MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

EC3501 WIRELESS COMMUNICATION

Semester - 05

LABORATORY MANUAL

MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Vision

To excel in providing value based education in the field of Electronics and Communication Engineering, keeping in pace with the latest technical developments through commendable research, to raise the intellectual competence to match global standards and to make significant contributions to the society upholding the ethical standards.

Mission

- ✓ To deliver Quality Technical Education, with an equal emphasis on theoretical and practical aspects.
- ✓ To provide state of the art infrastructure for the students and faculty to upgrade their skills and knowledge.
- ✓ To create an open and conducive environment for faculty and students to carry out research and excel in their field of specialization.
- ✓ To focus especially on innovation and development of technologies that is sustainable and inclusive, and thus benefits all sections of the society.
- ✓ To establish a strong Industry Academic Collaboration for teaching and research, that could foster entrepreneurship and innovation in knowledge exchange.
- ✓ To produce quality Engineers who uphold and advance the integrity, honour and dignity of the engineering.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

1. To provide the students with a strong foundation in the required sciences in order to pursue studies in Electronics and Communication Engineering.
2. To gain adequate knowledge to become good professional in electronic and communication engineering associated industries, higher education and research.
3. To develop attitude in lifelong learning, applying and adapting new ideas and technologies as their field evolves.
4. To prepare students to critically analyze existing literature in an area of specialization and ethically develop innovative and research oriented methodologies to solve the problems identified.
5. To inculcate in the students a professional and ethical attitude and an ability to visualize the engineering issues in a broader social context.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Design, develop and analyze electronic systems through application of relevant electronics, mathematics and engineering principles.

PSO2: Design, develop and analyze communication systems through application of fundamentals from communication principles, signal processing, and RF System Design & Electromagnetics.

PSO3: Adapt to emerging electronics and communication technologies and develop innovative solutions for existing and newer problems.

LIST OF EXPERIMENTS:

1. Modeling of wireless communication systems using Matlab(Two ray channel and Okumura –Hata model)
2. Modeling and simulation of Multipath fading channel
3. Design, analyze and test Wireless standards and evaluate the performance measurements such as BER, PER, BLER, throughput, capacity, ACLR, EVM for 4G and 5G using Matlab
4. Modulation: Spread Spectrum – DSSS Modulation & Demodulation
5. Wireless Channel equalization: Zero-Forcing Equalizer (ZFE), MMSE Equalizer(MMSEE),Adaptive Equalizer (ADE),Decision Feedback Equalizer (DFE)
6. Modeling and simulation of TDMA, FDMA and CDMA for wireless communication

EX.NO :1 MODELLING OF WIRELSS COMMUNICATION SYSTEMS USING
DATE : MATLAB

a) Two ray channel

AIM:
To model and simulate wireless communication system for two ray channel and okumura hata model using matlab

Software Required:
Matlab version 2014a

THEORY:
The two-rays ground-reflection model is a multipath radio propagation model which predicts the path losses between a transmitting antenna and a receiving antenna when they are in line of sight (LOS). Generally, the two antenna each have different height. The received signal having two components, the LOS component and the reflection component formed predominantly by a single ground reflected wave. When the distance between antennas is less than the transmitting antenna height, two waves are added constructively to yield bigger power. As distance increases, these waves add up constructively and destructively, giving regions of up-fade and down-fade. As the distance increases beyond the critical distance or first Fresnel zone, the power drops proportionally to an inverse of fourth power of an approximation to critical distance may be obtained by setting $\Delta\varphi$ to π as the critical distance to a local maximum.

PROCEDURE:

1. Start the MATLAB program.

2. Open new M-file

3. Type the program

4. Save in current directory

5. Compile and Run the program

6. If any error occurs in the program correct the error and run it again

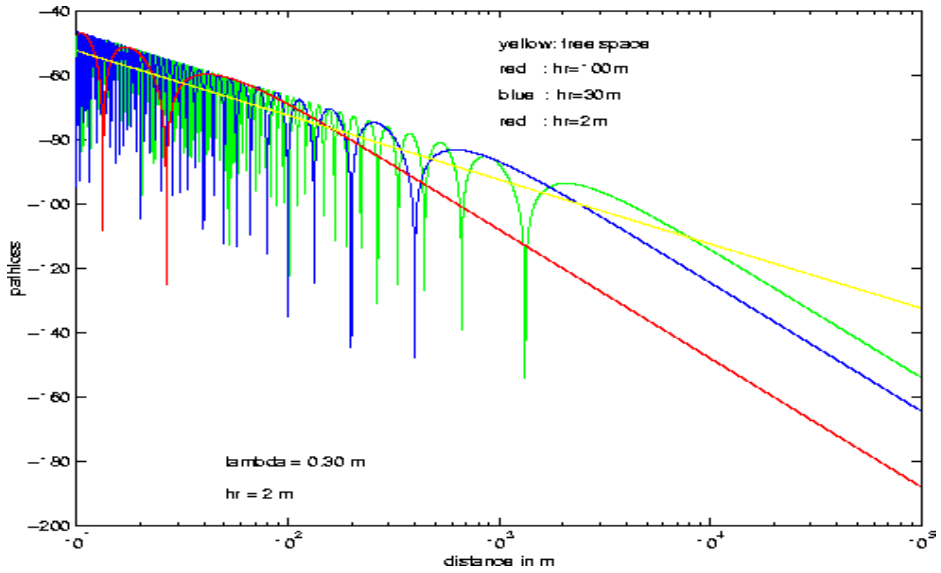
7. For the output see command window\ Figure window

8.Stop the program.

PROGRAM:

```
lambda = 0.3;
ht100=100;
ht30=30;
ht2=2;
hr=2;

axis=[];
p100=[];
p30=[];
p2=[];
pfsl=[];

for i=1000:5000
 d=10^(i/1000);
 axis =[axis d];
 fspower  = (lambda/(4*3.1415*d))^2 ;
 power100 = fspower * 4 *(sin(2*3.1415*hr*ht100/(lambda*d)))^2;
 power30 = fspower* 4 *(sin(2*3.1415*hr*ht30/(lambda*d)))^2;
 power2  = fspower * 4 *(sin(2*3.1415*hr*ht2/(lambda*d)))^2;

 p100 =[p100, 10*log10(power100)];
 p30 =[p30, 10*log10(power30)];
 p2 =[p2, 10*log10(power2)];
pfsl=[pfsl, 10*log10(fspower)];
end

text('FontSize',18)

semilogx(axis,p100, 'g-',axis,p30, 'b-',axis,p2, 'r-',axis,pfsl,'y-')


xlabel('distance in m');
ylabel('pathloss');
text(1000,-66,'blue  : hr=30m');
text(1000,-74,'red  : hr=2m');
text(1000,-58,'red : hr=100m');
text(1000,-50,'yellow: free space');

text(50,-180,'lambda = 0.30 m');
text(50,-190,'hr = 2 m');
```

OUTPUT:


yellow: free space
red : hr=100m
blue : hr=30m
red : hr=2m
lambda = 0.30 m
hr = 2 m
pathloss
distance in m

b) OKUMURA-HATA MODEL

PROGRAM

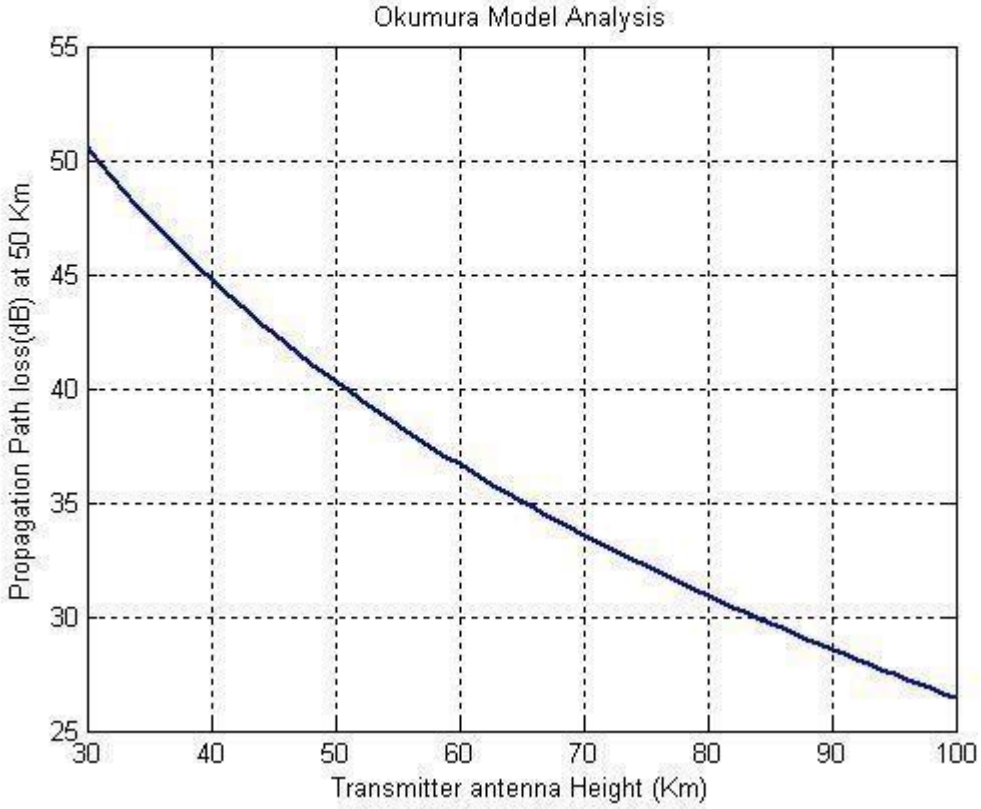
```
clc;
close all;
clear all;
hte=30:1:100;
hre=input('Enter the receiver antenna height 3m<hre<10m :');
d=input('Enter the distance from base station 1km<d<100km :');
f=input('Enter the frequency 150Mhz<f<1920Mhz :');
c=3*10^8;
lamda=c/(f*10^6);
lf=10*log((lamda^2)/((4*pi)^2));
amu=35;
garea=9;
ghte=20*log(hte/200);
if(hre>3)
ghre=20*log(hre/3);
else
ghre=10*log(hre/3);
end
l50=lf+amu-ghte-ghre-garea;
display('propagation path loss is: ');
disp(l50);
plot(hte,l50,'Linewidth',1.5);
title('okumura model analysis');
xlabel('transmitter antenna height (km)');
ylabel('propagation path loss(db) at 50 km');
grid on;
```

comparison between okumura and hata model:

Freq	Distance	$L_{(suburban)}$			$L_{(urban)}$		
		Okumura	Hata	$\Delta L_{(suburban)}$ (dB)	Okumura	Hata	$\Delta L_{(urban)}$ (dB)
150	50	-52.75	94.72	-147.47	-58.75	190.68	-249.43
	100	-38.89	107.64	-146.53	-44.89	203.60	-248.49
	150	-30.78	115.20	-145.98	-36.78	211.16	-247.94
	200	-25.02	120.56	-145.58	-31.02	216.52	-247.54
500	50	-76.83	93.11	-169.94	-82.83	203.68	-286.51
	100	-62.97	106.03	-169	-68.97	216.61	-285.58
	150	-54.86	113.59	-168.42	-60.86	224.17	-285.03
	200	-49.10	118.95	-168.05	-55.10	229.53	-284.63
900	50	-88.58	91.92	-180.5	-94.58	210.04	-304.62
	100	-74.72	104.85	-179.57	-80.72	222.96	-303.68
	150	-66.61	112.40	-179.01	-72.61	230.52	-303.13
	200	-60.86	117.77	-178.63	-66.86	235.88	-302.74
1300	50	-95.94	91.05	-186.99	-101.94	214.01	-315.95
	100	-82.08	103.97	-186.05	-88.08	226.93	-315.01
	150	-73.97	111.53	-185.5	-79.97	234.49	-314.46
	200	-68.21	116.90	-185.11	-74.21	239.85	-314.06

OUTPUT:

```
Enter the receiver antenna height 3m<hre<10m : 7
Enter distance from base station 1Km<d<100Km : 50
Enter the frequency 150Mhz<f<1920Mhz : 1000
Propagation pathloss is :
                          26.4575
```



RESULT:

Thus modeling and Simulation of wireless communication system for two ray channel and okumura - hata model was done using matlab.

EX.NO 2 MODELLING AND SIMULATION OF MULTIPATH FADING CHANNEL
DATE:

AIM:
To model and simulate multipath fading channel and display the spectral characteristics of the channel using Rayleigh and Rician multipath fading channel System objects

SOFTWARE REQUIRED:
Matlab version 2014

THEORY:
In wireless communication**,** fading is a phenomenon in which the strength and quality of a radio signal fluctuate over time and distance. Fading is caused by a variety of factors, including multipath propagation, atmospheric conditions, and the movement of objects in the transmission path. Fading can have a significant impact on the performance of wireless communication systems, particularly those that operate in high-frequency bands.

- Multipath delay spread is a type of small-scale fading that occurs when a transmitted signal takes multiple paths to reach the receiver.
- The different components of the signal can arrive at the receiver at different times, causing interference and rapid variations in signal amplitude and phase.
- Multipath delay spread can cause Inter-Symbol Interference (ISI), where symbols in the transmitted signal overlap and interfere with each other, leading to errors in the received signal.
- The root means square (RMS) delay spread is a measure of the dispersion of the signal and determines the frequency-selective characteristics of the channel.
- A higher RMS delay spread indicates a more frequency-selective channel, while a lower RMS delay spread indicates a flatter, more frequency-invariant channel.
- Multipath delay spread can be mitigated by using techniques such as equalization, diversity, and adaptive modulation.
- Equalization techniques are used to compensate for the time dispersion caused by multipath delay spread.
- Diversity techniques are used to combine multiple signal paths to mitigate the effects of fading.
- Adaptive modulation techniques are used to adjust the modulation scheme and data rate based on the channel conditions, allowing the system to adapt to changes in the channel and maintain a reliable communication link.

PROCEDURE:

1. Start the MATLAB program.

2. Open new M-file

3. Type the program

4. Save in current directory

5. Compile and Run the program

6. If any error occurs in the program correct the error and run it again

7. For the output see command window\ Figure window

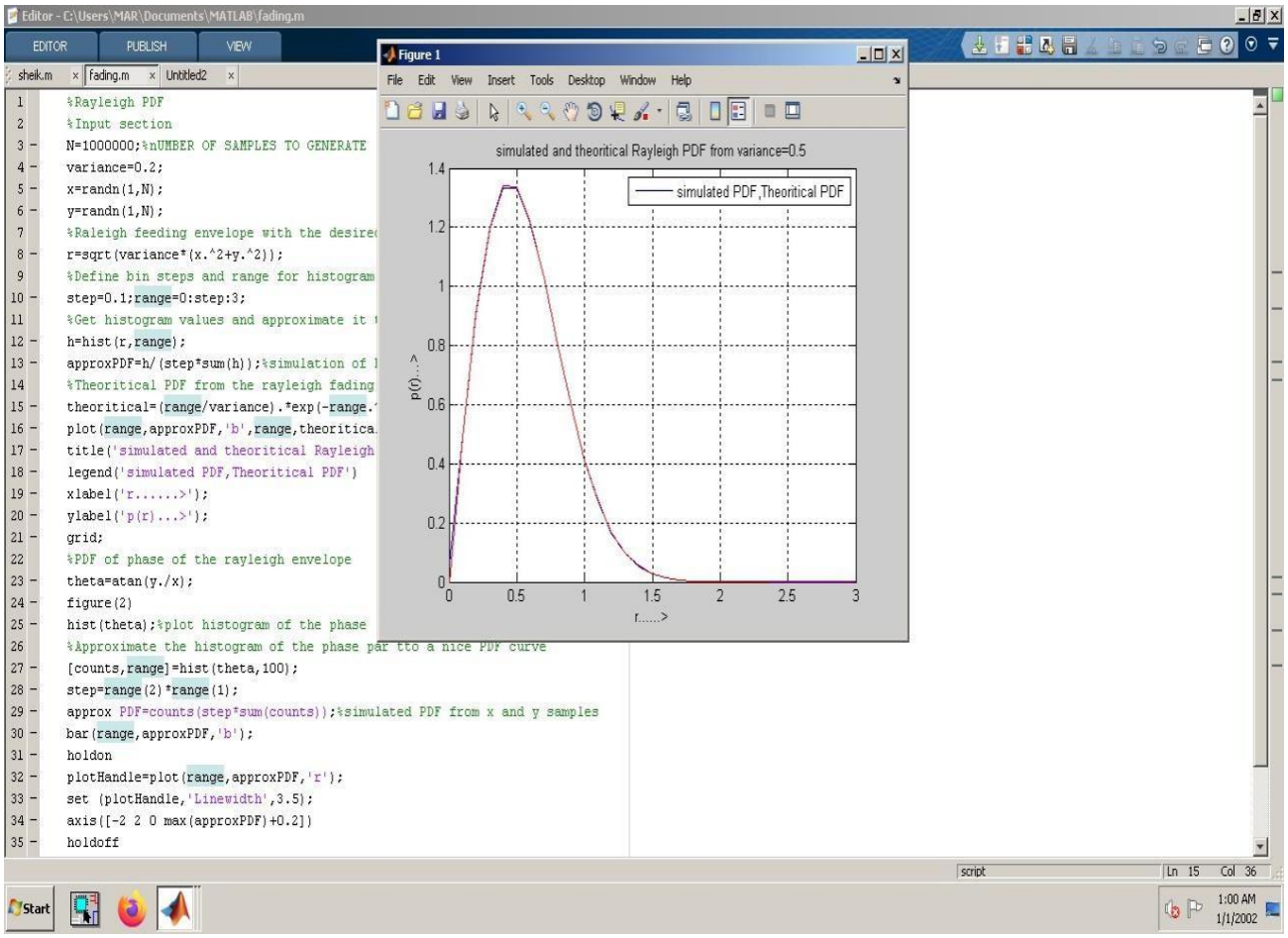
8.Stop the program.

PROGRAM:

```
%Rayleigh PDF
%Input section
N=1000000;%nUMBER OF SAMPLES TO GENERATE
variance=0.2;
x=randn(1,N);
y=randn(1,N);
%Raleigh feeding envelope with the desired variance
r=sqrt(variance*(x.^2+y.^2));
%Define bin steps and range for histogram plotting
step=0.1;range=0:step:3;
%Get histogram values and approximate it to get the pdf curve
h=hist(r,range);
approxPDF=h/(step*sum(h));%simulation of PDF from the x and y saqmples
%Theoritical PDF from the rayleigh fading equation
theoritical=(range/variance).*exp(-range.^2/(2*variance));
plot(range,approxPDF,'b',range,theoritical,'r');
title('simulated and theoritical Rayleigh PDF from variance=0.5')
legend('simulated PDF,Theoritical PDF')
xlabel('r..... >');
ylabel('p(r).. >');
grid;
%PDF of phase of the rayleigh envelope
theta=atan(y./x);
figure(2)
hist(theta);%plot histogram of the phase
%Approximate the histogram of the phase par tto a nice PDF curve
[counts,range]=hist(theta,100);
step=range(2)*range(1);
approx PDF=counts(step*sum(counts));%simulated PDF from x and y samples
bar(range,approxPDF,'b');
holdon
```

```
plotHandle=plot(range,approxPDF,'r');
set (plotHandle,'Linewidth',3.5);
axis([-2 2 0 max(approxPDF)+0.2])
holdoff
title('simulated PDF of phase of Rayleigh Distribution');
xlabel('theta.... >');
ylabel('p(\theta)...>');
grid;
```

OUTPUT:



Result:

Thus the modeling and simulation of multipath fading channel using Rayleigh and Rician channel system was simulated successfully.

EX.NO:5 WIRELESS CHANNEL EQUALIZATION

DATE: Zero-Forcing Equalizer (ZFE)

AIM:

To simulate Wireless Channel Equalization using MATLAB.

APPARATUS REQUIRED :

PC with MATLAB Software

THEORY:

If the modulation bandwidth exceeds the coherent bandwidth of the radio channel ISI occur and modulation pulses are spread in time. Equalization compensates for ISI(Inter symbol interference)created by multipath within time dispersive channels. An equalizer within a receiver compensates for the average range of expected channel amplitude and delay characteristics. Equalizers can be classified as linear equalizer, non linear equalizer, zero forcing equalizer, Adaptive equalizer,(MMSE)Minimum mean square error equalizer and decision feedback equalizer.In zero forcing equalization the equalizer g_k attempts to completely inverse the channel by forcing $c_k * g_k = \delta(k-ko)$.

PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window

8.Stop the program.

