Laboratory Journal

of

DIGITAL SYSTEM DESIGN

For completion of term work of 6th *semester curriculum program*

Bachelor of Technology

In

ELECTRONICS AND TELECOMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION

ENGINEERING

Dr. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY

Lonere-402 103, Tal. Mangaon, Dist. Raigad (MS)

INDIA

LIST OF EXPERIMENTS

Sr. No.	Title
1	To study different scales of integration and advantages of VLSI
2	To study CMOS logic design
3	Implementation of logic gates using VHDL
4	Implementation of HA and FA
5	To study and design multiplexer and decoder
6	To study flip-flop
7	To design 8-bit counter(up-down) using VHDL
8	Study Experiment

EXPERIMENT NO:1

Aim: To study different scales of integration and advantages of VLSI.

Theory:

The no. of components fitted into standard size IC represents its integration scale in other words. It is density of components. It is classified as follows:

1. SSI(small scale integration):

   It have less than 100 components(about 10 gates).

2. MSI(medium scale integration):

   It contains less than 500 components(more than 10 but less than 100 gates).

3. LSI(large scale integration):

   It contains components between 500&30000(more than 100 gates).

4. VLSI(very large scale integration):

   The no. of components exceeds 30000(more than 1000 gates).

VLSI is the current level of computer microchip miniaturization and refers to microchip containing in hundred of thousand transistor. VLSI is the method of putting the functionality of many of different types of electronic component into small space or chip.

This method essentially:

1. Reduce size of device.
2. Reduce cost of device.
3. Reduce current consumption of device.
4. Increase speed of operation.
5. Offers lots of employment.

Problem:

Solve one problem of implementing 8:1 MUX using logic gates, 4:1 MUX and 8:1 MUX.

Conclusion:

EXPERIMENT NO: 2

Aim: To study CMOS logic design.

Theory:

A circuit build using opposite type MOSFET transistor in a CMOS circuit. The CMOS is a form of circuit popular in a diagram circuitry and uses both n-channel and p-channel enhancement MOSFET transistors. This circuit uses opposite type transistor.

N-MOS on/off operation-

An input of 0v leaves the N-MOS off where as input of +5v turns N-MOS on.

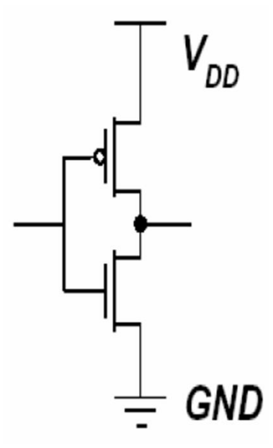
P-MOS on/off operation-

An input of 0v leaves the P-MOS on where as input of +5v turns P-MOS off.

The various circuits can be bulid up using CMOS. Also the basic universal gates can be implemented by CMOS.

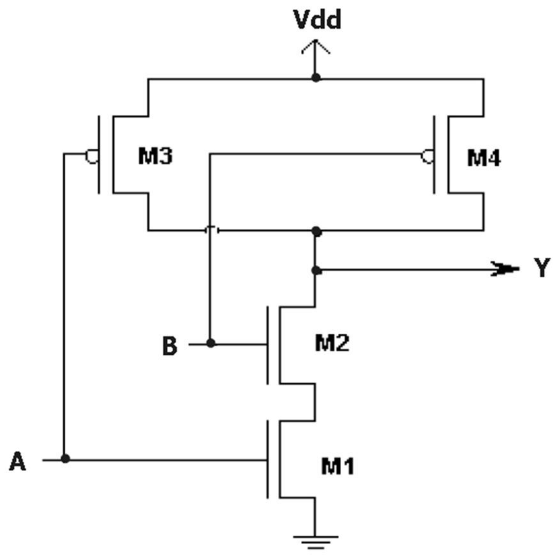
1. **NOT gate implemented using CMOS**

The implementation is as shown in figure.It requires two CMOS:1 PMOS & 1 NMOS.



