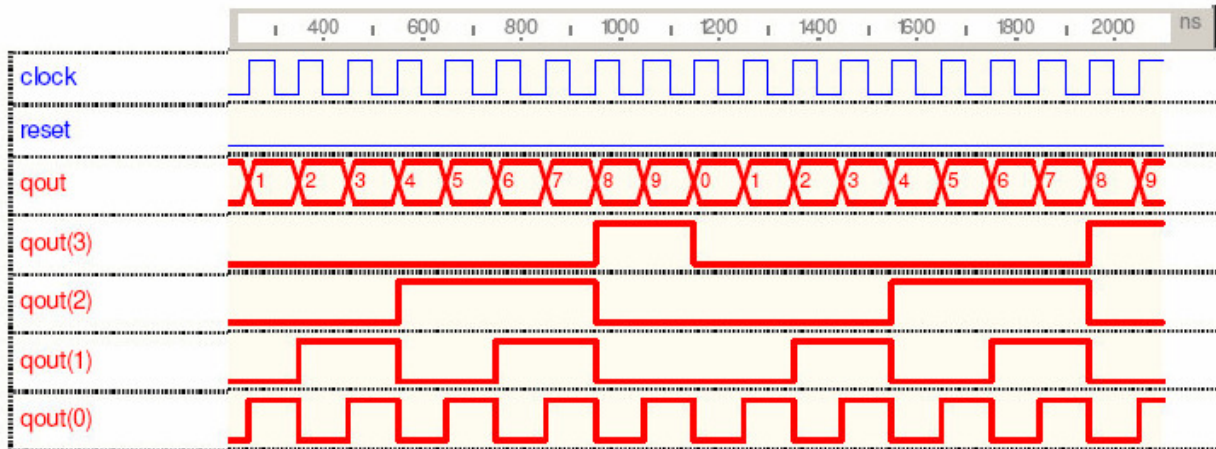
```
NET RST_IN LOC = P55;  
NET CLK_IN LOC = P80;  
NET BCD_OUT<0> LOC = P20;  
NET BCD_OUT<1> LOC = P21;  
NET BCD_OUT<2> LOC = P22;  
NET BCD_OUT<3> LOC = P23;
```

Simulator Waveform for 4 bit Binary Up Counter

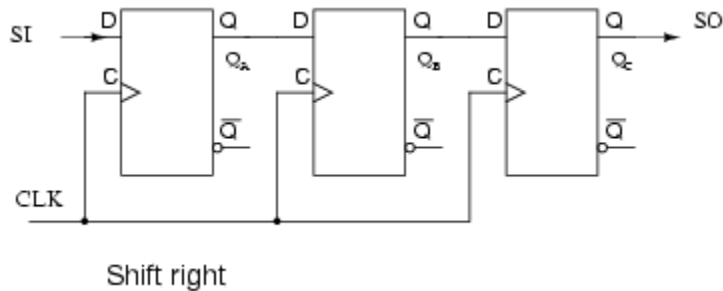




Simulator Waveform for 4 bit BCD UP Counter



Design Module: Shifter



VHDL Source Code:

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

entity rtshft is
PORT ( Resetn,w : IN STD_LOGIC ;
clock:in std_logic;
Q_out : buffer STD_LOGIC_VECTOR(1 TO 4) );
end rtshft;
```



```
architecture Behavioral of rtshft is
SIGNAL CLK_SIGNAL : STD_LOGIC_VECTOR(25 DOWNT0 0);
SIGNAL q_OUT_SIGNAL : STD_LOGIC_VECTOR(1 To 4);
begin
PROCESS(clock,Resetn)
BEGIN
IF(Resetn='1')THEN
CLK_SIGNAL<=(OTHERS=>'0');
ELSIF(Clock= '1' AND CLock'EVENT) THEN
CLK_SIGNAL<=CLK_SIGNAL+1;
END IF;
END PROCESS;
PROCESS ( Resetn, Clk_signal(25))
BEGIN
IF (Resetn = '1') THEN
Q_out_signal<=(OTHERS=>'0') ;
elsIF (Clk_signal(25)'event AND Clk_signal(25) = '1') THEN
Genbits: FOR i IN 4 DOWNT0 2 LOOP
Q_out_signal(i) <= Q_out(i-1) ;
END LOOP ;
Q_out_signal(1) <= w ;
END IF ;
END PROCESS ;
q_out<=q_out_signal;
end Behavioral;
```

USER CONSTRAINTS