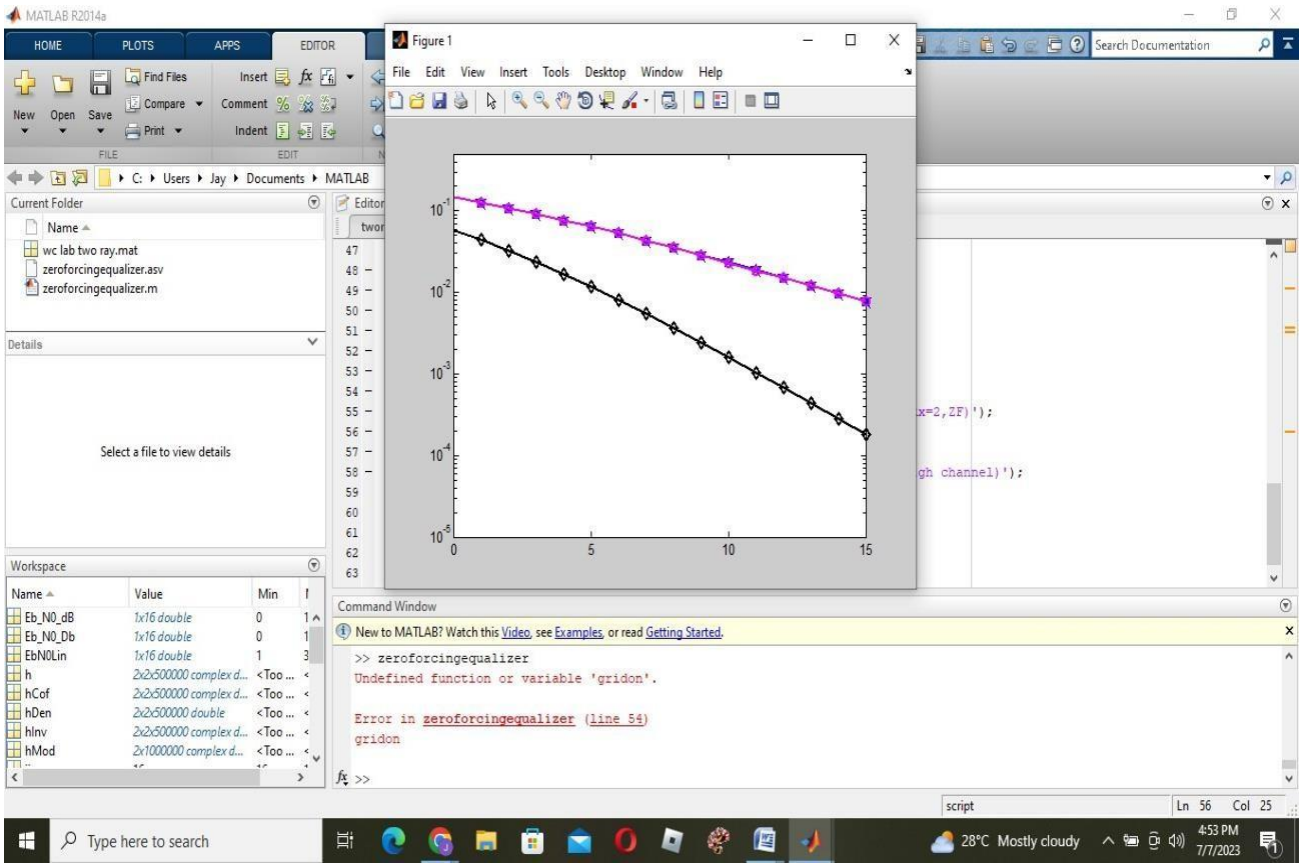
ALGORITHM

Simulate the link by following these steps:

1.Generate the number of bits and Eb/No value

2.Modulate BPSK Signal

ZERO FORCING EQUALIZER OUTPUT:



RESULT:

Thus channel equalization for wireless channel using zero forcing equalizer has been simulated using matlab.

EX.NO 4 MODULATION AND DEMODULATION OF DSSS
DATE:

AIM:
To modulate and demodulate Direct sequence spread spectrum using Matlab.

SOFTWARE REQUIRED:
Matlab version 2014

THEORY:
Direct-sequence spread spectrum in Wireless Networks is a technique that transmits a data signal over a range of frequencies, spreading it uniformly across the allocated spectrum. Direct-sequence spread spectrum is used to ensure that a particular frequency band (and its corresponding range of frequencies) is kept free from interference. This technique can be related to escaping the problem of co-channel interference (like two different wireless networks transmitting on the same frequency band) and cross-talk interference. Direct-sequence spread spectrum can also be used as an alternative approach to orthogonal frequency division multiplexing, where the baseband signal is encoded and transmitted across a quantity of fixed, predetermined channels. In this situation, each channel may carry different information, data signals, or time slots for different applications within the same network. Direct-sequence spread spectrum has also been used to transmit data that is encrypted and, in some processes, it is used to transmit non-data signals like power signaling or control signals.
Direct Sequence Spread Spectrum (DSSS) is a communication system that divides the bandwidth of a radio channel into wide frequency bands and transmits these signals over separate frequencies. In this frequency-hopping process, each signal is assigned a different orthogonal sequence of frequencies.All other radios in the range must gain each signal sequentially and then transmit it, which significantly reduces the risk of interference from outside sources or jamming. The time required for this process is proportional to the number of frequencies used for transmission. When security agencies need to be ready to communicate secretly, DSSS can be implemented so that their transmissions cannot be spied upon by other parties who are monitoring broadcasts on a shorter wavelength or through tapping devices.

PROCEDURE:

1. Start the MATLAB program.

2. Open new M-file

3. Type the program

4. Save in current directory

5. Compile and Run the program

6. If any error occurs in the program correct the error and run it again

7. For the output see command window\ Figure window

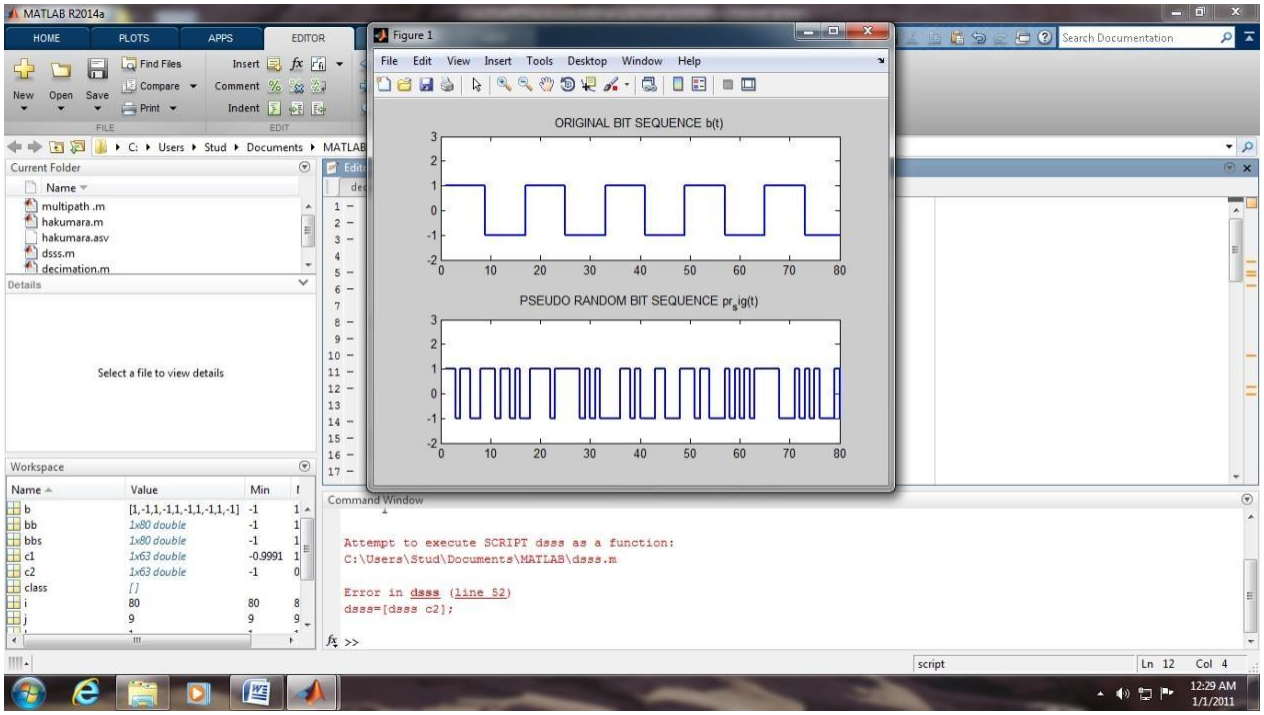
8.Stop the program.

PROGRAM:

```
Clc;
Close all;
Clear all;
%b=input('Enter the input bits:');
b =[1 0 1 0 1 0 1 0 1 0];
ln=length(b);
%converting bit 0 to -1
For i=1:in
If b(i) = = 0
b (i) = -1;
end
end
%Generating the bit sequence with each bit 8 samples long
K=1
for i=1:ln
for j=1:8
bb(k)=b(i);
j=j+1;
k=k+1;
end
i=i+1;
end
len=length(bb);
subplot(2,1,1);
stairs(bb,'Linewidth',2);
axis([0 len -2 3];
title('ORIGINAL BIT SEQUENCE b(t)');
%Generating the pseudo random bit pattern for spreading
pr_sig=round(rand(1,len));
for i=1:len
if pr_sig(i)==0
pr_sig(i)= -1;
end
```

```
end
subplot(2,1,2);
stairs(pr_sig,'Linewidth',2);
axis([0 len -2 3]);
title('PSEUDO RANDOM BIT SEQUENCE pr_sig(t)');
%Multiplying bit sequence with pseudo random sequence
for i=1:len
bbs(i)=bb(i).*pr_sig(i);
end
%Modulating the hopped signal
class=[];
t=0:1/10:2*pi;
c1=cos(t);
c2=cos(t+pi);
for k=1:len
if bbs(1,k)==-1
dsss=[dsss c1];
else
dsss=[dsss c2];
end
end
figure
subplot(2,1,1);
stairs(bbs,'Linewidth',2);
axis([0 len -2 3]);
title('MULTIPLIER OUTPUT SEQUENCE b(t)*pr_sig(t)');
subplot(2,1,2);
plot(dsss);
title('DS-SS SIGNAL….');
```

OUTPUT:



RESULT:

Thus modulation and demodulation of DSSS was done using matlab.

EX.NO 6 MODELING AND SIMULATION OF TDMA, FDMA AND CDMA FOR
DATE: WIRELESS COMMUNICATION

AIM:

To model and simulate TDMA,FDMA and CDMA accessing techniques in wireless communication using Matlab.

SOFTWARE REQUIRED:

Matlab version 2014

THEORY:

Access methods are multiplexing techniques that provide communications services to multiple users in a single-bandwidth wired or wireless medium. There are five basic access or multiplexing methods: frequency division multiple access (FDMA), time division multiple access (TDMA), code division multiple access (CDMA), orthogonal frequency division multiple access (OFDMA), and spatial division multiple access (SDMA). Each one of these takes advantage of multiplexing methods, dividing the bandwidth of the signal into different sub-bands, which are then assigned to different users in order to allow multiple users to share a single channel. Multiplexing is a communications technique that multiplexes, or combines, multiple signals into a single signal. The reverse process is called demultiplexing.

In FDMA, the time slots are assigned to the users in a sequential fashion. In TDMA, the time slots are assigned to the users in a random fashion. In CDMA, the time slots are assigned to the users based on their code sequences.

FDMA:

FDMA is the process of dividing one channel or bandwidth into multiple individual bands, each for use by a single user . Each individual band or channel is wide enough to accommodate the signal spectra of the transmissions to be propagated. The data to be transmitted is modulated on to each subcarrier, and all of them are linearly mixed together.

TDMA:

TDMA is a digital technique that divides a single channel or band into time slots. Each time slot is used to transmit one byte or another digital segment of each signal in sequential serial data format. This technique works well with slow voice data signals, but it’s also useful for compressed video and other high-speed data.

CDMA:

CDMA is another pure digital technique. It is also known as spread spectrum because it takes the digitized version of an analog signal and spreads it out over a wider bandwidth at a lower power level. This method is called direct sequence spread spectrum (DSSS) as well *as* The digitized and

compressed voice signal in serial data form is spread by processing it in an XOR circuit along with a chipping signal at a much higher frequency.

PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM:

TDMA

```
clc
    clear all;
    close all;
    num_node=input('Enter number of node for network=');
    No_of_time_slots=num_node;
    Bandwidth=200; %Khz
    time_slots_length=input('Enter the length of time slot');%seconds
    Guard_interval=input('Enter the value of guard interval');
    for i=1:1:No_of_time_slots
        pause(time_slots_length);
        a(i)=time_slots_length;
    end
  endclc;
```

FDMA

```
fs=2000;
fm1=4;
fm2=5;
fm3=6;
fcm1=25;
fcm2=50;
fcm3=75;

tiv=1/fs;
t=0:tiv:1;
A=2;
mu=0;
sigma=10;
m1=A*cos(2*pi*fm1*t);
sound(m1,fs);
pause(5);
len=length(m1);
y=lognpdf(mu,sigma);
m1=m1+y';

m2=2*A*cos(2*pi*fm2*t);
m3=3*A*cos(2*pi*fm3*t);

c1=A*cos(2*pi*fcm1*t

c2=A*cos(2*pi*fcm2*t);
c3=A*cos(2*pi*fcm3*t);

x=m1.*c1+m2.*c2+m3.*c3;
x=awgn(x,.02);%adding awgn in channel

[num1,den1]=butter(5,[.5*(fcm1-
fm1),fcm1+fm1]*4/fs);
[num2,den2]=butter(5,[.5*(fcm2-
fm2),fcm2+fm2]*4/fs);
[num3,den3]=butter(5,[.5*(fcm3-
fm3),fcm3+fm3]*4/fs);


filtr1=filter(num1,den1,x);
filtr2=filter(num2,den2,x);
filtr3=filter(num3,den3,x);

lp1=filtr1.*c1;
lp2=filtr2.*c2;
lp3=filtr3.*c3;
```

```
[num11,den11]=butter(5,4*fm1/fs);
[num22,den22]=butter(5,4*fm2/fs);
[num33,den33]=butter(5,4*fm3/fs);

lpf_out1=filter(num11,den11,lp1);

lpf_out2=filter(num22,den22,lp2);
lpf_out3=filter(num33,den33,lp3);

subplot(3,3,1);
plot(t,m1);
title('messege signal 1');
grid on;

subplot(3,3,2);
plot(t,m2);title('messege signal 2');
grid on;

subplot(3,3,3);
plot(t,m3);title('messege signal 3');
grid on;

subplot(3,3,4);
plot(t,c1);title('carrier signal 1');
grid on;

subplot(3,3,5);
plot(t,c2);title('carrier signal 2');
grid on;

subplot(3,3,6);
plot(t,c3);title('carrier signal 3');
grid on;
```

OUTPUT:

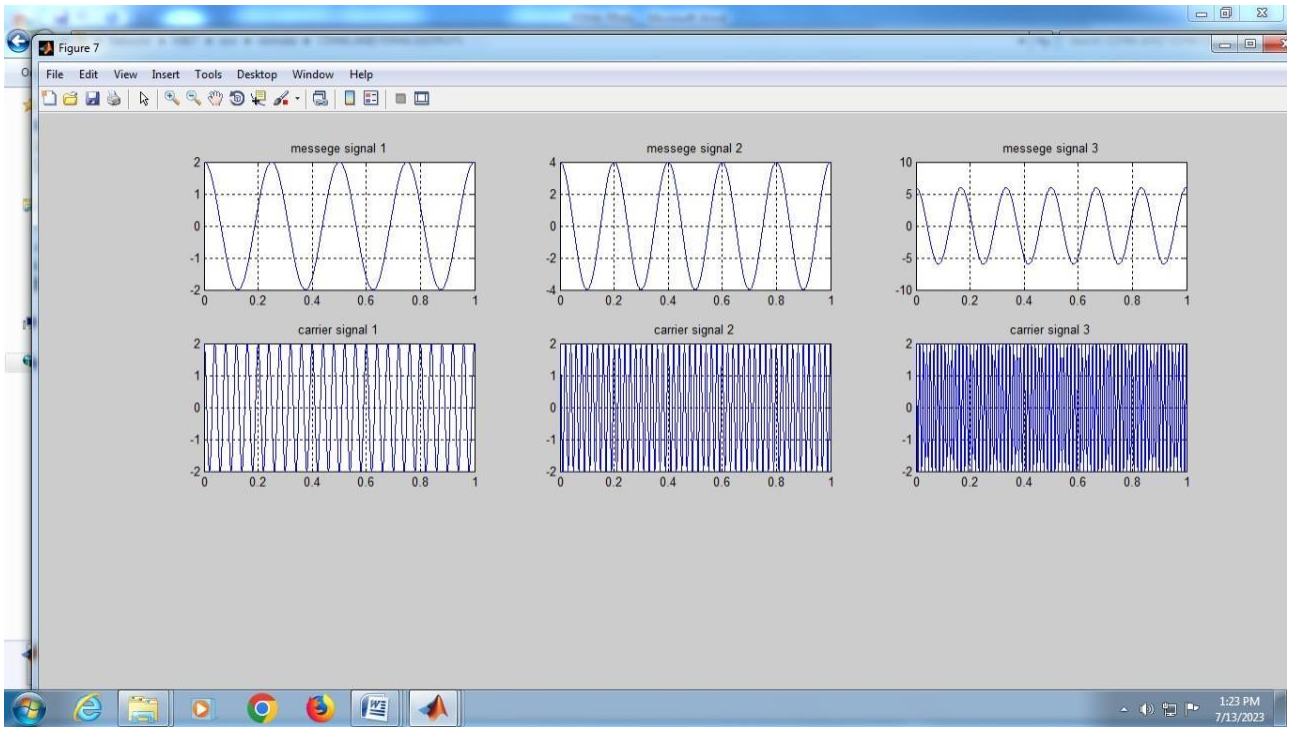


Figure 7
File Edit View Insert Tools Desktop Window Help
messege signal 1
messege signal 2
messege signal 3
carrier signal 1
carrier signal 2
carrier signal 3
1:23 PM
7/13/2023

CDMA:

```
close all;
clc;
%N=6;
%data=input('Enter Binary Data In Form of 0 and 1 in [ ] : ');
data=[1 0 0 1 1 0 1 1];
figure('Name','Message BPSK Modulation','NumberTitle','off')
subplot(2,2,1);
plot(rectpulse(data,100));
axis([0 length(rectpulse(data,100)) -0.2 1.2]);
title('Message Signal');
xlabel('n');
ylabel('x(n)');
grid on;
data(data(:)==0)=-1;
length_data=length(data);
fc1=10;
eb=2;
tb=1;
T=1;
msg=rectpulse(data,100);
subplot(2,2,2);
plot(msg);
title('Message Signal in NRZ form');
xlabel('n');
ylabel('x(n)');
```

```
axis([0 100*length_data -1.2 1.2]);
grid on;
N=length_data;
Tb = 0.0001;
nb=100;
br = 1/Tb;
Fc = br*10;
t2 = Tb/nb:Tb/nb:Tb;
t1=0.01:0.01:length_data;
bpskmod=sqrt(2/T)*cos(2*pi*fc1*t1);
bpsk_data=msg.*bpskmod;
subplot(2,2,3)
plot(bpsk_data)
title(' BPSK signal');
xlabel('Time Period(t)');
ylabel('x(t)');
axis([0 100*length_data -2 2]);
grid on;
subplot(2,2,4);
plot(real(fft(bpsk_data)));
title('FFT of BPSK signal');
xlabel('Frequency');
ylabel('PSD');
grid on;
sr=[1 -1 1 -1];
pn1=[];
```

```
for i=1:length_data
    for j=1:10
        pn1=[pn1 sr(4)];
        if sr (4)==sr(3)
        temp=-1;
        else temp=1;
        end
            sr(4)=sr(3);
            sr(3)=sr(2);
            sr(2)=sr(1);
            sr(1)=temp;
    end
end
figure('Name','PN Generation and CDMA','NumberTitle','off');
 subplot(2,2,1);
stem(pn1);
axis([0,length(pn1),-1.2,1.2])
title('PN sequence for data')
xlabel('n');
ylabel('x(n)');
grid on;
pnupsampled1=[];
len_pn1=length(pn1);
for i=1:len_pn1
    for j=0.1:0.1:tb
    pnupsampled1=[pnupsampled1 pn1(i)];
```

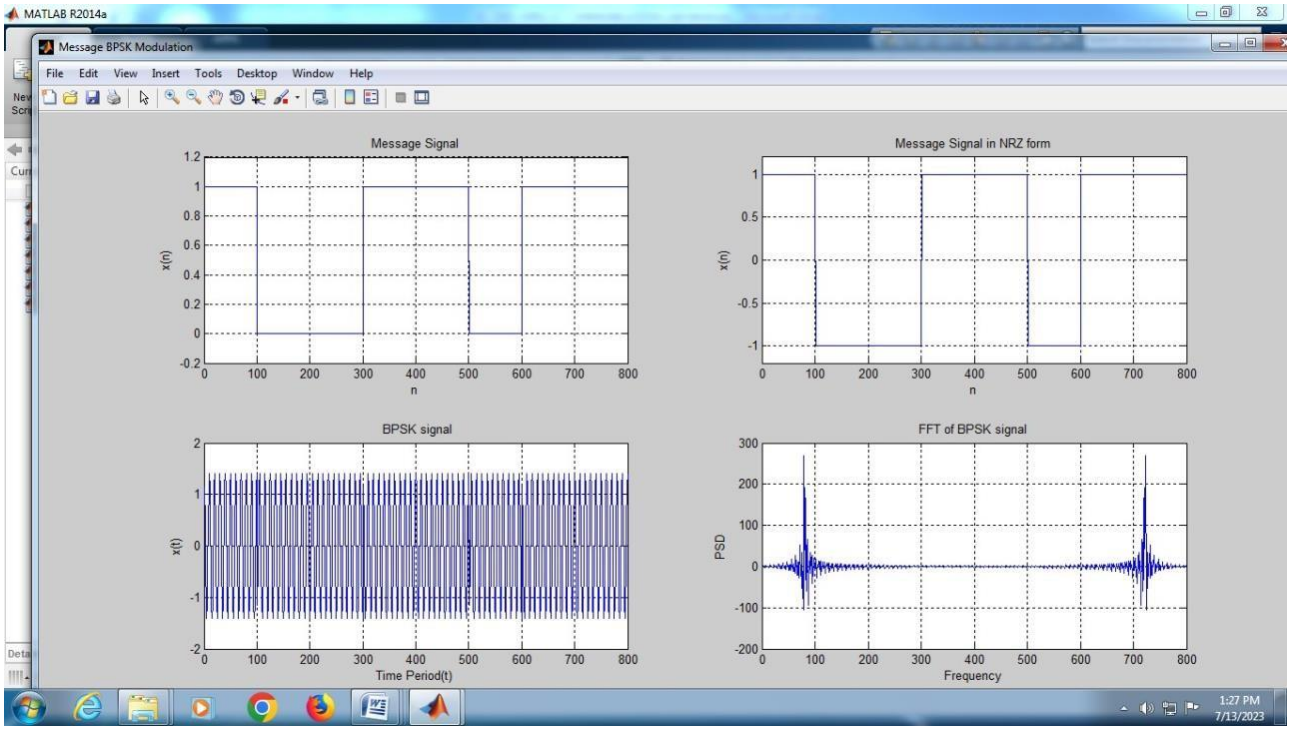
```
    end
end
length_pnupsampled1=length(pnupsampled1);
subplot(2,2,2)
stem(pnupsampled1);
axis([0,length(pnupsampled1),-1.2,1.2])
title('PN sequence for data upsampled');
xlabel('n');
ylabel('x(n)');
grid on;
subplot(2,2,3);
sigtx1=bpsk_data.*pnupsampled1;
plot(sigtx1);
title('CDMA Signal');
xlabel('Time Period(t)');
ylabel('x(t)');
subplot(2,2,4);
plot(real(fft(sigtx1)));
title('FFT of spreaded CDMA Signal');
xlabel('Frequency');
ylabel('PSD');
grid on;
sigtonoise=20;
composite_signal=awgn(sigtx1,sigtonoise);
figure('Name','CDMA Reciever','NumberTitle','off')
subplot(2,2,1);
```