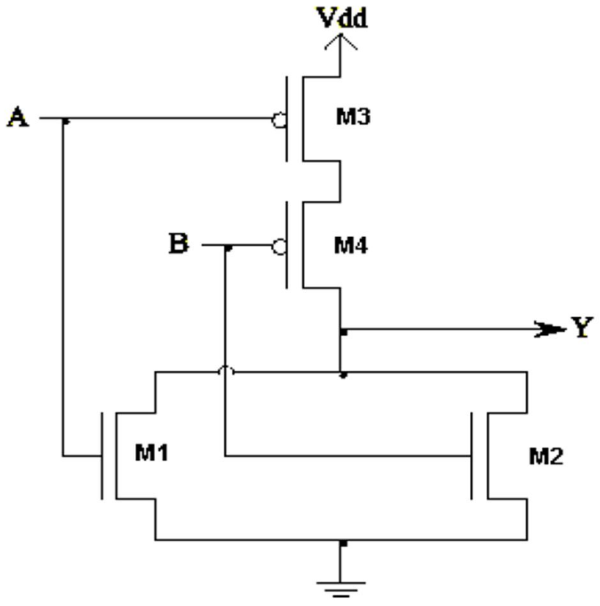
2. NAND gate implemented using CMOS

The implementation is as shown in figure. It requires four CMOS:2 PMOS & 2 NMOS.



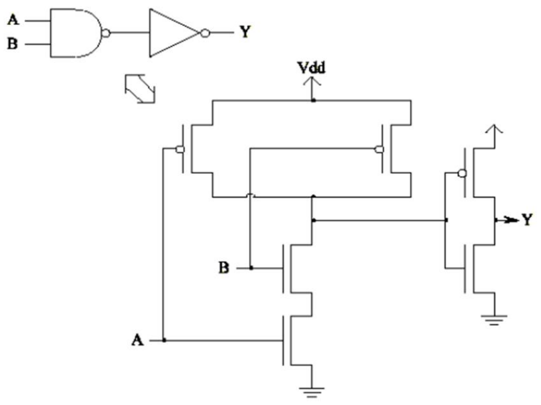
3. NOR gate implemented using CMOS

The implementation is as shown in figure.It requires four CMOS:2 PMOS & 2 NMOS.



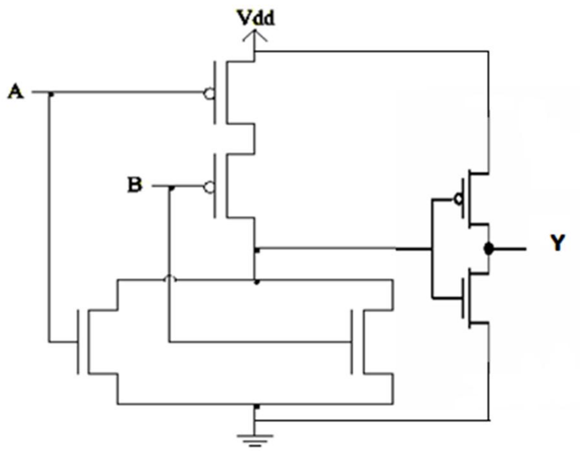
4. **AND gate implemented using CMOS**

The implementation is as shown in figure. The AND gate is implemented by adding an invertor circuit to the output of NAND gate. Thus it requires six CMOS:3 PMOS & 3 NMOS.



5. **OR gate implemented using CMOS**

The implementation is as shown in figure.The OR gate is implemented by adding an invertor circuit to the output of NOR implemented circuit.Thus it requires six CMOS:3 PMOS & 3 NMOS.



Conclusion:

EXPERIMENT NO: 3

Aim: Implementation of logic gates using VHDL.

1) using dataflow style.
2) using behavioural style.

Theory: Logic gates are classified into three categories.

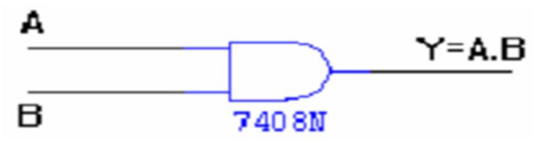
1. Basic gates
2. Universal gates
3. Special purpose gates.

1. Basic gates:

AND GATE:

The AND gate performs a logical multiplication commonly known as AND function. AND gate is a logical operator. The output is high when both the inputs are high. The output is low level when any one of the inputs is low.