```
NET "Q_out<1>" LOC = "p33" ;
NET "Q_out<2>" LOC = "p34" ;
NET "Q_out<3>" LOC = "p35" ;
NET "Q_out<4>" LOC = "p36" ;
NET "Resetn" LOC = "p55" ;
NET "w" LOC = "p56" ;
net clock loc=p80;
```




VHDL Source 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 univshtreg is
port (A: inout STD_LOGIC_vector(7 downto 0);--I/O left ABCD EFGH right
CLR: in STD_LOGIC;--ACTIVE LO)
OE1: in STD_LOGIC;--OUTPUT ENABLE ACTIVE LOW
OE2: in STD_LOGIC;--OUTPUT ENABLE ACTIVE LOW
S0: in STD_LOGIC;--FUNTION SELECT
S1: in STD_LOGIC;--FUNTION SELECT
CLK: in STD_LOGIC;--POSITIVE EDGE TRIGGER
SR: in STD_LOGIC;--SERIAL RIGHT INPUT
SL: in STD_LOGIC;--SERIAL LEFT INPUT
QA: out STD_LOGIC;--SERIAL OUTPUT
QH: out STD_LOGIC);--SERIAL OUTPUT);
end univshtreg;
architecture Behavioral of univshtreg is
signal OE,SX:std_logic;
signal SE:STD_LOGIC_vector(2 downto 0);
signal S:STD_LOGIC_vector(1 downto 0);
SIGNAL CLK_SIGNAL : STD_LOGIC_VECTOR(25 DOWNT0 0);
SIGNAL q_OUT_SIGNAL : STD_LOGIC_VECTOR(1 To 4);
begin
OE<=OE1 or OE2; --OE is oring of OE1 and OE2
S<=S1 & S0;--two bit
SE<=S & OE;--three bit
process(clk,CLR,SE)
variable x:STD_LOGIC_vector(7 downto 0);
variable y:STD_LOGIC_vector(7 downto 0);
begin
IF(clr='1')THEN
CLK_SIGNAL<=(OTHERS=>'0');
ELSif( Clk'EVENT AND Clk= '1' ) then
CLK_SIGNAL<=CLK_SIGNAL+1;
```



```
END IF;
END PROCESS;
process(clk_signal(25),CLR,SE)
variable x:STD_LOGIC_vector(7 downto 0);
variable y:STD_LOGIC_vector(7 downto 0);
begin
if(CLR='0') then
case SE is
when "000" => x:="00000000";
when "100" => x:="00000000";
when "010" => x:="00000000";
when others => x:=y;
end case;
elsif(SE="000") then
x:=y;
elsif(clk_signal(25)'event and clk_signal(25)='1') then
case SE is
when "010"=> x:=SR & y(7 downto 1);A<=x;
when "100"=> x:=y(6 downto 0) & SL;A<=x;
when "110"=> A<="ZZZZZZZZ";x:=A;
when "111"=> A<="ZZZZZZZZ";x:=A;
when others => x:=y;
end case;
end if;
end process;
QA<=A(0);
QH<=A(7);
end Behavioral;
```

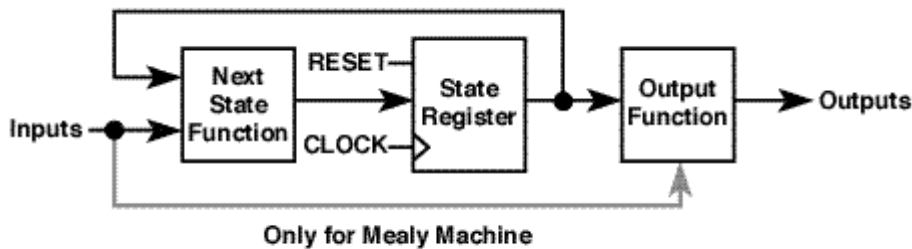
USER CONSTRAINTS

```
NET "A<0>" LOC = "p55" ;
NET "A<1>" LOC = "p56" ;
NET "A<2>" LOC = "p57" ;
NET "A<3>" LOC = "p58" ;
NET "A<4>" LOC = "p59" ;
NET "A<5>" LOC = "p60" ;
NET "A<6>" LOC = "p61" ;
NET "A<7>" LOC = "p62" ;
NET "CLR" LOC = "p95" ;
NET "OE1" LOC = "p96" ;
NET "OE2" LOC = "p97" ;
```



```
NET "QA" LOC = "p20" ;
NET "QH" LOC = "p21" ;
NET "S0" LOC = "p98" ;
NET "S1" LOC = "p99" ;
NET "SL" LOC = "p102" ;
NET "SR" LOC = "p101" ;
net clk loc=p80;
```

State Machine Design



X8993

VHDL Source 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;
--moore machine: outputs = f(present state)
-- moore machine changes its state first i.e wait a cloock cycle before the
--output values can be changed.
-- mealy machine is work one cycle in advance of moore machine.
-- it changes its output immediately when inputs changes.

entity MooreMachine_2p is
  Port ( clk : in  STD_LOGIC;
        in1 : in  STD_LOGIC_VECTOR(1 DOWNTO 0);
        reset : in  STD_LOGIC;
        out1 : out STD_LOGIC_vector(3 downto 0)
        );
end MooreMachine_2p;
```



architecture Behavioral of MooreMachine_2p is
type state_type is (s0,s1,s2,s3);----State declaration

signal state:state_type;

begin

process(clk,reset)---clocked process

begin

 if reset= '1' then

 state<= s0;-----reset state

 elsif clk'event and clk= '1' then

 case state is

 when s0=>

 if in1 = "00" then

 state<=s1;

 end if;

 when s1=>

 if in1 = "01" then

 state<=s2;

 end if;

 when s2=>

 if in1 = "10" then

 state<=s3;

 end if;

 when s3=>

 if in1 = "11" then

 state<=s0;

 end if;

 when others=> null;

 end case;

 end if;

end process;

output_p:process(state) ---- combinational process

begin

 case state is

 when s0=> out1<= "0000";

 when s1=> out1<= "1001";

 when s2=> out1<= "1100";

 when s3=> out1<= "1111";

 when others=> null;

 end case;

end process;

end Behavioral;