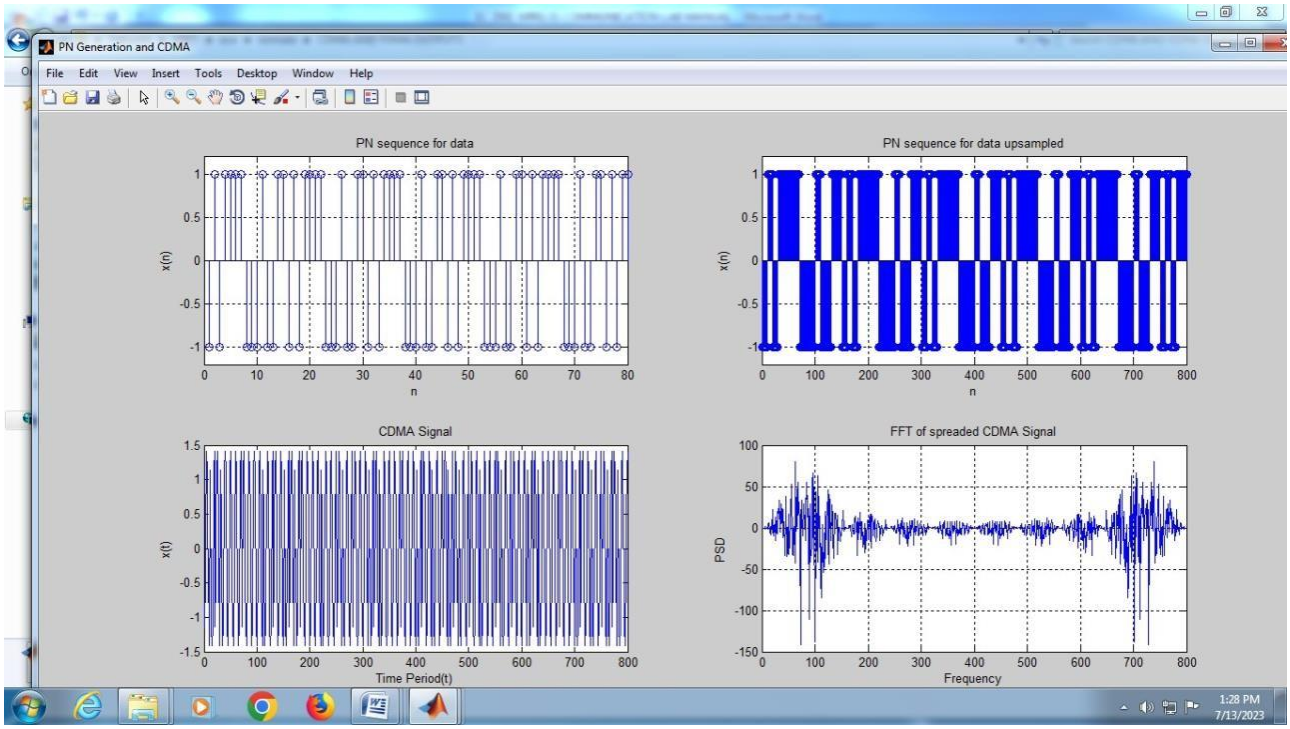
```
plot(sigtx1);
title(' Tx Signal');
xlabel('Time Period(t)');
ylabel('x(t)');
grid on;
subplot(2,2,2);
plot(composite_signal);
title(sprintf('Tx signal + noise\n SNR=%ddb',sigtonoise));
xlabel('Time Period(t)');
ylabel('x(t)');
grid on;

%Rx
rx=composite_signal.*pnupsampled1;
subplot(2,2,3);
plot(rx);
title('CDMA Demodulated signal');
xlabel('Time Period(t)');
ylabel('x(t)');
grid on;
%BPSK Demodulation
y=[];
bpskdemod=rx.*bpskmod;
for i=1:100:size(bpskdemod,2)
    y=[y trapz(t1(i:i+99),bpskdemod(i:i+99))];
end
```

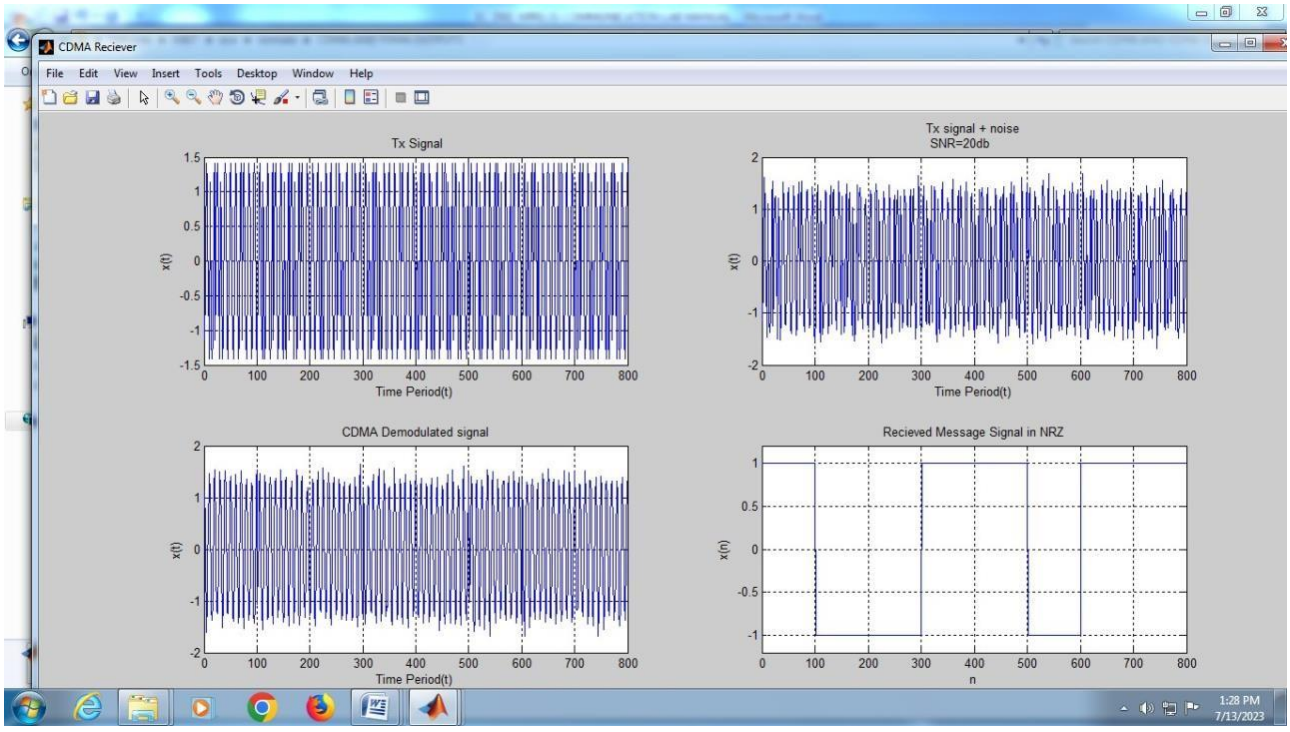
```
  y(y(:)<=0)=-1;
   y(y(:)>=0)=1;
  rxdata=y;
subplot(2,2,4);
plot(rectpulse(rxdata,100));
axis([0 length(rectpulse(rxdata,100)) -1.2 1.2]);
title('Recieved Message Signal in NRZ');
xlabel('n');
ylabel('x(n)');
grid on;
rxdata(rxdata(:)==-1)=0;
rxdata(rxdata(:)==1)=1;
rxmsg=rxdata;
 figure('Name','Diffrent SNR','NumberTitle','off')
 subplot(3,1,1)
plot(rectpulse(rxmsg,100));
axis([0 length(rectpulse(rxmsg,100)) -0.2 1.2]);
title('Recieved Message Signal');
xlabel('n');
ylabel('x(n)');
grid on;
sigtonoise1=5;
composite_signal1=awgn(sigtx1,sigtonoise1);
subplot(3,1,2);
plot(composite_signal);
title(sprintf('Tx signal + noise\n SNR=%ddb',sigtonoise1));
```

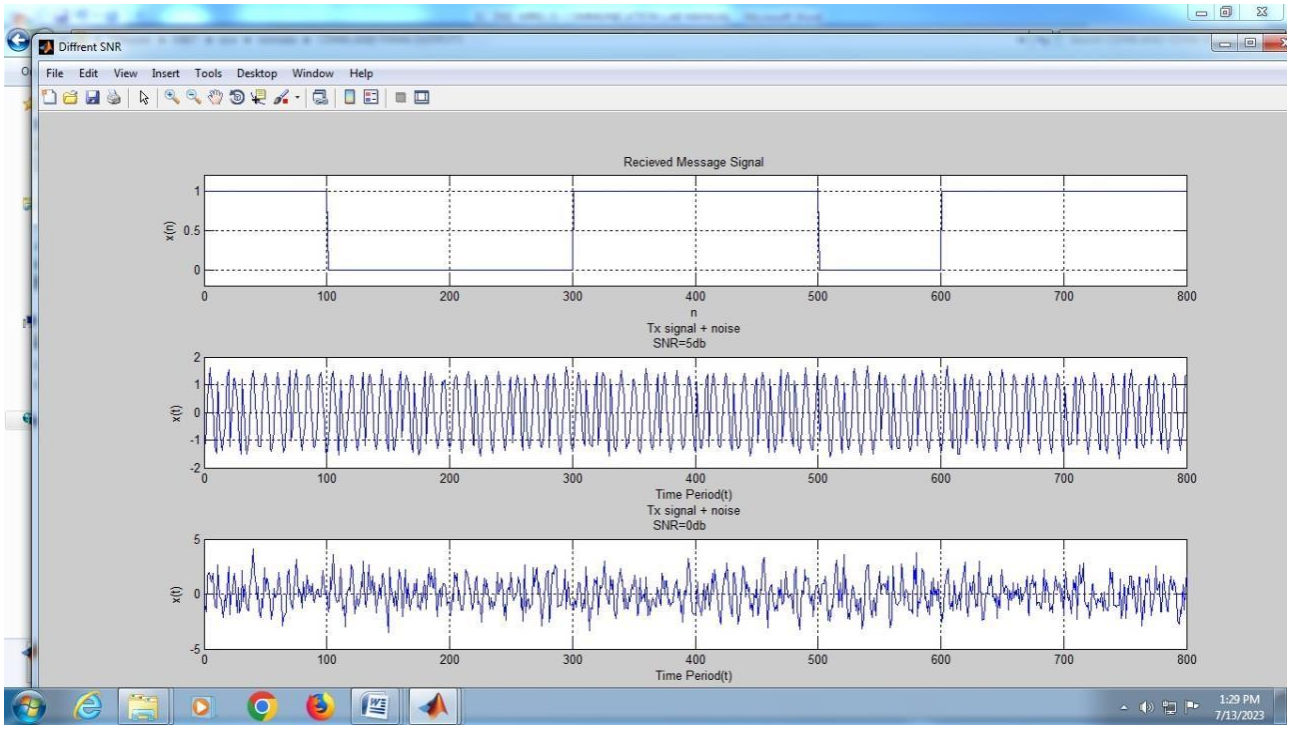
```
xlabel('Time Period(t)');
ylabel('x(t)');
grid on;
sigtonoise2=0;
composite_signal2=awgn(sigtx1,sigtonoise2);
subplot(3,1,3);
plot(composite_signal2);
title(sprintf('Tx signal + noise\n SNR=%ddb',sigtonoise2));
xlabel('Time Period(t)');
ylabel('x(t)');
grid on;
scatterplot(composite_signal); grid minor;
title('Constellation Diagram of BPSK with Noise')
grid on
scatterplot(bpsk_data); grid minor;
title('Constellation Diagram of BPSK')
grid on
```

OUTPUT:


MATLAB R2014a
Message BPSK Modulation
File Edit View Insert Tools Desktop Window Help
Message Signal
x(n)
n
Message Signal in NRZ form
x(n)
n
BPSK signal
x(t)
Time Period(t)
FFT of BPSK signal
PSD
Frequency
1:27 PM
7/13/2023


PN Generation and CDMA
File Edit View Insert Tools Desktop Window Help
PN sequence for data
x(n)
n
PN sequence for data upsampled
x(n)
n
CDMA Signal
x(t)
Time Period(t)
FFT of spreaded CDMA Signal
PSD
Frequency
1:28 PM
7/13/2023


CDMA Reciever
File Edit View Insert Tools Desktop Window Help
Tx Signal
x(t)
Time Period(t)
Tx signal + noise
SNR=20db
x(t)
Time Period(t)
CDMA Demodulated signal
x(t)
Time Period(t)
Recieved Message Signal in NRZ
x(n)
n
1:28 PM
7/13/2023


Diffrent SNR
File Edit View Insert Tools Desktop Window Help
Recieved Message Signal
x(n)
n
Tx signal + noise
SNR=5db
x(t)
Time Period(t)
Tx signal + noise
SNR=0db
x(t)
Time Period(t)
1:29 PM
7/13/2023

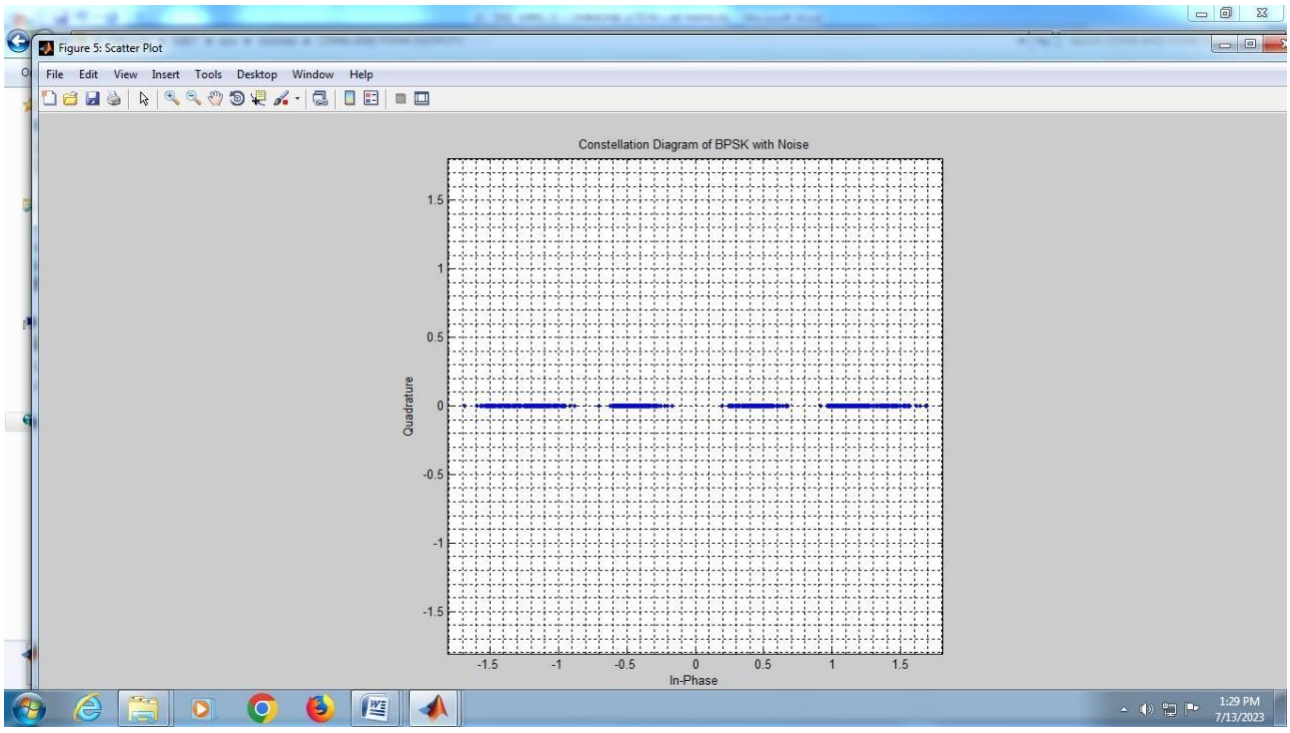
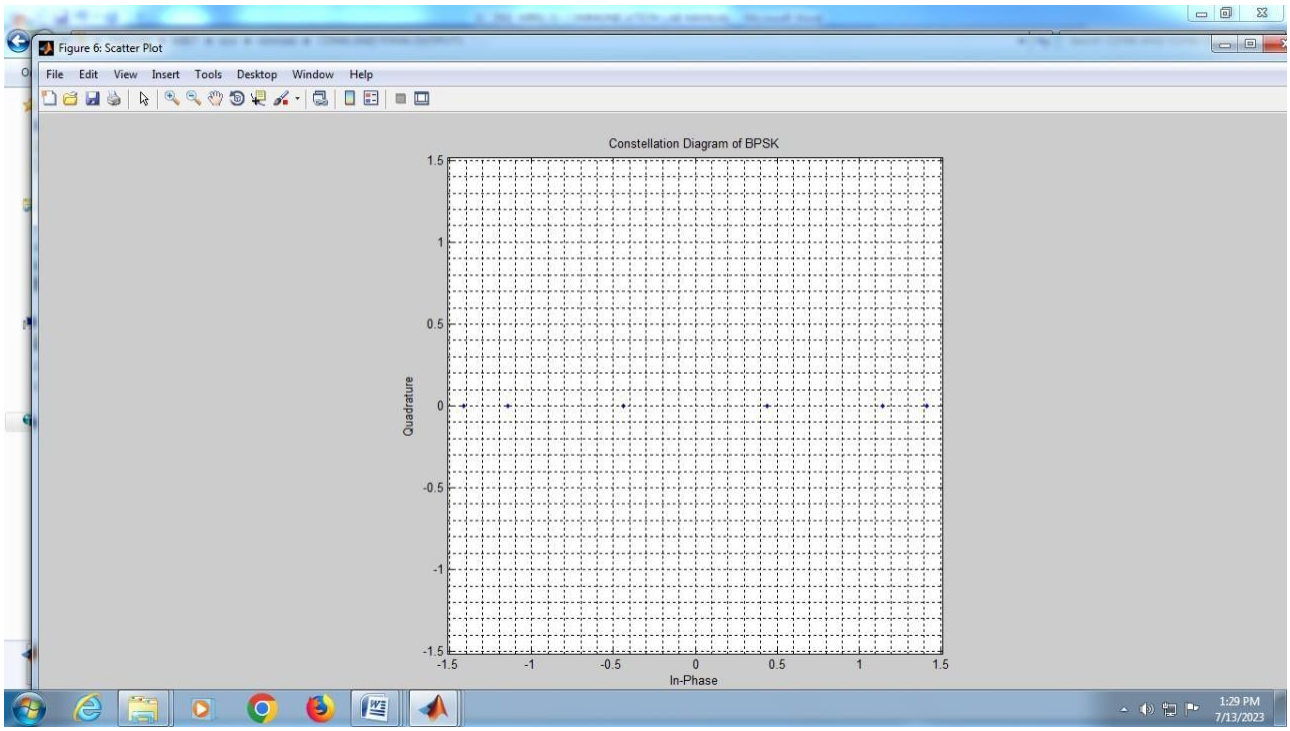


Figure 5: Scatter Plot
File Edit View Insert Tools Desktop Window Help
Constellation Diagram of BPSK with Noise
Quadrature
In-Phase
1:29 PM
7/13/2023



RESULT:

Thus multiple access techniques in wireless communication has been simulated using matlab.

MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

EC3561 VLSI LABORATORY

Semester - 05

LABORATORY MANUAL

MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Vision

To excel in providing value based education in the field of Electronics and Communication Engineering, keeping in pace with the latest technical developments through commendable research, to raise the intellectual competence to match global standards and to make significant contributions to the society upholding the ethical standards.

Mission

- ✓ To deliver Quality Technical Education, with an equal emphasis on theoretical and practical aspects.
- ✓ To provide state of the art infrastructure for the students and faculty to upgrade their skills and knowledge.
- ✓ To create an open and conducive environment for faculty and students to carry out research and excel in their field of specialization.
- ✓ To focus especially on innovation and development of technologies that is sustainable and inclusive, and thus benefits all sections of the society.
- ✓ To establish a strong Industry Academic Collaboration for teaching and research, that could foster entrepreneurship and innovation in knowledge exchange.
- ✓ To produce quality Engineers who uphold and advance the integrity, honour and dignity of the engineering.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

1. To provide the students with a strong foundation in the required sciences in order to pursue studies in Electronics and Communication Engineering.
2. To gain adequate knowledge to become good professional in electronic and communication engineering associated industries, higher education and research.
3. To develop attitude in lifelong learning, applying and adapting new ideas and technologies as their field evolves.
4. To prepare students to critically analyze existing literature in an area of specialization and ethically develop innovative and research oriented methodologies to solve the problems identified.
5. To inculcate in the students a professional and ethical attitude and an ability to visualize the engineering issues in a broader social context.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Design, develop and analyze electronic systems through application of relevant electronics, mathematics and engineering principles.

PSO2: Design, develop and analyze communication systems through application of fundamentals from communication principles, signal processing, and RF System Design & Electromagnetics.

PSO3: Adapt to emerging electronics and communication technologies and develop innovative solutions for existing and newer problems.