Symbol:



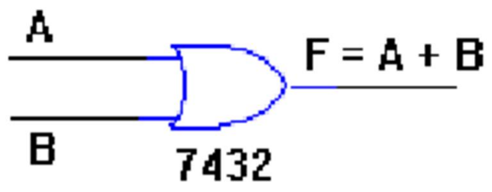
Truth table:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:

The OR gate performs a logical addition commonly known as OR function. The output is high when any one of the inputs is high. The output is low level when both the inputs are low.

Symbol:



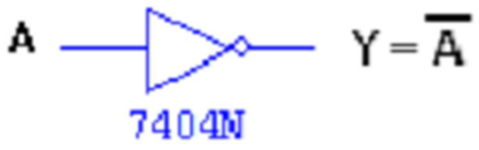
Truth table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:

The NOT gate is called an inverter. The output is high when the input is low. The output is low when the input is high.

Symbol:



Truth table:

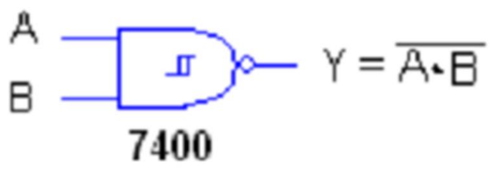
A	Y
0	1
1	0

2. Universal gates

NAND GATE:

The NAND gate is a contraction of AND-NOT. The output is high when both inputs are low and any one of the input is low .The output is low level when both inputs are high.

Symbol:



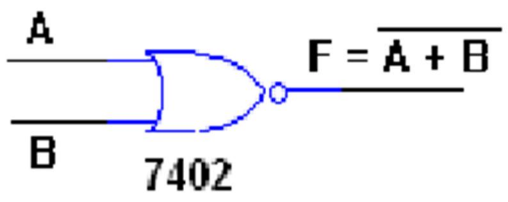
Truth table:

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE:

The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low. The output is low when one or both inputs are high.

Symbol:



Truth table:

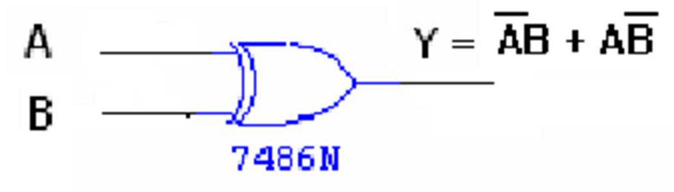
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

3. Special purpose gates

X-OR GATE:

The output is high when any one of the inputs is high. The output is low when both the inputs are low and both the inputs are high.

Symbol:



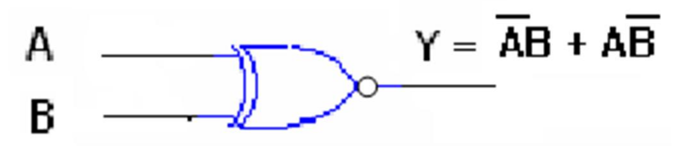
Truth table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

X-NOR GATE:

Output is high if both inputs are identical and output is low if input are not identical.

Symbol:



Truth table:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Conclusion:

Programs:

NOT Gate:

```
entity not_gate is
port (a :in STD_LOGIC;
        c: out STD_LOGIC);
end not_gate;
architecture dataflow of not_gate is
begin
c<= not a;
```

```
end dataflow;
```

AND Gate:

```
entity and_gate is
port (a, b :in STD_LOGIC;
        c: out STD_LOGIC);
end and_gate;
architecture dataflow of and_gate is
begin
c<= a and b;
end dataflow;
```

OR Gate:

```
entity or_gate is
port (a, b :in STD_LOGIC;
        c: out STD_LOGIC);
end or_gate;
architecture dataflow of or_gate is
begin
c<=  a or b;
end dataflow;
```

XOR Gate:

```
entity xor_gate is
port (a,b :in STD_LOGIC;
            c: out STD_LOGIC);
end xor_gate;
architecture dataflow of xor_gate is
begin
c<=  a xor b;
end dataflow;
```