LIST OF EXPERIMENTS:

1. Design of basic combinational and sequential (Flip-flops) circuits using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
2. Design an Adder ; Multiplier (Min 8 Bit) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
3. Design and implement Universal Shift Register using HDL. Simulate it using Xilinx/Altera Software
4. Design Memories using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
5. Design Finite State Machine (Moore/Mealy) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
6. Design 3-bit synchronous up/down counter using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
7. Design 4-bit Asynchronous up/down counter using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
8. Design and simulate a CMOS Basic Gates & Flip-Flops. Generate Manual/Automatic Layout.
9. Design and simulate a 4-bit synchronous counter using a Flip-Flops. Generate Manual/Automatic Layout
10. Design and Simulate a CMOS Inverting Amplifier.
11. Design and Simulate basic Common Source, Common Gate and Common Drain Amplifiers.
12. Design and simulate simple 5 transistor differential amplifier.

Exp No :01(a)	DESIGN AND IMPLEMENTATION OF FULL ADDER AND SUBRACTOR
Date :	

AIM:

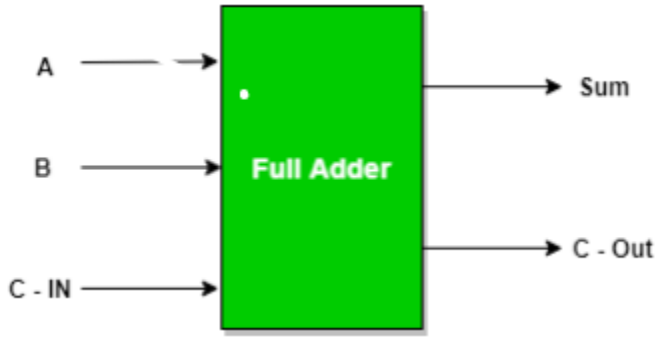
To design a Full adder and subractor using Verilog Hardware Description Language, simulate it using Xilinx software and implement by Xilinx.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 ISE software

FULL ADDER:

Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to the another.



FULL ADDER TRUTH TABLE:

Inputs			Outputs	
A	B	C – IN	Sum	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

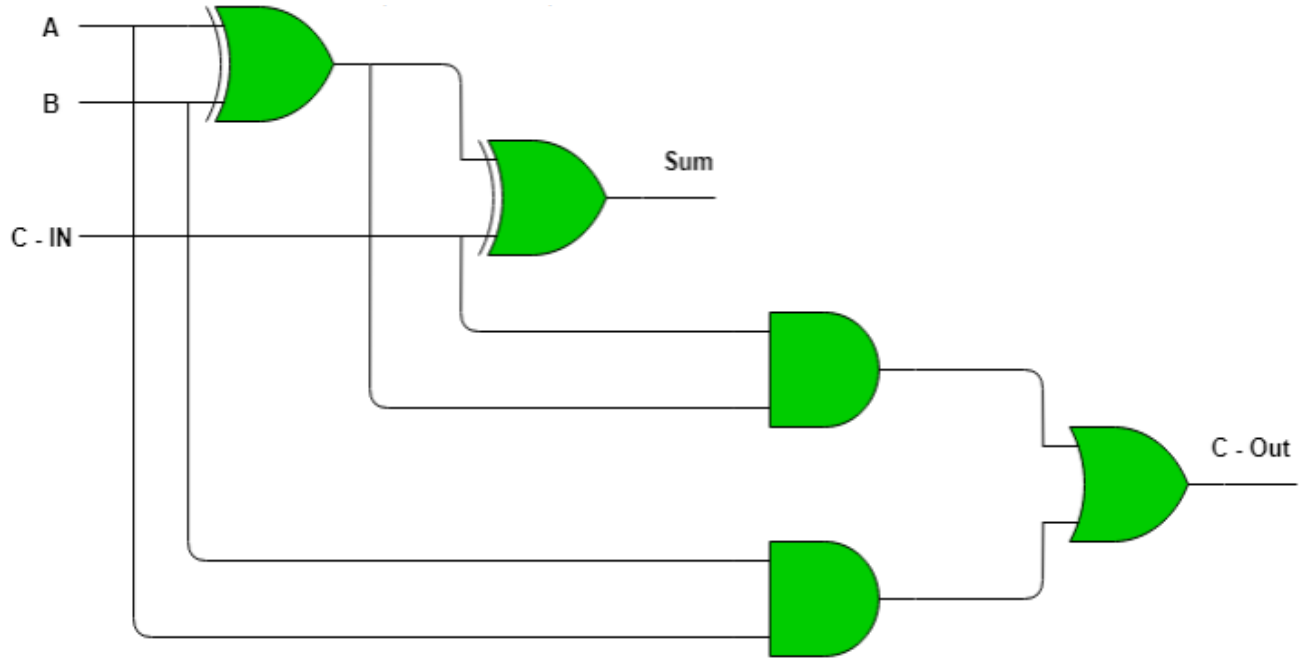
Logical Expression for SUM:

= A’ B’ C-IN + A’ B C-IN’ + A B’ C-IN’ + A B C-IN
= C-IN (A’ B’ + A B) + C-IN’ (A’ B + A B’)
= C-IN XOR (A XOR B) = (1,2,4,7)

Logical Expression for C-OUT:

= A’ B C-IN + A B’ C-IN + A B C-IN’ + A B C-IN
= A B + B C-IN + A C-IN = (3,5,6,7)

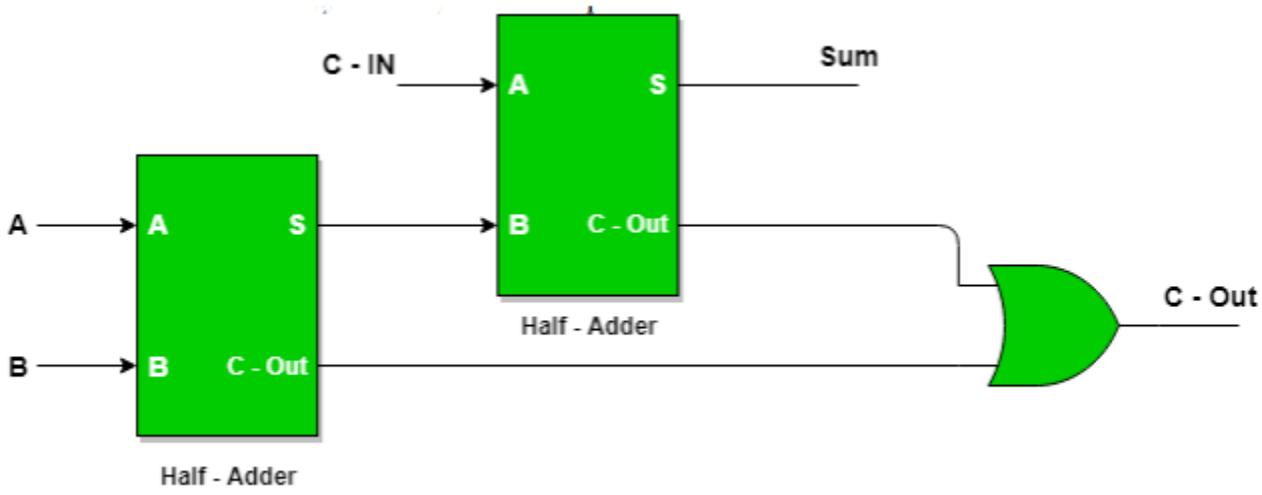
Another form in which C-OUT can be implemented:
= A B + A C-IN + B C-IN (A + A')
= A B C-IN + A B + A C-IN + A' B C-IN
= A B (1 +C-IN) + A C-IN + A' B C-IN
= A B + A C-IN + A' B C-IN
= A B + A C-IN (B + B') + A' B C-IN
= A B C-IN + A B + A B' C-IN + A' B C-IN
= A B (C-IN + 1) + A B' C-IN + A' B C-IN
= A B + A B' C-IN + A' B C-IN
= AB + C-IN (A' B + A B')
Therefore
COUT = AB + C-IN (A EX – OR B)



Full Adder logic circuit.

Implementation of Full Adder using Half Adders :

2 Half Adders and a OR gate is required to implement a Full Adder

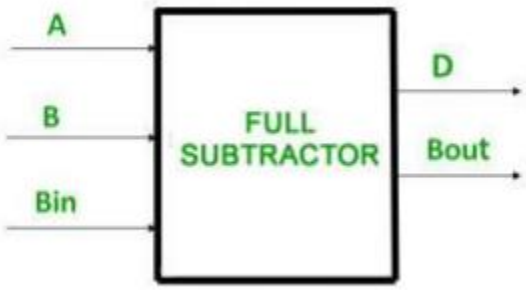


PROGRAM:

```
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, B, Cin : in STD_LOGIC;
          Sum : out STD_LOGIC;
          Cout : out STD_LOGIC);
end FullAdder;

architecture Behavioral of FullAdder is
begin
   process (A, B, Cin)
      variable X, Y, Z : STD_LOGIC;
   begin
      X := A XOR B;
      Y := X XOR Cin;
      Z := (A AND B) OR (X AND Cin);

      Sum <= Y;
      Cout <= Z;
   end process;
end Behavioral;
```

Full Subtractor

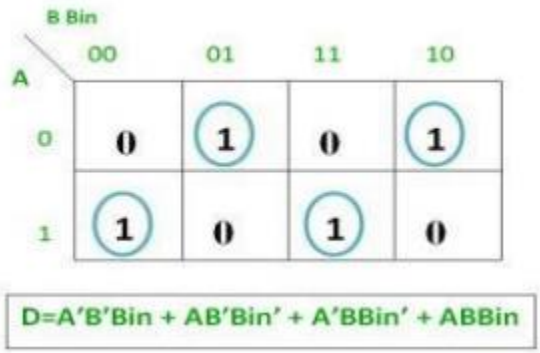
A full subtractor is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit. This circuit has three inputs and two outputs. The three inputs A, B and Bin, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and Bout represent the difference and output borrow, respectively

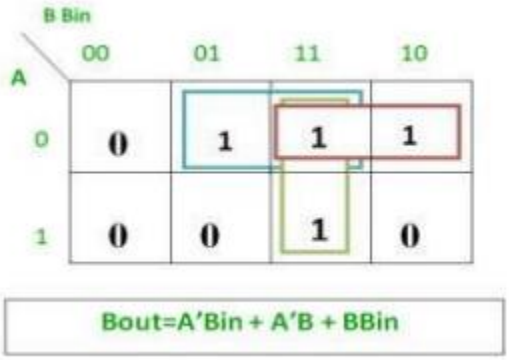


TRUTH TABLE:

INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

From above table we can draw the K-Map as shown for "difference" and "borrow"

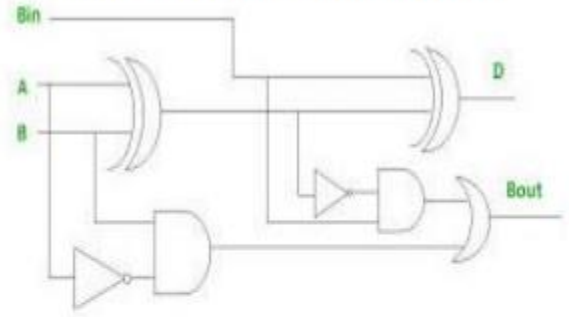




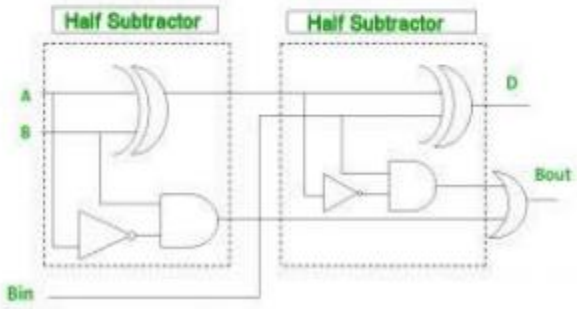
Logical expression for difference –
D = A'B'Bin + A'BBin' + AB'Bin' + ABBin
= Bin(A'B' + AB) + Bin'(AB' + A'B)
= Bin(A XNOR B) + Bin'(A XOR B)
= Bin (A XOR B)' + Bin'(A XOR B)
= Bin XOR (A XOR B)
= (A XOR B) XOR Bin
Logical expression for borrow –
Bout = A'B'Bin + A'BBin' + A'BBin + ABBin
= A'B'Bin +A'BBin' + A'BBin + A'BBin + A'BBin + ABBin
= A'Bin(B + B') + A'B(Bin + Bin') + BBin(A + A')

= A’Bin + A’B + BBin OR Bout
= A’B’Bin + A’BBin’ + A’BBin + ABBin
= Bin(AB + A’B’) + A’B(Bin + Bin’)
= Bin(A XNOR B) + A’B
= Bin (A XOR B)’ + A’B

Logic Circuit for Full Subtractor –



Implementation of Full Subtractor using Half Subtractors –
2 Half Subtractors and an OR gate is required to implement a Full Subtractor.




```
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, B, Bin : in STD_LOGIC;
           Difference, Bout : out STD_LOGIC;
           Cout : out STD_LOGIC);
end FullSubtractor;
```

```
architecture Behavioral of FullSubtractor is
begin
process (A, B, Bin)
        variable Sum1, Sum2, Sum3 : STD_LOGIC;
 begin
        Sum1 := A XOR B;
        Sum2 := Sum1 XOR Bin;
        Difference <= Sum2;
        Sum3 := (A AND NOT B) OR (Bin AND NOT Sum1);
        Bout <= Sum3;
        Cout <= (A AND NOT B) OR (Bin AND NOT A) OR (Bin AND NOT B);
    end process;
end Behavioral;
```

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

entity FullSubtractor_TB is
end FullSubtractor_TB;

architecture Behavioral of FullSubtractor_TB is
    signal A, B, Bin, Difference, Bout, Cout : STD_LOGIC;
begin
    UUT: entity work.FullSubtractor
        port map (
            A => A,
            B => B,
            Bin => Bin,
            Difference => Difference,
            Bout => Bout,
            Cout => Cout
        );

    process
    begin
        A <= '0';
        B <= '1';
        Bin <= '0';
        wait for 10 ns;

        A <= '1';
        B <= '0';
        Bin <= '1';
        wait for 10 ns;
```

```
  -- Add more test cases here

      wait;
   end process;
end Behavioral;
```

OUTPUT:

Exp No :01(b)	DESIGN AND IMPLEMENTATION OF MUX & DEMUX
Date :	

AIM:

To design an 4:1 multiplexer using Verilog Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1.PC with windows XP

2.Xilinx 12.1 ISE software

3.Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation s complete and the simulation result window will open.

THEORY:

Multiplexer:

The multiplexer or MUX is a digital switch, also called as data selector.

It is a Combinational Logic Circuit with more than one input line, one output line and more than one select line.

It accepts the binaryinformation from several input lines or sources and depending on the set of select lines,a particular input line is routed onto a single output line. If there are m selection lines, then the number of possible input lines is 2^m. Alternatively, we can say that if the number of input lines is equal to 2^m, then m selection lines are required to select one of n (consider $2^m = n$) input lines. A 4-to-1 multiplexer consists four data input lines as I0 to I3, two select lines as S0 and S1 and a single output line Y.

The select lines S0 and S1 select one of the four input lines to connect the output line.

Table 1: Truth Table of 4:1 MUX

S0	S1	I0	I1	I2	I3	Y
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

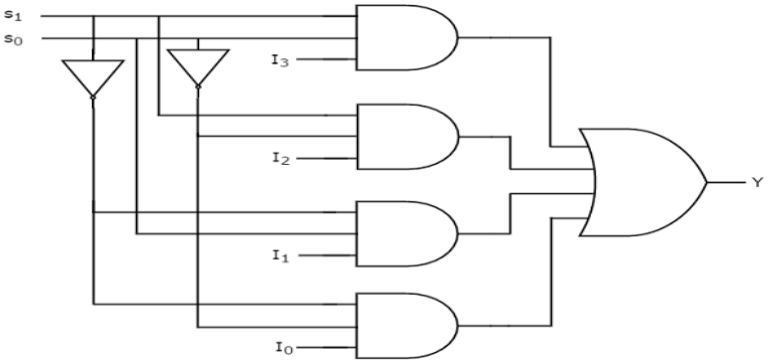


Figure 1: Logic Diagram of 4:1 MUX

From the above truth table, we can write the output expressions as follows:

Y = S0' S1' I0 + S0' S1 I1 + S0 S1' I2 + S0 S1 I3

Demultiplexer:

Demultiplexer performs the reverse operation of the multiplexer i.e. it takes a single output and can guide that single output through many outputs. The output to which the input signal is to be passed is decided by the control logic. The 1:4 Demultiplexer consists of 1 input signal, 2 control signals and 4 output signals.

Table 2: Truth Table of 1:4 DeMUX

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

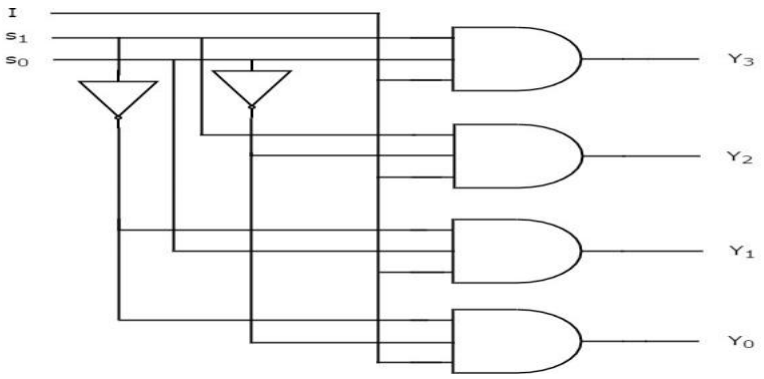


Figure 2: Logic Diagram of 1:4 DeMUX

Verilog Code for 4:1 MUX:

```
module mux_4to1(
   input I0, // Input 0
   input I1, // Input 1
   input I2, // Input 2
   input I3, // Input 3
   input S0, // Select line 0
   input S1, // Select line 1
   output Y // Output
);
   always @(S0,S1)
   begin
```

```
      // Behavioral modeling using if-else statements
      if (S1 && S0)       // S1 S0 = 11
         Y = I3;
      else if (S1 && !S0) // S1 S0 = 10
         Y = I2;
      else if (!S1 && S0) // S1 S0 = 01
         Y = I1;
      else                // S1 S0 = 00
         Y = I0;
   end
endmodule
```

Verilog Code for 1:4 DeMUX:

```
module demux_1to4(
   input I,
   input S0,
   input S1,
   output Y0,
   output Y1,
   output Y2,
   output Y3
);
   // Behavioral modeling using assign statements
   assign Y0 = I & ~S0 & ~S1;
   assign Y1 = I & S0 & ~S1;
   assign Y2 = I & ~S0 & S1;
   assign Y3 = I & S0 & S1;
endmodule
```

RESULT:

Exp No :01(c)	DESIGN AND IMPLEMENTATION OF SIMPLE SEQUENTIAL CIRCUITS
Date :	

AIM:

To design various flip flops using Verilog Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 ISE software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation s complete and the simulation result window will open.

THEORY:

A flip-flop is a sequential digital electronic circuit having two stable states that can be used to store one bit of binary data. Flip-flops are the fundamental building blocks of all memory devices. Types of Flip–Flops are:

- S-R flip-flop
- J-K flip-flop
- D flip-flop
- T flip-flop

Table 3: Truth table of S-R flip-flop

S	R	Q	State
0	0	0	No Change
0	1	0	Reset
1	0	1	Set
1	1	X	

Figure 3: Logic diagram of SR FF

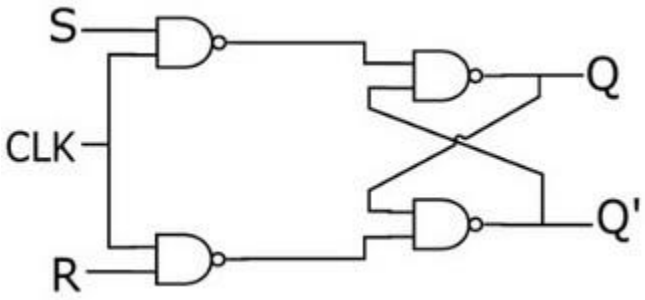


Table 4: Truth table of JK flip-flop

J	K	Q	State
0	0	0	No Change
0	1	0	Reset
1	0	1	Set
1	1	Toggles	Toggle

Figure 4: Logic diagram of JK FF

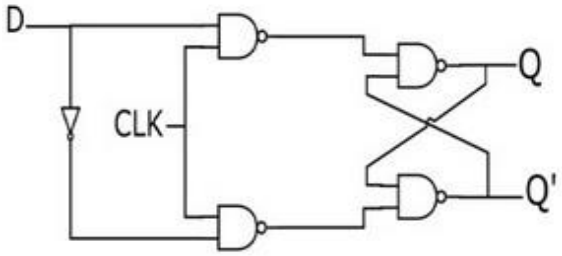


Table 5: Truth table of D flip-flop

D	Q
0	0
1	1

Figure 5: Logic diagram of D FF

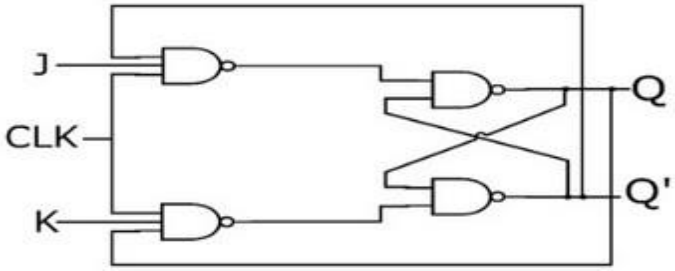


Table 6: Truth table of T flip-flop

T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

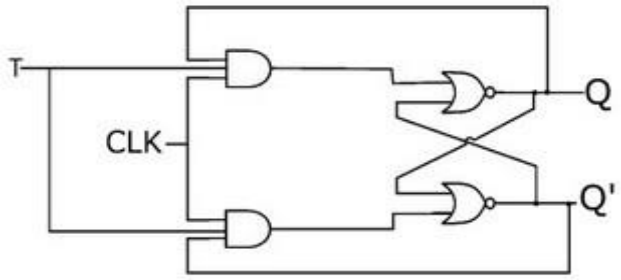


Figure 6: Logic diagram of T FF

Verilog Code for Flip flops:
SR Flip Flop:

```
module SRFlipFlop(S, R, clk, Q, Qn);
input S; // Set input
input R; // Reset input
```

```
input clk; // Clock input
output Q; // Output Q
output Qn; // Complementary output Qn
reg Q, Qn;
always @(posedge clk)
begin
  if (S && !R) // Only valid when S and R are not both 1 (invalid state)
  begin
    Q <= 1'b1; // Set Q to 1 when S=1, R=0 (Set condition)
  Qn <= 1'b0; // Set Qn to 0 when S=1, R=0 (Set condition)
  end
  else if (!S && R) // Only valid when S and R are not both 1 (invalid state)
  begin
    Q <= 1'b0; // Set Q to 0 when S=0, R=1 (Reset condition)
  Qn <= 1'b1; // Set Qn to 1 when S=0, R=1 (Reset condition)
  end
  // No change when S=0, R=0 or S=1, R=1 (No change condition)
end
endmodule
```

JK Flip Flop:

```
module JKFlipFlop(J, K, clk, Q, Qn);
input J; // J input
input K; // K input
input clk; // Clock input
output Q; // Output Q
output Qn; // Complementary output Qn

reg Q, Qn;

always @(posedge clk)
begin
  if (J && !K) // Only valid when J and K are not both 1 (invalid state)
  begin
    Q <= 1'b1; // Toggle Q when J=1, K=0 (Toggle condition)
  Qn <= 1'b0; // Set Qn to 0 when J=1, K=0 (Toggle condition)
  end
  else if (!J && K) // Only valid when J and K are not both 1 (invalid state)
  begin
    Q <= 1'b0; // Toggle Q when J=0, K=1 (Toggle condition)
  Qn <= 1'b1; // Set Qn to 1 when J=0, K=1 (Toggle condition)
  end
  else if (J && K) // Only valid when J and K are not both 1 (invalid state)
  begin
    Q <= ~Q; // Toggle Q when J=1, K=1 (Toggle condition)
```

```
   Qn <= ~Qn; // Toggle Qn when J=1, K=1 (Toggle condition)
  end
  // No change when J=0, K=0 (No change condition)
end
endmodule
```

D Flip Flop:

```
module RisingEdge_DFlipFlop(D,clk,Q);
input D; // Data input
input clk; // clock input
output Q; // output Q
always @(posedge clk)
begin
 Q <= D;
end
endmodule
```

T Flip Flop:

```
module TFlipFlop(T, clk, Q, Qn);
input T; // T input
input clk; // Clock input
output Q; // Output Q
output Qn; // Complementary output Qn
reg Q, Qn;
always @(posedge clk)
begin
  if (T) // Only valid when T is 1
  begin
    Q <= ~Q; // Toggle Q when T=1
    Qn <= ~Qn; // Toggle Qn when T=1
  end
  // No change when T=0 (No change condition)
end
endmodule
```

RESULT:

Exp No : 02 (a)	DESIGN AND IMPLEMENTATION OF 8 BIT ADDER
Date :	

AIM:

To design an 8 bit ripple carry adder using Verilog Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation s complete and the simulation result window will open.

THEORY

Half Adder:

A half adder is a combinational digital circuit, which takes two 1 bit numbers and produces a sum and carry as output.

Full Adder:

A one bit full adder is a device with three single bit binary inputs and two single bits binary outputs. Having both carry in and carry out capabilities, the full adder is highly scalable and found in many cascaded circuit implementations.

Ripple Carry Adder

Ripple Carry Adder works in different stages. Each full adder takes the carry-in as input and produces carry-out and sum bit as output. The carry-out produced by a full adder serves as carry-in for its adjacent most significant full adder. When carry-in becomes available to the full adder, it activates the full adder. After full adder becomes activated, it comes into operation.

Boolean Expression of Half Adder:

$$SUM = A \oplus B$$

$$CARRY = A \,\&\, B$$

Table 3: Truth Table of Half Adder

INPUT		OUTPUT	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

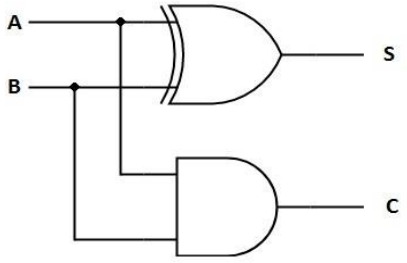


Figure 3: Logic Diagram of Half Adder

Boolean Expression of Full Adder:

$$SUM = A \oplus B \oplus Cin$$

$$CARRY = AB + BCin + ACin$$

Table 4: Truth Table of Full Adder

INPUT			OUTPUT	
A	B	Cin	S	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

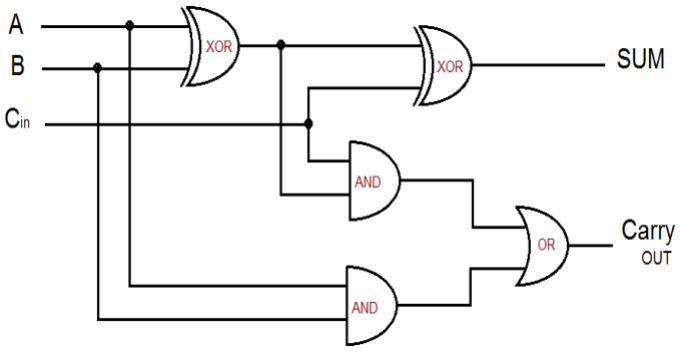


Figure 4: Logic Diagram of Full Adder

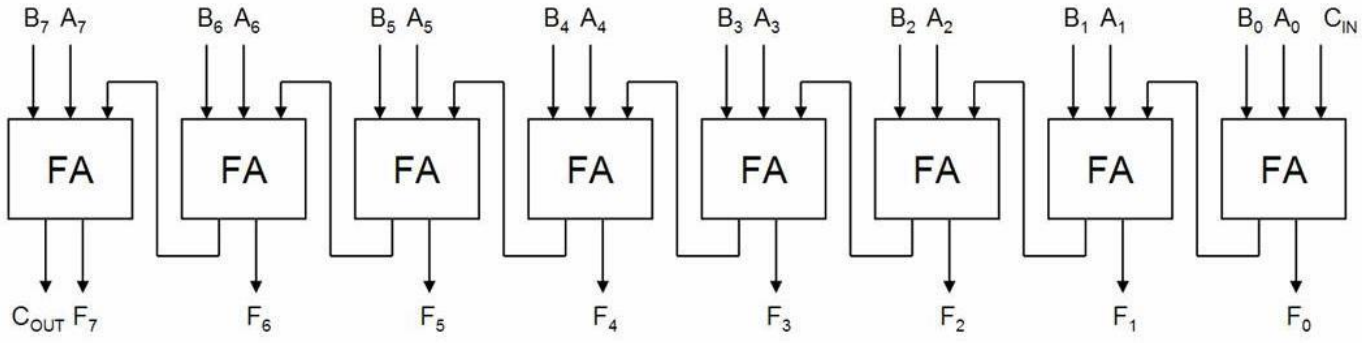


Figure 5: Block Diagram of 8 bit Ripple Carry Adder

Verilog code for an 8 bit Ripple Carry adder:

```
module ripplemod(a, b, cin, sum, cout);
input [07:0] a;
input [07:0] b;
input cin;
output [7:0]sum;
output cout;
wire[6:0] c;
fulladd a1(a[0],b[0],cin,sum[0],c[0]);
fulladd a2(a[1],b[1],c[0],sum[1],c[1]);
fulladd a3(a[2],b[2],c[1],sum[2],c[2]);
fulladd a4(a[3],b[3],c[2],sum[3],c[3]);
fulladd a5(a[4],b[4],c[3],sum[4],c[4]);
fulladd a6(a[5],b[5],c[4],sum[5],c[5]);
fulladd a7(a[6],b[6],c[5],sum[6],c[6]);
fulladd a8(a[7],b[7],c[6],sum[7],cout);
endmodule
 module fulladd(a, b, cin, sum, cout);
input a;
input b;
input cin;
output sum;
output cout;
```

```
assign sum=(a^b^cin);
assign cout=((a&b)|(b&cin)|(a&cin));
endmodule
```

UCF File

```
net “a[0]” loc = “p71”;                    net “b[0]” loc = “p136”;
net “a[1]” loc = “p72”;                    net “b[1]” loc = “p142”;
net “a[2]” loc = “p91”;                    net “b[2]” loc = “p148”;
net “a[3]” loc = “p101”;                   net “b[3]” loc = “p154”;
net “a[4]” loc = “p110”;                   net “b[4]” loc = “p159”;
net “a[5]” loc = “p118”;                   net “b[5]” loc = “p169”;
net “a[6]” loc = “p124”;                   net “b[6]” loc = “p194”;
net “a[7]” loc = “p130”;                   net “b[7]” loc = “p174”;
net “sum[0]” loc = “p102”;                 net “sum[4]” loc = “p109”;
net “sum[1]” loc = “p106”;                 net “sum[5]” loc = “p112”;
net “sum[2]” loc = “p107”;                 net “sum[6]” loc = “p113”;
net “sum[3]” loc = “p108”;                 net “sum[7]” loc = “p115”;
                    net “cout” loc = “p116”;
```

RESULT:

Exp No : 02 (b)	DESIGN AND IMPLEMENT MULTIPLIER
Date :	

AIM:

To design a 4 bit multiplier using Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

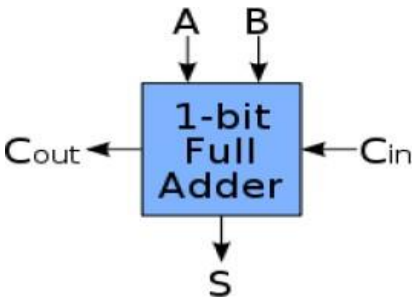
1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation s complete and the simulation result window will open.

Full Adder:

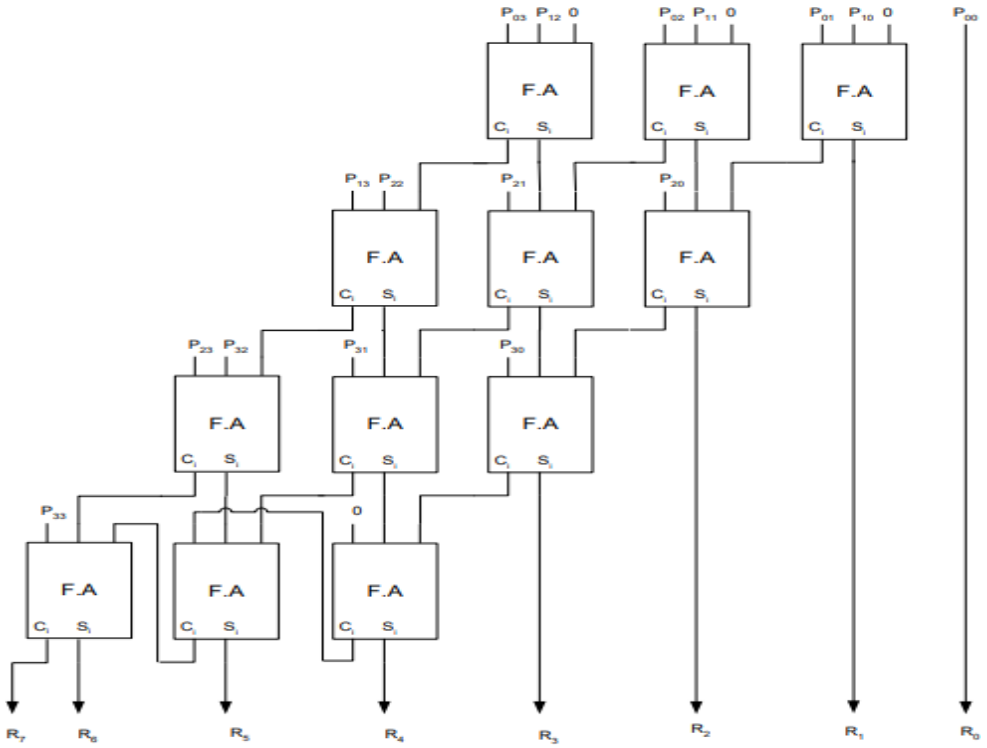
A one bit full adder is a device with three single bit binary inputs and two single bits binary outputs. Having both carry in and carry out capabilities, the full adder is highly scalable and found in many cascaded circuit implementations.



INPUT			OUTPUT	
A	B	Cin	S	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

Multiplier

Array multiplier is well known due to its regular structure. Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added. The addition can be performed with normal carry propagate adder. N-1 adders are required where N is the multiplier length.



Verilog Code for 8 bit Multiplier

```
module array4x4(a,b,p);
//inputs
input [3:0]a,b;
//outputs
output [7:0]p;

//wires
wire [39:0]w;

//andgate instantiations
and a1(w[0],a[0],b[0]);
and a2(w[1],a[1],b[0]);
and a3(w[2],a[2],b[0]);
and a4(w[3],a[3],b[0]);

and a5(w[4],a[0],b[1]);
and a6(w[5],a[1],b[1]);
and a7(w[6],a[2],b[1]);
and a8(w[7],a[3],b[1]);
```

```
and a9(w[8],a[0],b[2]);
and a10(w[9],a[1],b[2]);
and a11(w[10],a[2],b[2]);
and a12(w[11],a[3],b[2]);

and a13(w[12],a[0],b[3]);
and a14(w[13],a[1],b[3]);
and a15(w[14],a[2],b[3]);
and a16(w[15],a[3],b[3]);

assign p[0]=w[0];
//full adders instatiations
fulladder a17(1'b0,w[1],w[4],w[16],w[17]);
fulladder a18(1'b0,w[2],w[5],w[18],w[19]);
fulladder a19(1'b0,w[3],w[6],w[20],w[21]);

fulladder a20(w[8],w[17],w[18],w[22],w[23]);
fulladder a21(w[9],w[19],w[20],w[24],w[25]);
fulladder a22(w[10],w[7],w[21],w[26],w[27]);

fulladder a23(w[12],w[23],w[24],w[28],w[29]);
fulladder a24(w[13],w[25],w[26],w[30],w[31]);
fulladder a25(w[14],w[11],w[27],w[32],w[33]);

fulladder a26(1'b0,w[29],w[30],w[34],w[35]);
fulladder a27(w[31],w[32],w[35],w[36],w[37]);
fulladder a28(w[15],w[33],w[37],w[38],w[39]);

//output assignments
assign p[1]=w[16];
assign p[2]=w[22];
assign p[3]=w[28];
assign p[4]=w[34];
assign p[5]=w[36];
assign p[6]=w[38];
assign p[7]=w[39];

endmodule
```

FULL ADDER

```
module fulladder(a,b,c,s,ca);
//inputs
input a,b,c;
//outputs
output s,ca;
```

```
//full adder assignments.
assign s=(a^b^c);
assign ca=((a&b)|(b&c)|(c&a));
endmodule
```

UCF File

```
net “a[0]” loc = “p71”;
net “a[1]” loc = “p72”;
net “a[2]” loc = “p91”;
net “a[3]” loc = “p101”;
net “b[0]” loc = “p110”;
net “b[1]” loc = “p118”;
net “b[2]” loc = “p124”;
net “b[3]” loc = “p130”;
net “p[0]” loc = “p102”;
net “p[1]” loc = “p106”;
net “p[2]” loc = “p107”;
net “p[3]” loc = “p108”;
net “p[4]” loc = “p109”;
net “p[5]” loc = “p112”;
net “p[6]” loc = “p113”;
net “p[7]” loc = “p115”;
```

RESULT:

Exp No : 03	DESIGN AND IMPLEMENT ALU
Date :	

AIM:

To design a ALU using Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation s complete and the simulation result window will open.

THEORY

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers.

Verilog Code for ALU

```
module alu( a,b,alu_sel,alu_out,carryout);
input [3:0]a,b;
input [3:0]alu_sel;
output [3:0]alu_out;
output carryout;
reg [3:0] alu_result;
wire [4:0] tmp;
assign alu_out = alu_result;
assign tmp = {1'b0,a} + {1'b0,b};
assign carryout = tmp[4]; // Carryout flag
always @(*)
begin
case(alu_sel)
4'b0000:
        alu_result =a+b;
4'b0001:
        alu_result =a-b;
4'b0010:
        alu_result =a*b;
4'b0011:
        alu_result =a/b;
4'b0100:
        alu_result =a<<1;
4'b0101:
        alu_result =a>>1;
4'b0110:
        alu_result = {a[2:0],a[3]};
4'b0111:
        alu_result = {a[0],a[3:1]};
4'b1000:
        alu_result =a&b;
4'b1001:
        alu_result =a|b;
4'b1010:
        alu_result =a^b;
4'b1011:
        alu_result =~(a|b);
4'b1100:
        alu_result =~(a&b);
4'b1101:
        alu_result =~(a^b);
4'b1110:
```

```
                alu_result = (a>b)?4'd1:4'd0 ;
        4'b1111:
                alu_result = (a==b)?4'd1:4'd0 ;
        default: alu_result =a+b ;
        endcase
        end
        endmodule
```

UCF File

```
net “a[0]” loc = “p71”;                 net “alu_sel[0]” loc = “p136”;
net “a[1]” loc = “p72”;                 net “alu_sel [1]” loc = “p142”;
net “a[2]” loc = “p91”;                 net “alu_sel [2]” loc = “p148”;
net “a[3]” loc = “p101”;                net “alu_sel [3]” loc = “p154”;
net “b[0]” loc = “p110”;                net “alu_out[0]” loc = “p102”;
net “b[1]” loc = “p118”;                net “alu_out [1]” loc = “p106”;
net “b[2]” loc = “p124”;                net “alu_out [2]” loc = “p107”;
net “b[3]” loc = “p130”;                net “alu_out [3]” loc = “p108”;
                        net “carryout” loc = “p116”;
```

RESULT

Exp No : 04	DESIGN AND IMPLEMENT UNIVERSAL SHIFT REGISTER
Date :	

AIM:

To design a Universal Shift Register using Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

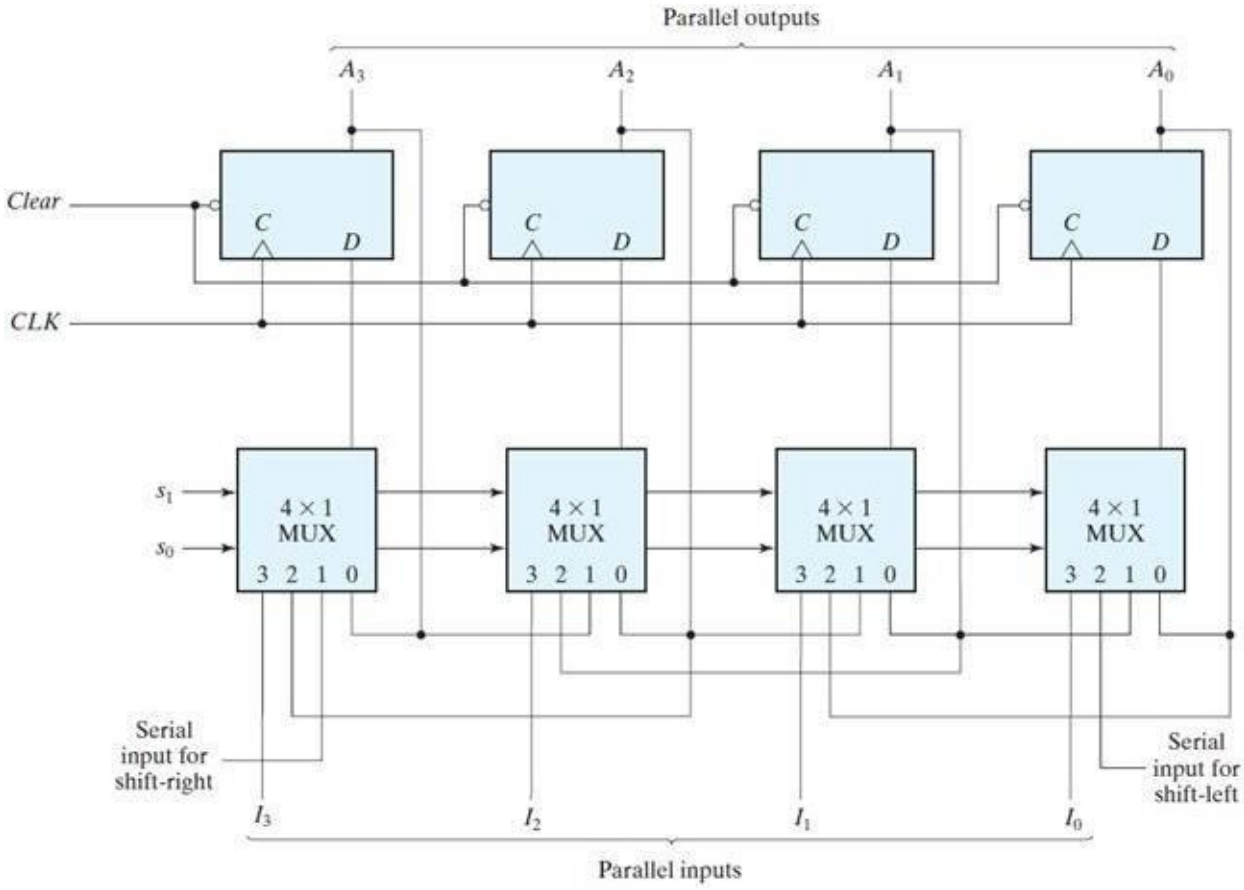
Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation is complete and the simulation result window will open.

THEORY

A register capable of shifting in one direction only is a unidirectional shift register & register that can shift in both directions is bi-directional shift registers. If the register has both the shifts & parallel load capabilities, it is referred to as universal shift register.

S_1	S_0	Register operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load



Verilog Code for Universal Shift Register

```
module universal_shift(a,s,clk,p);
input [3:0]a;
input [1:0]s;
input clk;
output reg [3:0]p;
initial
p<=4'b0110;
always@(posedge clk)
begin
case (s)
2'b00:
begin
p[3]<=p[3];
p[2]<=p[2];
p[1]<=p[1];
p[0]<=p[0];
end
2'b01:
begin
p[3]<=p[0];
```