XNOR Gate:

```
entity xnor_gate is
port (a,b :in STD_LOGIC;
            c: out STD_LOGIC);
end xnor_gate;
architecture dataflow of xnor_gate is
begin
c<=  a xnor b;
end dataflow;
```

NAND Gate:

Using dataflow:

```
entity nand_gate is
port (a,b :in STD_LOGIC;
            c: out STD_LOGIC);
end nand_gate;
architecture dataflow of nand_gate is
```

```
begin
c<= a nand b;
end dataflow;
```

Using behavioural:

```
entity nand_gate is
port (a,b :in STD_LOGIC;
        c: out STD_LOGIC);
end nand_gate;
architecture behavioural of nand_gate is
begin
c<= ‘0’ when a=’1’ and b=’1’
else ‘1’
end behavioural;
```

NOR Gate:

Using dataflow:

```
entity nor_gate is
port (a,b :in STD_LOGIC;
        c: out STD_LOGIC);
end nor_gate;
architecture dataflow of nor_gate is
begin
c<= a nor b;
end dataflow;
```

Using behavioural:

```
entity nor_gate is
port (a,b :in STD_LOGIC;
            c: out STD_LOGIC);
end nor_gate;
architecture behavioural of nor_gate is
begin
c<= ‘1’ when a=’0’ and b=’0’
else ‘0’
end behavioural;
```

EXPERIMENT NO: 4

Aim: Implementation of half adder and implementation of full adder using half adder using VHDL.

Theory:

Adders are used for most basic arithematic operation,addition of binary degits.

Four possible elementary operation.

1. 0+0=0
2. 1+0=1
3. 0+1=1
4. 1+1=0

Half adder:

A combinational circuit that performs the addition of two bits is called a half adder.

This circuit needs two binary inputs and two binary outputs.

The input variables designate the augend and addend bits and the output variables designate the augend and addend bits and the output variables produce the sum and carry.

Truth table:

INPUT		OUTPUT	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Simplified Boolean expression are:

$S=\bar{X}Y+X\bar{Y}$

$C=XY$

The half adder can be implemented with an X-OR and AND gate.

Full adder:

A combinational circuit that performs the addition of three bits(two significant bits and a previous carry) is called a full adder.

It consists of three inputs and two outputs.

Truth table

INPUT			OUTPUT	
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Simplified the sum of product expression are

$S=\bar{X}\bar{Y}Z+\bar{X}Y\bar{Z}+X\bar{Y}\bar{Z}+XYZ$

$C=XY+XZ+YZ$

A full adder can be implemented with two half adders and OR gate.

Conclusion:

Programs:

Half Adder:

```
entity ha is
port (a,b :in STD_LOGIC;
          s,c: out STD_LOGIC);
end ha;
architecture dataflow of ha is
begin
s<= a xor b;
c<= a and b;
end dataflow;
```

Full Adder:

Using Half Adder:

```
entity fa is
port (a,b,cin :in STD_LOGIC;
          s,cout: out STD_LOGIC);
end fa;
architecture structural of fa is
signal s1, s2,s3:std_logic;
component ha
port (a,b :in STD_LOGIC;
          s,c: out STD_LOGIC);
end component;
```

```
component or_gate

port(a,b : in STD_LOGIC;

            c: out STD_LOGIC);

end component;

begin

c1:ha

port map(a,b,s1,s2);

c2:ha

port map(s1,cin,s,s3);

c3:or_gate

port map(s2,s3,cout);

end structural;
```

EXPERIMENT NO:5

Aim: To study and design multiplexer and decoder

Theory:

Multiplexer is a combinational circuit and that has many i/p and select lines to select appropriate input to the output. For n select lines MUX has 2^n inputs and one output.

Mux 4:1

It has four inputs, two select lines and one output.

Truth table

S1	S2	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Where,

I0,I1,I2,I3 are input lines

S1, S2 are select lines

Y is output

MUX 8:1

It has three select lines and eight inputs and one output.

Truth table

S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

Decoder

A decoder is a combinational circuit that converts binary information from n input lines to a maximaum and of 2^n unique output lines. If the n bit decoded information has unused or don't care combinations, decoder output will have less than 2^n outputs.

As we know that sequential statement can be used for both combinational circuits as well as sequential circuits, we here have coded the decoder with the sequential statement.

Truth table:

Inputs			Outputs							
X2	X1	X0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

As we know, unlike MUX there is no select line in decoder. Thus output depends on input combinations along with En signal.

Conclusion:

Program:

MUX 4:1-

```
entity mux4x1 is
port( s: in STD_LOGIC_VECTOR (1 downto 0)
        x : in STD_LOGIC_VECTOR (3 downto 0)
         y : out STD_LOGIC);
end mux4x1;
architecture Behavioral of mux4x1 is
begin
with s select
y<= x(0) when “00”,
      x(1) when “01”,
      x(2) when “10”,
      x(3) when others;
end Behavioral;
```

MUX 8:1 using 4:1-

```
Entity mux8x1 is
Port( I : in STD_LOGIC_VECTOR (7 downto 0 );
        sel : in STD_LOGIC_VECTOR (2 downto 0);
         z : out STD_LOGIC );
end mux8x1;
architecture Behavioral of mux8x1 is
signal p1,p2 : std_logic;
```

```
component mux4x1
port( s: in STD_LOGIC_VECTOR (1 downto 0)
        x : in STD_LOGIC_VECTOR (3 downto 0)
         y : out STD_LOGIC);
end component;
begin
c1:mux4x1
port map (x(3)=>I(7), x(2)=>I(6), x(1)=>I(5), x(0)=>I(4),s(1)=>sel(1),s(0)=>sel(0),y=>p1);
c2: mux4x1
port map (x(3)=>I(3), x(2)=>I(2), x(1)=>I(1), x(0)=>I(0),s(1)=>sel(1),s(0)=>sel(0),y=>p2);
c2: mux4x1
port map (x(3)=>p1, x(2)=>'0', x(1)=>'0', x(0)=>p2,s(1)=>sel(2),s(0)=>sel(2),y=>z);
end Behavioral;
```

Decoder:

```
entity dec is
   Port ( EN : in  STD_LOGIC;
          X : in  STD_LOGIC_VECTOR (2 downto 0);
          Y : out  STD_LOGIC_VECTOR (7 downto 0));
end dec;

architecture Behavioral of dec is
begin
process(EN,X)
begin
if EN='1' then
```

```
          case X is
          when "000" =>Y<="00000001";
          when "001" =>Y<="00000010";
          when "010" =>Y<="00000100";
          when "011" =>Y<="00001000";
          when "100" =>Y<="00010000";
          when "101" =>Y<="00100000";
          when "110" =>Y<="01000000";
          when "111" =>Y<="10000000";
          when others =>Y<="ZZZZZZZZ";
          end case;
end if;
end process;

end Behavioral;
```

EXPERIMENT NO:6

Aim: To study flip-flop.

Theory:

Flip flops are basically the sequencial circuits i.e. they require clock for their operation. The flip flops are nothing but 1 bit storage elements. The different flip flops are D,T,SR, JK flip flop.

1. **D flip flop**
   The D flip flop is nothing but a latch. It just store the data it receives and on receiving a clock signal, it gives it as an output.

Truth Table

D	Q^+
0	0
1	1

Excitation Table

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

2. **T flip flop**
   The 'T' letter stands for 'toggling'. It toggles the i/p provided to it on receiving a clock signal.

Truth Table

T	Q^+
0	Q_n
1	$\overline{Q_n}$

Excitation Table

Q	Q^+	T
0	0	0
0	1	1
1	0	0
1	1	1

3. **SR flip flop**

Here SR stand for set and reset flip flop. In SR flip flop, a race around condition occurs on receiving both inputs high.

Truth Table

S	R	Q^+
0	0	Q
0	1	0
1	0	1
1	1	Race

Excitation Table

Q	Q^+	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

4. **JK flip flop**

JK flip flop has been designed in such a way that it overcomes the race around condition of the SR flip flop. It simply toggles the previous output when it receives both high input.