```
p[2]<=p[3];
p[1]<=p[2];
p[0]<=p[1];
end
2'b10:
begin
p[0]<=p[3];
p[1]<=p[0];
p[2]<=p[1];
p[3]<=p[2];
end
2'b11:
begin
p[0]<=a[0];
p[1]<=a[1];
p[2]<=a[2];
p[3]<=a[3];
end
endcase
end
endmodule
```

UCF File

```
net “a[0]” loc = “p71”;
net “a[1]” loc = “p72”;
net “a[2]” loc = “p91”;
net “a[3]” loc = “p101”;
net “s[0]” loc = “p110”;
net “s[1]” loc = “p118”;
net “p[0]” loc = “p102”;
net “p[1]” loc = “p106”;
net “p[2]” loc = “p107”;
net “p[3]” loc = “p108”;
net “clk” loc = “”;
```

RESULT

Exp No : 05	DESIGN AND IMPLEMENT FINITE STATE MACHINE
Date :	

AIM:

To design a Finite State Machine using Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

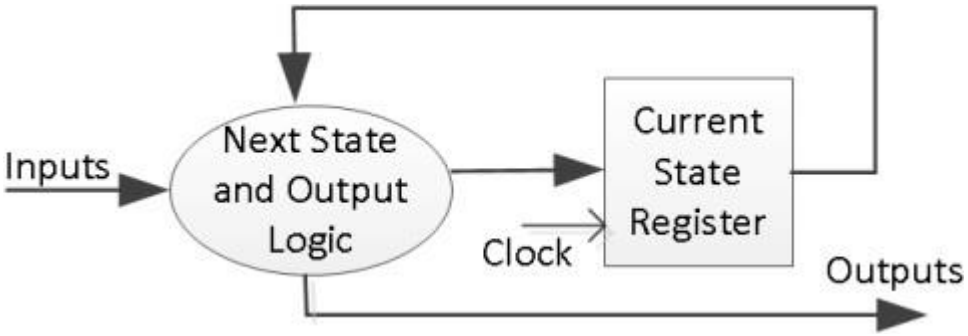
Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation is complete and the simulation result window will open.

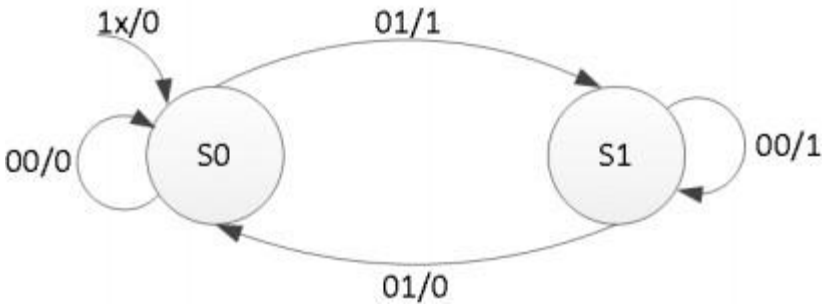
THEORY

Basically a FSM consists of combinational, sequential and output logic. Combinational logic is used to decide the next state of the FSM, sequential logic is used to store the current state of the FSM. The output logic is a mixture of both combo and sequential logic.

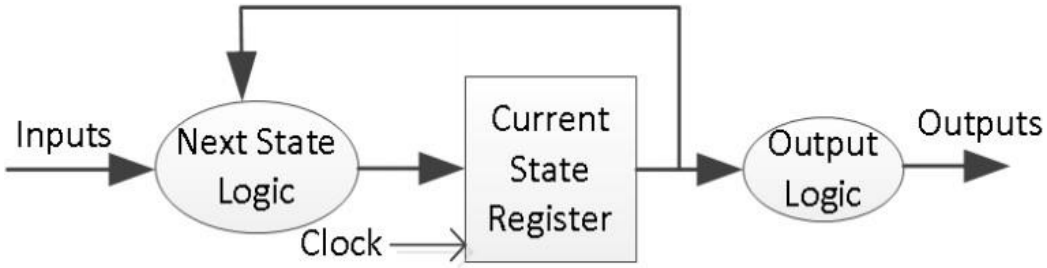
- **Mealy State Machine:** Its output depends on current state and current inputs. In the above picture, the blue dotted line makes the circuit a mealy state machine.
- **Moore State Machine:** Its output depends on current state only. In the above picture, when blue dotted line is removed the circuit becomes a Moore state machine.



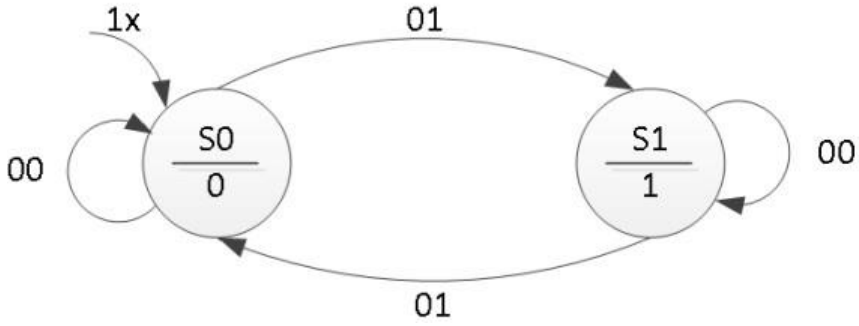
Mealy Machine – Block Diagram



Mealy Machine – Model



Moore Machine – Block Diagram



Moore Machine – Model

Verilog Code for Mealy FSM

```
module mealy_3processes(clk, reset, x, parity);
input clk,reset,x;
output parity;
reg parity;
reg state, nextstate;
parameter s0=0, s1=1;
always @(posedge clk or posedge reset) // always block to update state
if (reset)
 state <= s0;
else
 state <= nextstate;
always @(state or x) // always block to compute output
begin
 parity = 1'b0;
 case(state)
 s0: if(x)
 parity = 1;
 s1: if(!x)
 parity = 1;
 endcase
end
always @(state or x) // always block to compute nextstate
begin
 nextstate = s0;
 case(state)
 s0: if(x)
 nextstate = s1;
 s1: if(!x)
 nextstate = s1;
 endcase
end
endmodule
```

Verilog Code for Moore FSM

```
module moore_3processes(clk, reset, x, parity);
input clk,reset,x;
output parity;
reg parity;
 reg state, nextstate;
 parameter s0=0, s1=1;
always @(posedge clk or posedge reset) // always block to update state
if (reset)
 state <= s0;
else
 state <= nextstate;
always @(state) // always block to compute output
begin
 case(state)
 s0: parity = 0;
 s1: parity = 1;
 endcase
end
always @(state or x) // always block to compute nextstate
begin
 nextstate = s0;
 case(state)
 s0: if(x)
 nextstate = s1;
 s1: if(!x)
 nextstate = s1;
 endcase
end
endmodule
```

UCF File

```
net "a" loc = "p71";                    net "clk" loc = "";
net "reset" loc = "";                   net "parity" loc = "p106";
```

RESULT

Exp No : 06	DESIGN AND IMPLEMENT MEMORIES
Date :	

AIM:

To design a Finite State Machine using Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

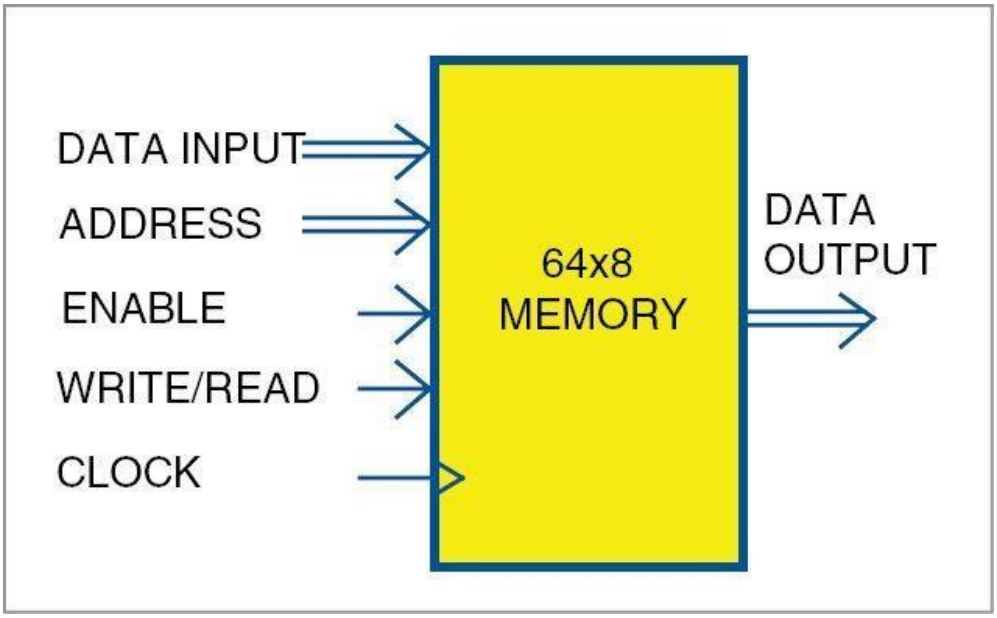
Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation is complete and the simulation result window will open.

THEORY

The memory block diagram is shown in Fig. It takes a few assumptions into consideration for easing the operations of the circuit. While data input pin and address pin may have any value depending on the specifications of memory used and your need, clock used in the circuit is active high.

Enable pin triggers the circuit when it is active high, and read operation is performed when read/write pin is high, while write operation is performed when read/write pin is active low.



Block diagram of 64×8 bit memory

Verilog Code for Memories