TruthTable

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\bar{Q}$

Excitation Table

Q	Q^+	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Conclusion:

Programs:

D flip flop:

```
entity dff is
   Port ( clk,rst,D : in  STD_LOGIC;
          Q : out  STD_LOGIC);
end dff;

architecture Behavioral of dff is


begin
process(clk,rst)
begin
          if rst='1' then
          Q<='0';
          elsif rising_edge(clk)then
          Q<=D;
          end if;
end process;
end Behavioral;
```

T flip flop:

```
entity tff is
   Port ( clk,rst,T : in  STD_LOGIC;
          Q : out  STD_LOGIC);
end tff;

architecture Behavioral of tff is
```

```
begin
  process(clk,rst,T )
  variable temp :STD_LOGIC;
            begin
             if rst='1' then
             Q<='0';
             temp:='0';
             elsif rising_edge(clk) then
                        if T='0' then
                                    temp:=temp;
                        else
                                    temp:= not temp;
                        end if;
            Q<=temp;
             end if;
  end process;
end Behavioral;
```

SR flip flop:

```
entity srff is
    Port ( clk,rst,s,r : in  STD_LOGIC;
           q : out  STD_LOGIC);
end srff;

architecture Behavioral of srff is
begin
process(clk,rst,s,r)
variable t:std_logic;
variable p:std_logic_vector(1 downto 0);
```

```
begin
p:=s&r;
if rst='0' then
        q<='0';
        t:='0';
elsif rising_edge(clk) then
case p is
        when "00"=>
                t:=t;
        when "01"=>
                t:='0';
        when "10"=>
                t:='1';
        when others=>
                t:=not t;
        end case;
        q<=t;
end if;
end process;
end Behavioral;
```

JK flip flop:

```
entity jkff is
   Port ( clk,rst : in  STD_LOGIC;
                jk :in  STD_LOGIC_vector(1 downto 0);
        Q : out  STD_LOGIC);
end jkff;
architecture Behavioral of jkff is
```

```
begin
 process(clk,rst,jk)
  variable temp :STD_LOGIC;
           begin
            if rst='1' then
            Q<='0';
            temp:='0';
            elsif rising_edge(clk) then
                       if jk="00" then
                       temp:=temp;
                       elsif jk="01" then
                       temp:='0';
                       elsif jk="10" then
                       temp:='1';
                       else
                       temp:= not temp;
                       end if;
    Q<=temp;
    end if;
  end process;
end Behavioral;
```

EXPERIMENT NO:7

Aim: To design binary up-down counter using VHDL.

Theory:

'Counter' as the name suggest are for counting a sequence of values. However, there are many different types of counters depending on the total no. of count values, the sequence of values that it outputs, whether it counts up or down and so on. The simplest is a modulo-n counter that counts the decimal sequence 0-1-2 upto n-1 back to 0.

1. Binary up counter:
   As the name suggest, the binary up counter simply counts the value from 0 to n-1 with the step size of 1. It is generally used for day to day purpose where counting objects is a task to evaluate.
2. Binary down counter:
   Like binary up counter, here too the name itself suggests the counting system of the binary down counter. This counter counts the values from n-1 down to 0 with step size of 1. It is generally used for gaming purpose where countdown is necessary.

Binary up-down counter:

The binary up-down counter is a combination of 'up counter' and 'down counter'. i.e. it counts upwardly and downwardly both depending on the select line i.e. mode selection.

Whenever mode select line='1', the counter act as a down counter and when select line='0', the counter act as a up counter.

Here a 8-bit counter is designed. Thus providing three bits as input along with a mode select 'm' input and also setting 3 bits as output. Depending on the value of 'm' the 3 bit counter will act as up or down counter.

Conclusion:

Program:

```
entity updncnt is
   Port ( clk,rst,m : in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (2 downto 0));
end updncnt;

architecture Behavioral of updncnt is
begin
process(clk,rst,m)
variable temp:std_logic_vector(2 downto 0);
begin
         if rst='1' then
         Q<="000";
         temp:="000";
         elsif rising_edge (clk) then
                  if m='0' then
                   temp:=temp+1;
                  else
                   temp:=temp-1;
                  end if;
                  Q<=temp;
         end if;
end process;
```