```
module single_port_ram();
input [7:0] data;
input [5:0] addr;
input we, clk;
output [7:0] q;
reg [7:0] ram[63:0];
reg [5:0] addr_reg;
always @ (posedge clk)
begin
if (we)
ram[addr] <= data;
addr_reg <= addr;
end
 assign q = ram[addr_reg];
endmodule
```

UCF File

net “data[0]” loc = “p71”;
net “data[1]” loc = “p72”;
net “data[2]” loc = “p91”;
net “data[3]” loc = “p101”;
net “data[4]” loc = “p110”;
net “data[5]” loc = “p118”;
net “data[6]” loc = “p124”;
net “data[7]” loc = “p130”;
net “q[0]” loc = “p102”;
net “q[1]” loc = “p106”;
net “q[2]” loc = “p107”;
net “q[3]” loc = “p108”;
net “addr[0]” loc = “p136”;
net “addr[1]” loc = “p142”;
net “addr[2]” loc = “p148”;
net “addr[3]” loc = “p154”;
net “addr[4]” loc = “p159”;
net “addr[5]” loc = “p169”;
net “we” loc = “p194”;
net “clk” loc = “p174”;
net “q[4]” loc = “p109”;
net “q[5]” loc = “p112”;
net “q[6]” loc = “p113”;
net “q[7]” loc = “p115”;

RESULT

Exp No : 07	DESIGN AND SIMULATE A CMOS INVERTER
Date :	

AIM:

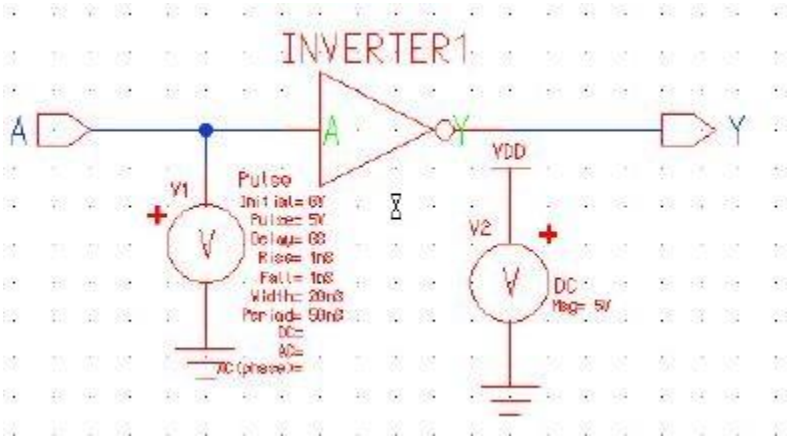
To design and simulate a CMOS inverter using Mentor Graphics EDA tool. Generate layout and post layout extraction for CMOS inverter. Also analyze the power, area and timing by performing Pre Layout and Post Layout Simulations.

APPARATUS REQUIRED:

1. PC with windows XP
2. Mentor Graphics EDA Tool

THEORY:

CMOS Inverter consists of nMOS and pMOS transistor in series connected between VDD and GND. The gate of the two transistors are shorted and connected to the input. When the input to the inverter A = 0, nMOS transistor is OFF and pMOS transistor is ON. The output is pull-up to VDD. When the input A = 1, nMOS transistor is ON and pMOS transistor is OFF. The Output is Pull-down to GND.



PROCEDURE:

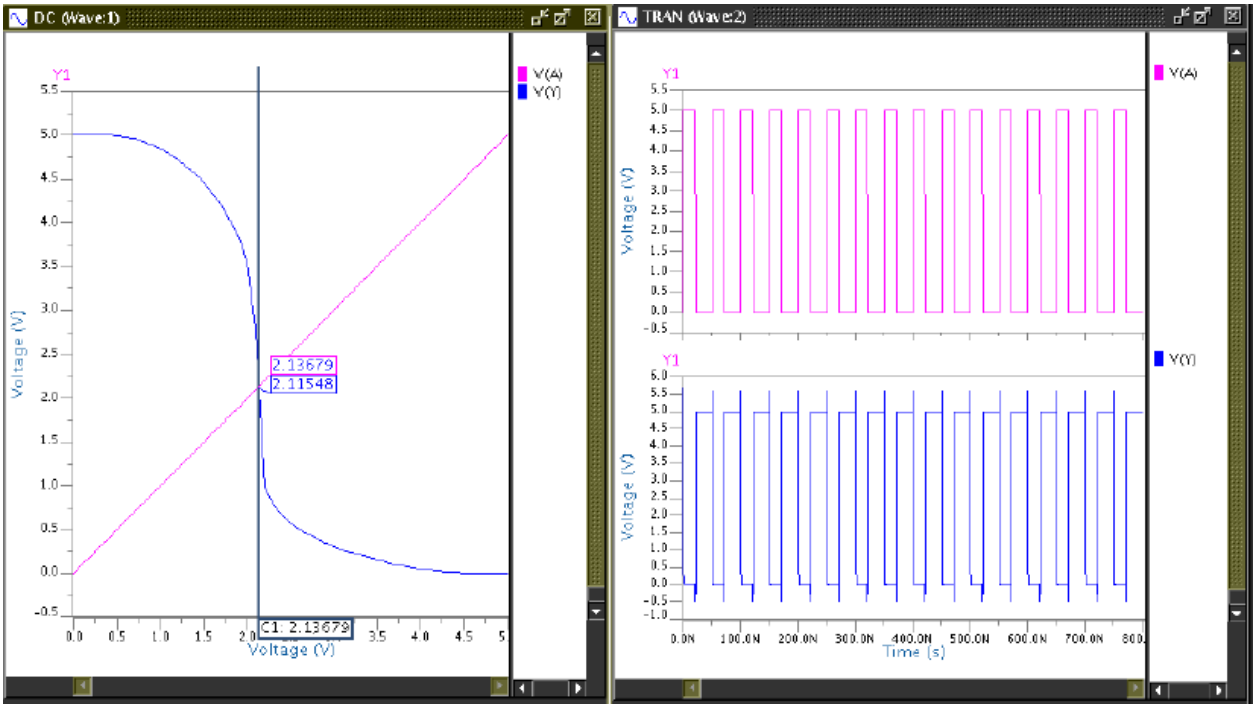
Schematic Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, sedit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Type the name of the cell and select view type as schematic.
6. Add nMOS and pMOS transistors and connect using wires.
7. Click on cell and generate the symbol.
8. Click on cell and select new view.
9. Type the name of the cell and select view type as schematic. Press “I” to instance the symbol.
10. Make connect between Vin, Vout, Vdd, Gnd and Vdc

11. Click on Setup Simulation Icon and add libraries.
12. Enable the Transient Fourier analysis. Provide simulation stop time 100 ns, Step time 1ns, and print time 1ns.
13. Save, run the simulation and check the Functionality of the Design in waveform window.

Layout Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, ledit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Provide the Cell name same as the Schematic design cell name.
6. Go to Tools> SDL Navigator > Load Netlist.
7. Then click on Layout tab enable Metal1 Layer and Mention the Size in Microns.
8. Tool > SDL Navigator> route> route all.
9. Go to file > Export > GDSII and Click on DRC icon in Ledit window
10. Load the Rule file path, load input layout file and click on run DRC
11. For LVS, load Netlist file from Inverter test bench folder and run LVS.
12. run PEX and click on Start RVE



RESULT

Exp No : 08	DESIGN AND SIMULATE A CMOS BASIC GATES AND FLIP FLOPS
Date :	

AIM:

To design and simulate a CMOS basic gates and flip flop using Mentor Graphics EDA tool. Generate layout and post layout extraction for CMOS basic gates and flip flop. Also analyze the power, area and timing by performing Pre Layout and Post Layout Simulations.

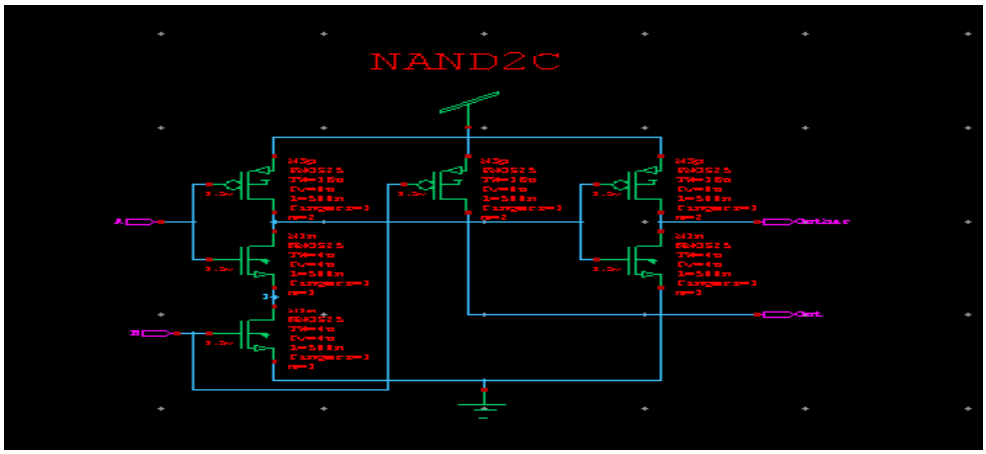
APPARATUS REQUIRED:

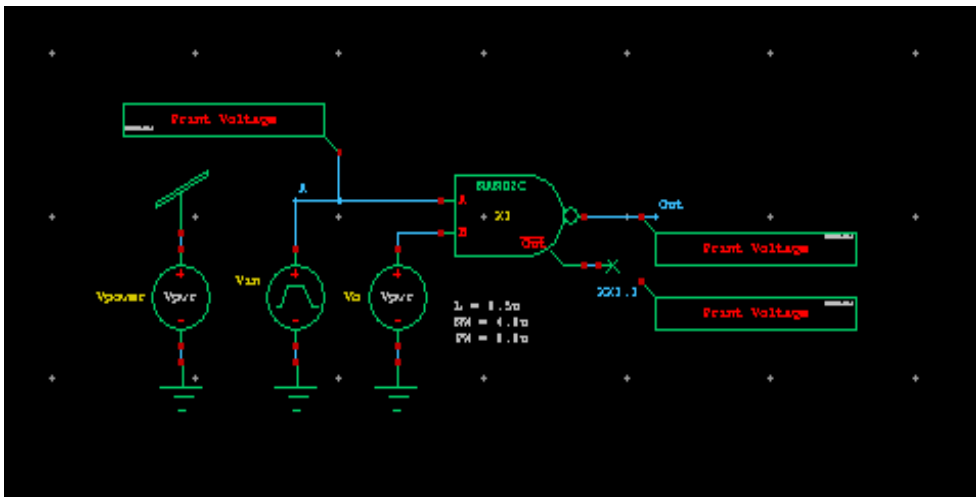
1. PC with windows XP
2. Mentor Graphics EDA Tool

THEORY:

NAND Gate

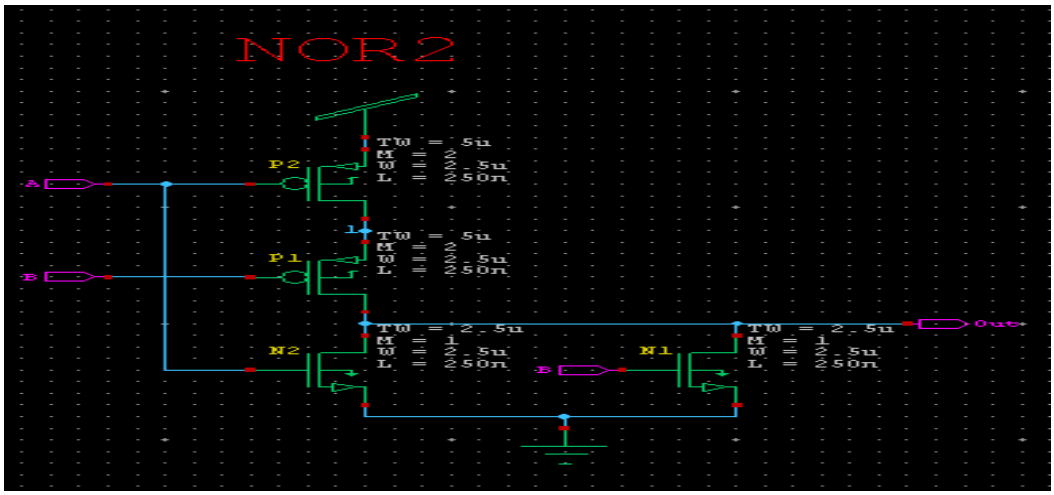
NAND and NOR gates are known as universal gates as any function can be implemented with them NAND functionality can be implemented by parallel combination of PMOS and series combination of NMOS transistor. When any one of the inputs is zero, then the output will be one and when both the inputs are one the output will be low.



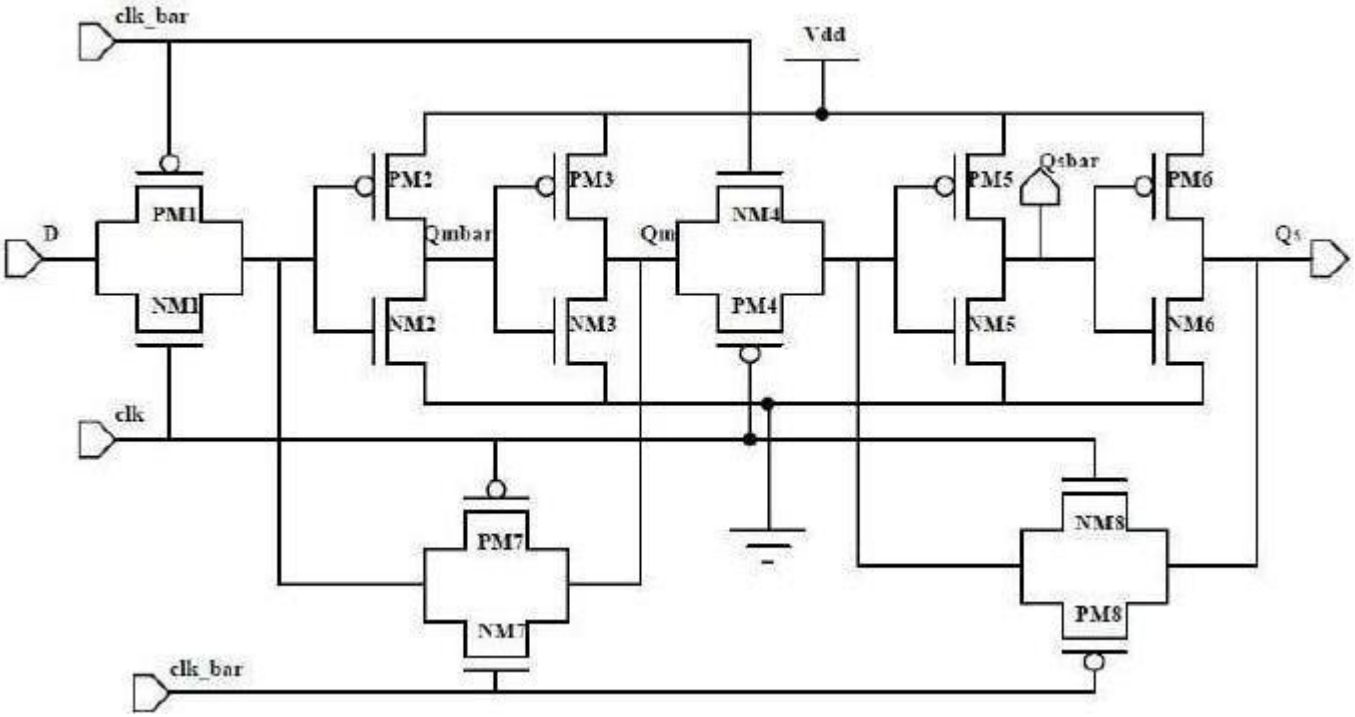


NOR Gate

NOR functionality can be implemented by parallel combination of NMOS and series combination of PMOS transistor. When any one of the inputs is one, then the output will be one and when both the inputs are zero the output will be low.



CMOS Negative Edge Triggered Flip Flop



PROCEDURE:

Schematic Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, sedit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Type the name of the cell and select view type as schematic.

6. Add nMOS and pMOS transistors and connect using wires.
7. Click on cell and generate the symbol.
8. Click on cell and select new view.
9. Type the name of the cell and select view type as schematic. Press “I” to instance the symbol.
10. Make connect between Vin, Vout, Vdd, Gnd and Vdc
11. Click on Setup Simulation Icon and add libraries.
12. Enable the Transient Fourier analysis. Provide simulation stop time 100 ns, Step time 1ns, and print time 1ns.
13. Save, run the simulation and check the Functionality of the Design in waveform window.

Layout Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, ledit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Provide the Cell name same as the Schematic design cell name.
6. Go to Tools> SDL Navigator > Load Netlist.
7. Then click on Layout tab enable Metal1 Layer and Mention the Size in Microns.
8. Tool > SDL Navigator> route> route all.
9. Go to file > Export > GDSII and Click on DRC icon in Ledit window
10. Load the Rule file path, load input layout file and click on run DRC
11. For LVS, load Netlist file from Inverter test bench folder and run LVS.
12. run PEX and click on Start RVE

RESULT

Exp No : 09	DESIGN AND SIMULATE A 4-BIT SYNCHRONOUS COUNTER USING A FLIP-FLOPS
Date :	

AIM:

To design and simulate a 4-bit synchronous counter using a flip-flops using Mentor Graphics EDA tool. Generate layout and post layout extraction for a 4-bit synchronous counter using a flip-flops. Also analyze the power, area and timing by performing Pre Layout and Post Layout Simulations.

APPARATUS REQUIRED:

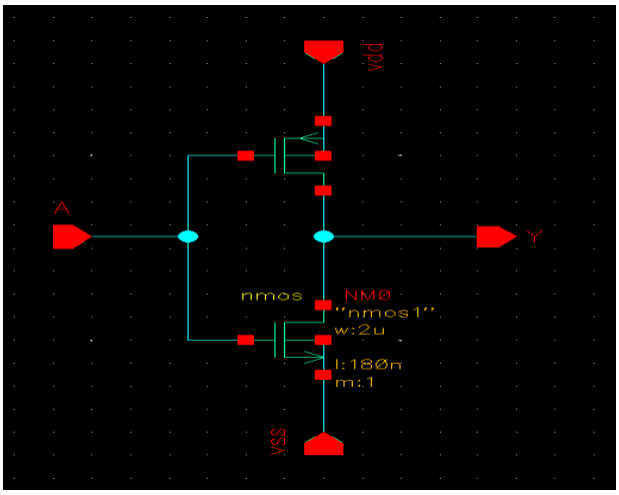
1. PC with windows XP
2. Mentor Graphics EDA Tool

THEORY:

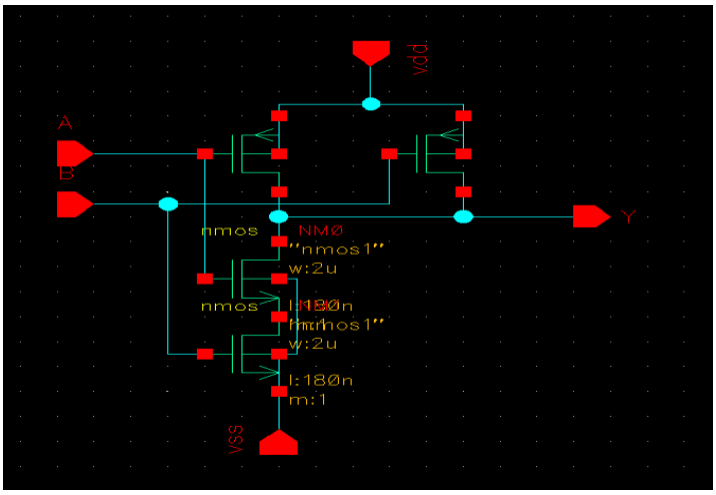
Synchronous Counter

Synchronous counter is the most popular type of counter. It typically consists of a memory element, which is implemented using flip-flops and a combinational element, which is traditionally implemented using logic gates. Logic gates are logic circuits with one or more input terminals and one output terminal in which the output is switched between two voltage levels determined by a combination of input signals. The use of logic gates for combinational logic typically reduces the cost of components for counter circuits to an absolute minimum, so it remains a popular approach.

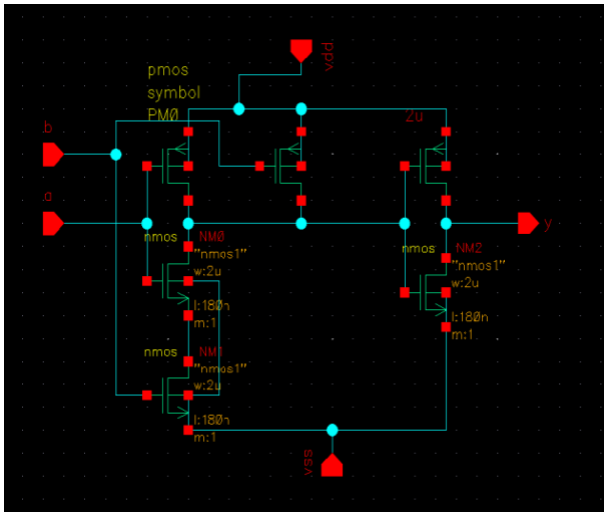
Synchronous counters have an internal clock, whereas asynchronous counters do not. As a result, all the flip-flops in a synchronous counter are driven simultaneously by a single, common clock pulse. In an asynchronous counter, the first flip-flop is driven by a pulse from an external clock and each successive flip-flop is driven by the output of the preceding flip-flop in the sequence. This is the essential difference between synchronous and asynchronous counters.



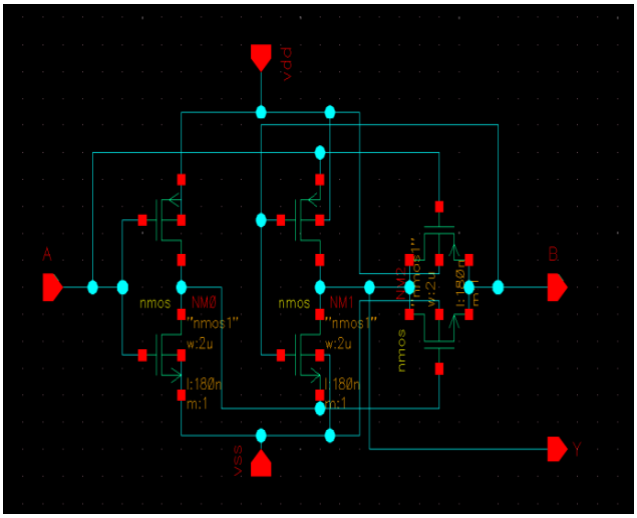
Inverter Schematic Diagram



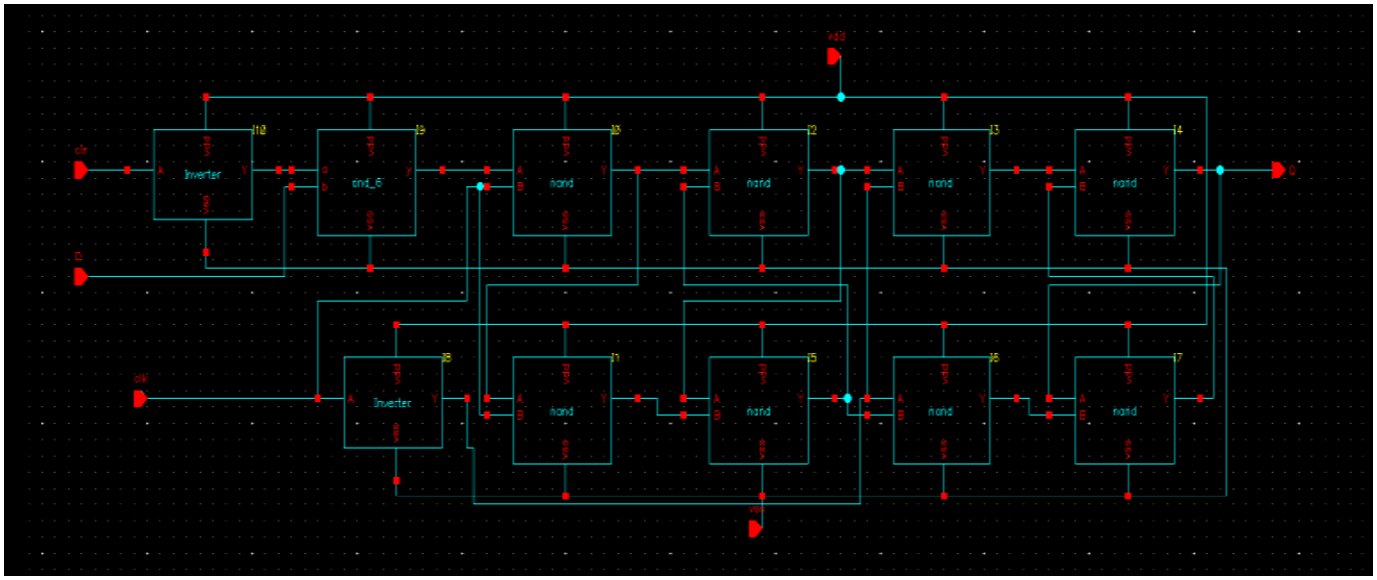
NAND Gate Schematic Diagram



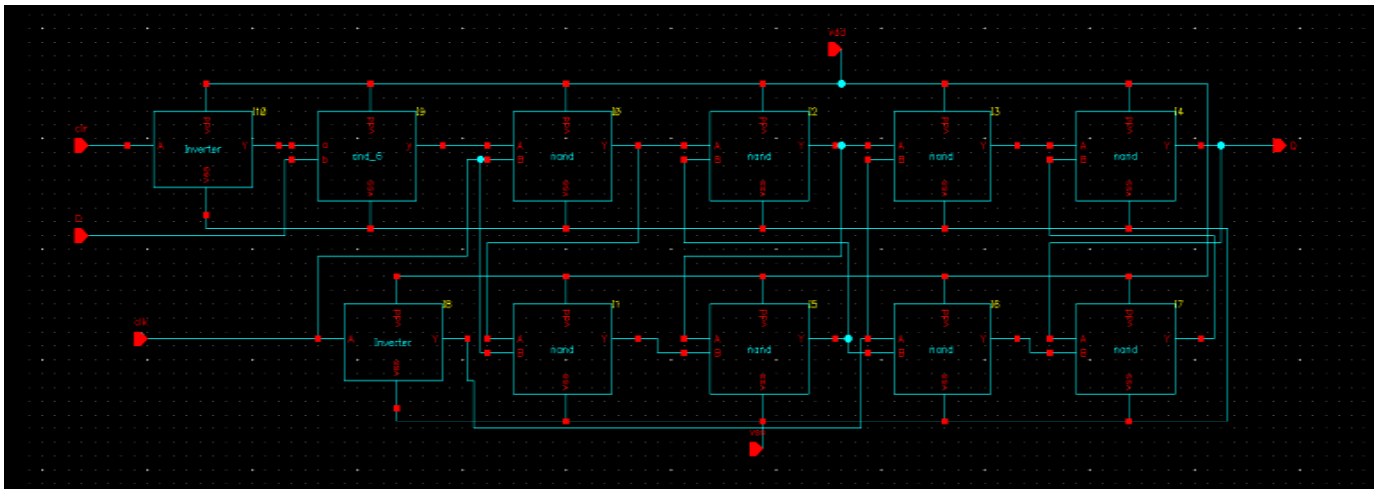
AND Gate Schematic Diagram



XOR Gate Schematic Diagram



Master Slave D Flip Flop Schematic Diagram



Synchronous 4 bit UP counter Schematic Diagram

PROCEDURE:

Schematic Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, sedit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Type the name of the cell and select view type as schematic.
6. Add nMOS and pMOS transistors and connect using wires.
7. Click on cell and generate the symbol.
8. Click on cell and select new view.
9. Type the name of the cell and select view type as schematic. Press "I" to instance the symbol.
10. Make connect between Vin, Vout, Vdd, Gnd and Vdc
11. Click on Setup Simulation Icon and add libraries.
12. Enable the Transient Fourier analysis. Provide simulation stop time 100 ns, Step time 1ns, and print time 1ns.
13. Save, run the simulation and check the Functionality of the Design in waveform window.

Layout Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, ledit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Provide the Cell name same as the Schematic design cell name.
6. Go to Tools> SDL Navigator > Load Netlist.
7. Then click on Layout tab enable Metal1 Layer and Mention the Size in Microns.
8. Tool > SDL Navigator> route> route all.
9. Go to file > Export > GDSII and Click on DRC icon in Ledit window
10. Load the Rule file path, load input layout file and click on run DRC
11. For LVS, load Netlist file from Inverter test bench folder and run LVS.
12. run PEX and click on Start RVE

RESULT

Exp No : 10	DESIGN AND SIMULATE A CMOS INVERTING AMPLIFIER
Date :	

AIM:

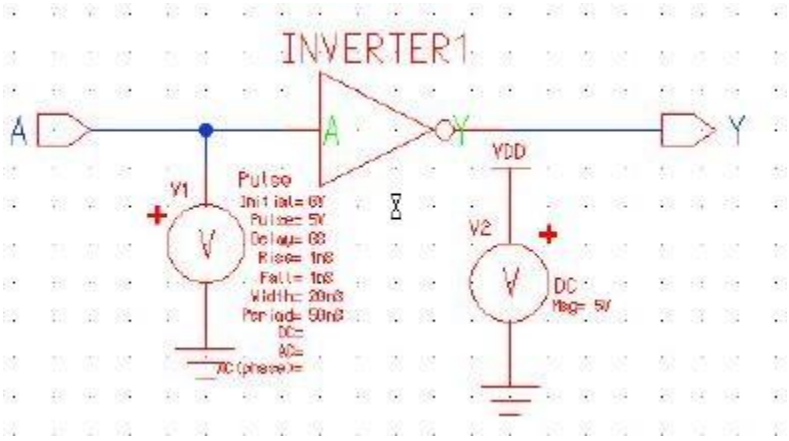
To design and simulate a CMOS inverting amplifier using Mentor Graphics EDA tool. Generate layout and post layout extraction for CMOS inverting amplifier. Analyze the input impedance, output impedance, gain and bandwidth by performing Schematic Simulations.

APPARATUS REQUIRED:

1. PC with windows XP
2. Mentor Graphics EDA Tool

THEORY:

CMOS Inverter consists of nMOS and pMOS transistor in series connected between VDD and GND. The gate of the two transistors are shorted and connected to the input. When the input to the inverter A = 0, nMOS transistor is OFF and pMOS transistor is ON. The output is pull-up to VDD. When the input A = 1, nMOS transistor is ON and pMOS transistor is OFF. The Output is Pull-down to GND.



PROCEDURE:

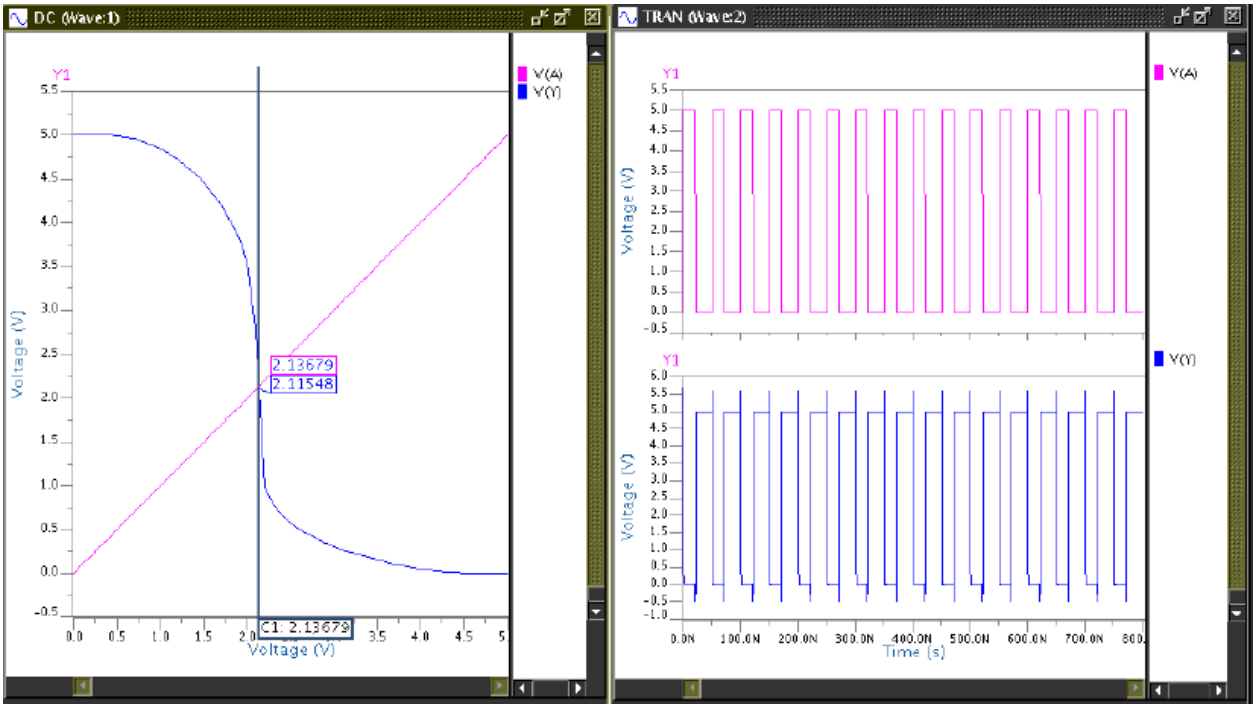
Schematic Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, sedit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Type the name of the cell and select view type as schematic.
6. Add nMOS and pMOS transistors and connect using wires.
7. Click on cell and generate the symbol.
8. Click on cell and select new view.
9. Type the name of the cell and select view type as schematic. Press “I” to instance the symbol.
10. Make connect between Vin, Vout, Vdd, Gnd and Vdc

11. Click on Setup Simulation Icon and add libraries.
12. Enable the Transient Fourier analysis. Provide simulation stop time 100 ns, Step time 1ns, and print time 1ns.
13. Save, run the simulation and check the Functionality of the Design in waveform window.

Layout Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, ledit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Provide the Cell name same as the Schematic design cell name.
6. Go to Tools> SDL Navigator > Load Netlist.
7. Then click on Layout tab enable Metal1 Layer and Mention the Size in Microns.
8. Tool > SDL Navigator> route> route all.
9. Go to file > Export > GDSII and Click on DRC icon in Ledit window
10. Load the Rule file path, load input layout file and click on run DRC
11. For LVS, load Netlist file from Inverter test bench folder and run LVS.
12. run PEX and click on Start RVE



RESULT

Exp No : 11	DESIGN AND SIMULATE BASIC COMMON SOURCE, COMMON GATE AND COMMON DRAIN AMPLIFIERS
Date :	

AIM:

To design and simulate a basic Common Source, Common Gate and Common Drain Amplifiers using Mentor Graphics EDA tool. Generate layout and post layout extraction for ommon Source, Common Gate and Common Drain Amplifiers. Analyze the input impedance, output impedance, gain and bandwidth by performing Schematic Simulations.

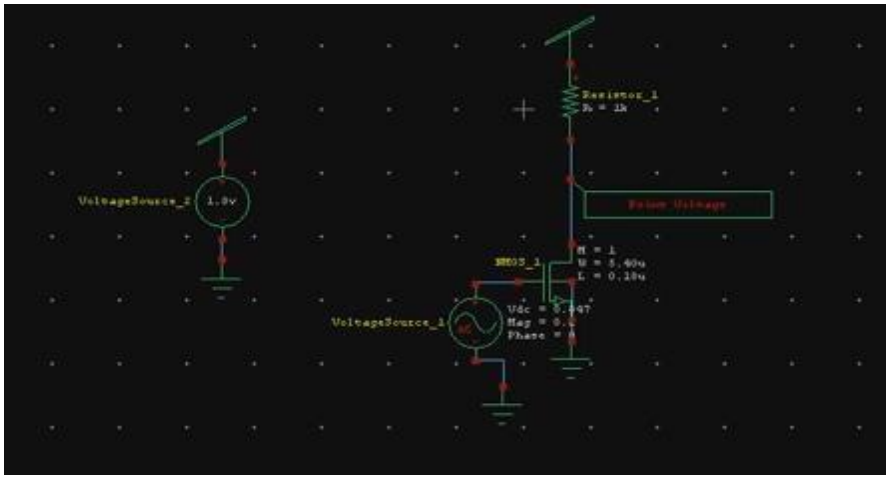
APPARATUS REQUIRED:

1. PC with windows XP
2. Mentor Graphics EDA Tool

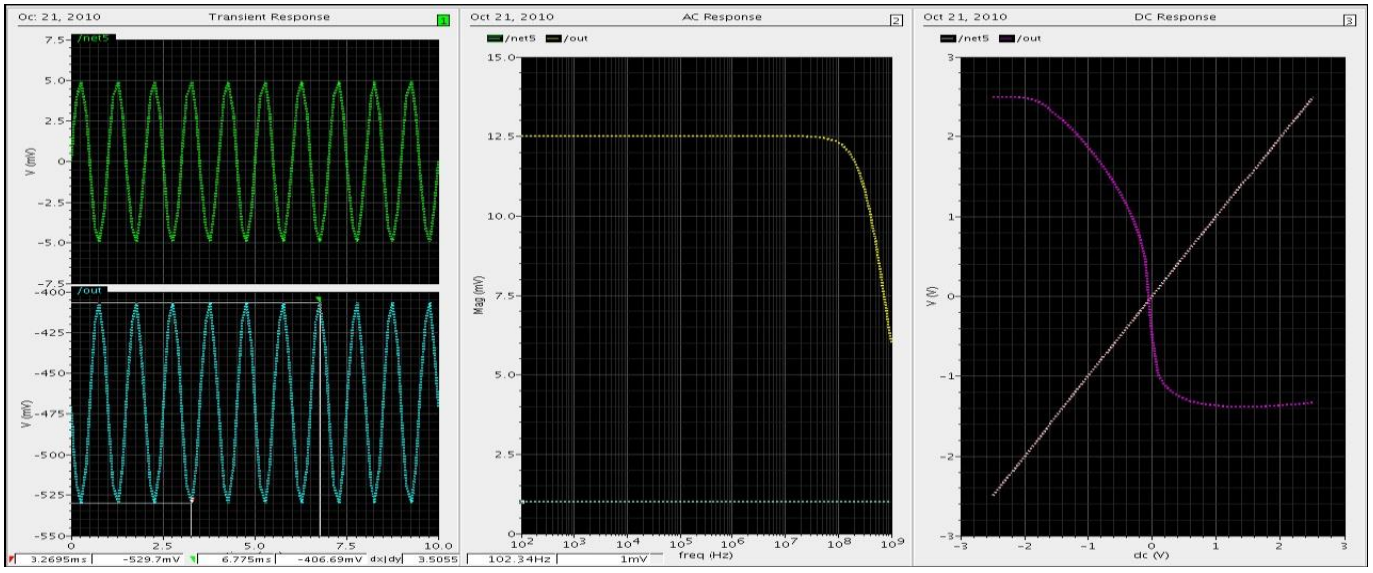
THEORY:

<u>Common Source Amplifier</u>

In electronics, a common-source amplifier is one of three basic single-stage field-effect transistor (FET) amplifier topologies, typically used as a voltage or transconductance amplifier.

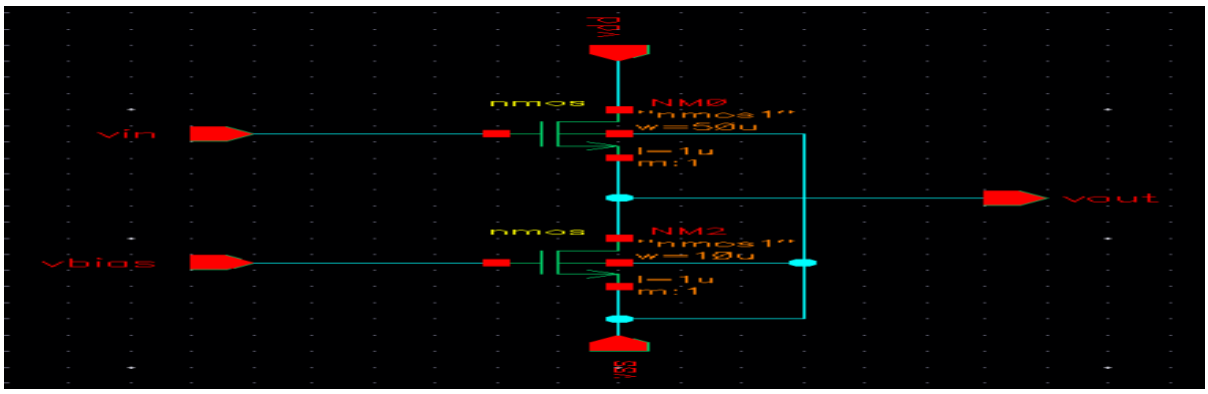


<u>Output Waveform</u>

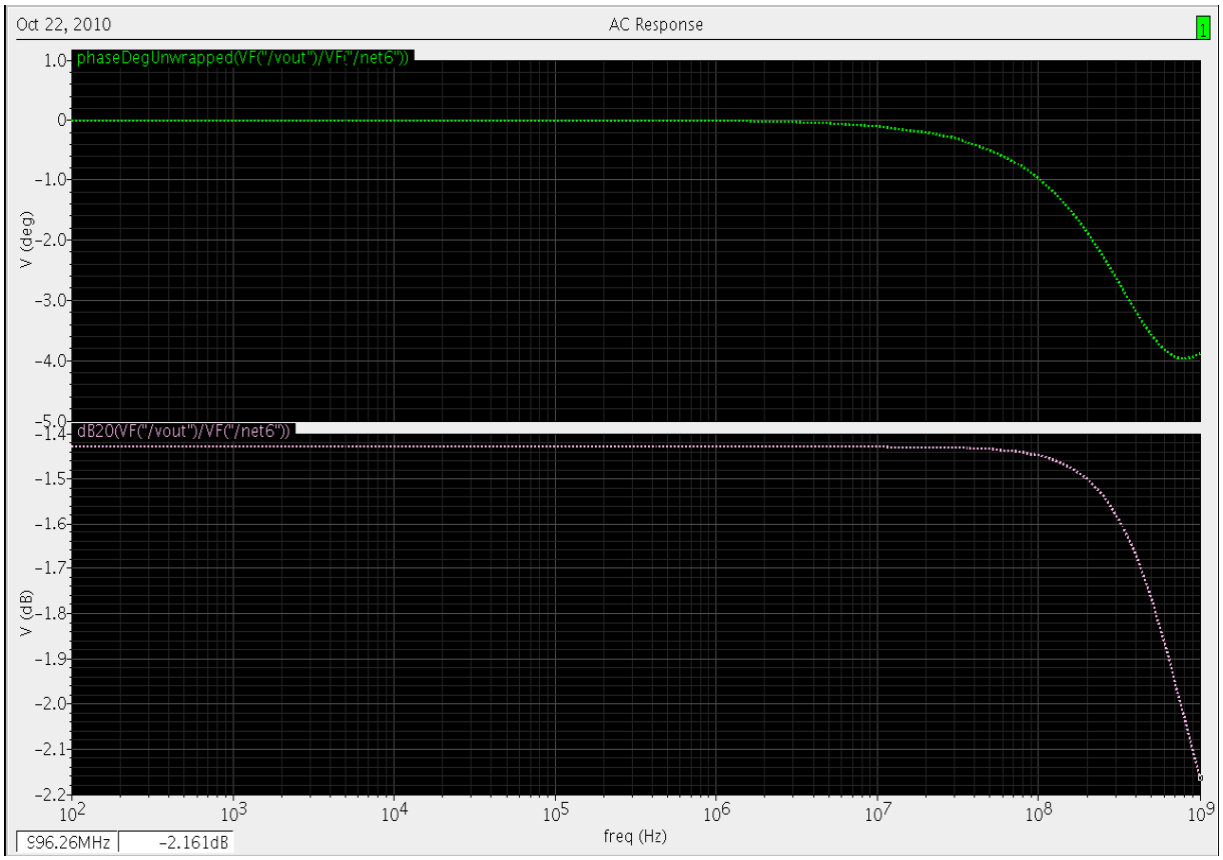


Common Drain Amplifier

In electronics, a common-drain amplifier, also known as a source follower, is one of three basic single-stage field effect transistor (FET) amplifier topologies, typically used as avoltage buffer. In this circuit (NMOS) the gate terminal of the transistor serves as the input, the source is the output, and the drain is common to both (input and output), hence its name.



Output Waveform



Common Gate Amplifier

In electronics, a common-gate amplifier is one of three basic single-stage field-effect transistor (FET) amplifier topologies, typically used as a current buffer or voltage amplifier. In this circuit the source terminal of the transistor serves as the input, the drain is the output and the gate is connected to ground, or "common," hence its name

PROCEDURE:

Schematic Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, sedit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Type the name of the cell and select view type as schematic.
6. Add nMOS and pMOS transistors and connect using wires.
7. Click on cell and generate the symbol.
8. Click on cell and select new view.
9. Type the name of the cell and select view type as schematic. Press "I" to instance the symbol.
10. Make connect between Vin, Vout, Vdd, Gnd and Vdc
11. Click on Setup Simulation Icon and add libraries.
12. Enable the Transient Fourier analysis. Provide simulation stop time 100 ns, Step time 1ns, and print time 1ns.
13. Save, run the simulation and check the Functionality of the Design in waveform window.

Layout Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, ledit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Provide the Cell name same as the Schematic design cell name.
6. Go to Tools> SDL Navigator > Load Netlist.
7. Then click on Layout tab enable Metal1 Layer and Mention the Size in Microns.
8. Tool > SDL Navigator> route> route all.
9. Go to file > Export > GDSII and Click on DRC icon in Ledit window
10. Load the Rule file path, load input layout file and click on run DRC
11. For LVS, load Netlist file from Inverter test bench folder and run LVS.
12. run PEX and click on Start RVE

RESULT

Exp No : 12	DESIGN AND SIMULATE A DIFFERENTIAL AMPLIFIER
Date :	

AIM:

To design and simulate a differential amplifier using Mentor Graphics EDA tool. Generate layout and post layout extraction for a 4 differential amplifier. Analyze Gain, Bandwidth and CMRR by performing Schematic Simulations.

APPARATUS REQUIRED:

1. PC with windows XP
2. Mentor Graphics EDA Tool

THEORY:

A **differential amplifier** is a type of electronic amplifier that multiplies the difference between two inputs by some constant factor (the differential gain). Many electronic devices use differential amplifiers internally. The output of an ideal differential amplifier is given by:

$$V_{\text{out}} = A_{\text{d}}(V_{\text{in}}^{+} - V_{\text{in}}^{-})$$

Where Vin+ and Vin- are the input voltages and Ac is the differential gain. In practice, however, the gain is not quite equal for the two inputs. This means that if Vin+ and Vin-are equal, the output will not be zero, as it would be in the ideal case. A more realistic expression for the output of a differential amplifier thus includes a second term.

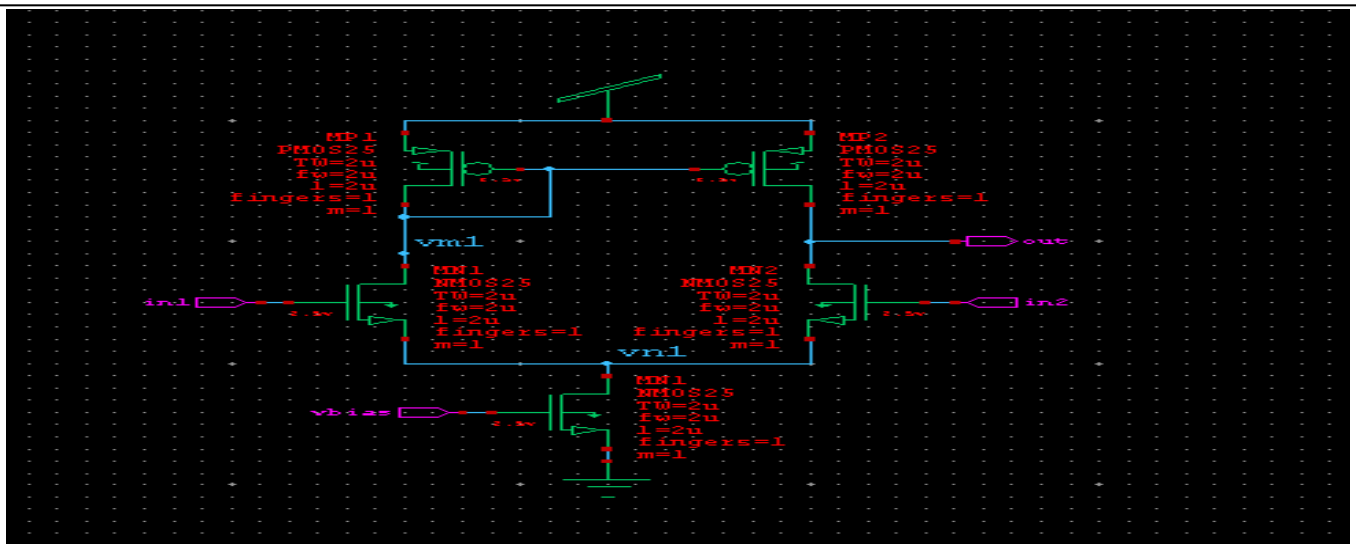
$$V_{\text{out}} = A_{\text{d}}(V_{\text{in}}^{+} - V_{\text{in}}^{-}) + A_{\text{c}}\left(\frac{V_{\text{in}}^{+} + V_{\text{in}}^{-}}{2}\right)$$

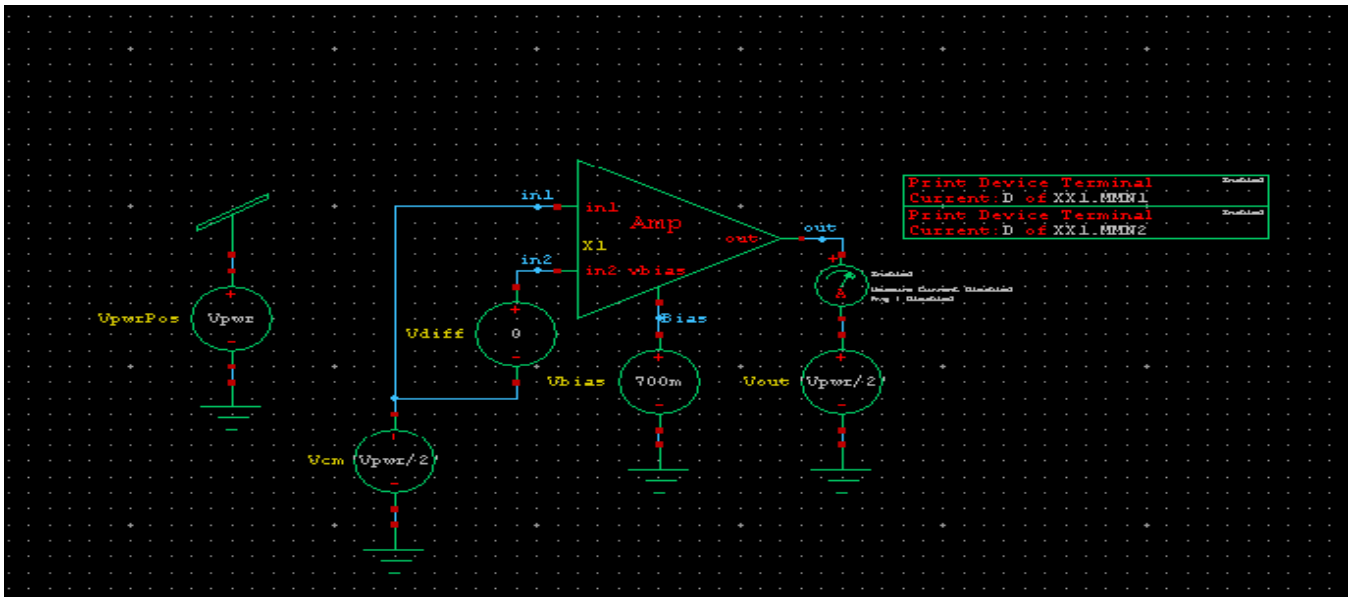
*A*c is called the common-mode gain of the amplifier. As differential amplifiers are often used when it is desired to null out noise or bias-voltages that appear at both inputs, a low common-mode gain is usually considered good.

The common-mode rejection ratio, usually defined as the ratio between differential-mode gain and common-mode gain, indicates the ability of the amplifier to accurately cancel voltages that are common to both inputs. Common-mode rejection ratio (CMRR):

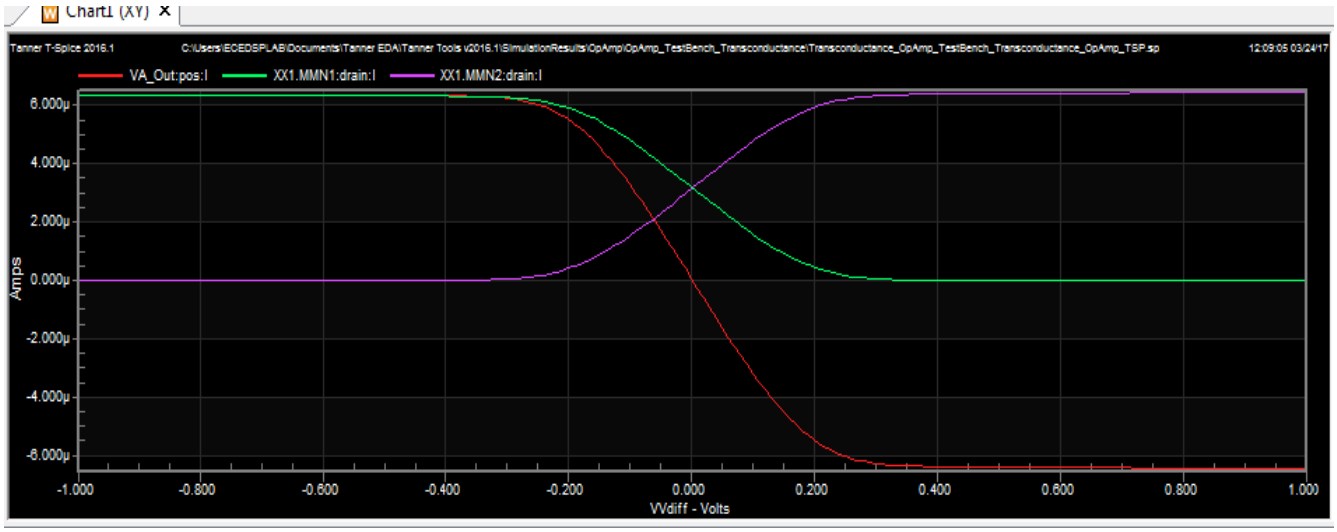
$$\text{CMRR} \triangleq \frac{A_{\text{d}}}{A_{\text{c}}}$$

Differential Amplifier Schematic Diagram





Output Waveform



PROCEDURE:

Schematic Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, sedit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Type the name of the cell and select view type as schematic.
6. Add nMOS and pMOS transistors and connect using wires.
7. Click on cell and generate the symbol.
8. Click on cell and select new view.
9. Type the name of the cell and select view type as schematic. Press "I" to instance the symbol.
10. Make connect between Vin, Vout, Vdd, Gnd and Vdc
11. Click on Setup Simulation Icon and add libraries.
12. Enable the Transient Fourier analysis. Provide simulation stop time 100 ns, Step time 1ns, and print time 1ns.
13. Save, run the simulation and check the Functionality of the Design in waveform window.

Layout Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, ledit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Provide the Cell name same as the Schematic design cell name.
   Go to Tools> SDL Navigator > Load Netlist.
6. Then click on Layout tab enable Metal1 Layer and Mention the Size in Microns.
7. Tool > SDL Navigator> route> route all.
8. Go to file > Export > GDSII and Click on DRC icon in Ledit window
9. Load the Rule file path, load input layout file and click on run DRC
10. For LVS, load Netlist file from Inverter test bench folder and run LVS.
11. run PEX and click on Start RVE

RESULT

Exp No : 13	DESIGN AND IMPLEMENT PRBS GENERATOR AND ACCUMULATOR
Date :	

AIM:

To design a PRBS generator and accumulator using Hardware Description Language, simulate it using Xilinx software and implement by Xilinx FPGA Spartan 3E kit.

APPARATUS REQUIRED:

1. PC with windows XP
2. Xilinx 12.1 software
3. Spartan 3E FPGA Kit

PROCEDURE:

Design Entry

1. Launch Xilinx by navigating to Xilinx ISE Design Suite 12.1 and select ISE Design Tools.
2. Create a New Project by going to the file menu and selecting "New Project".
3. Now give the project a name, select a location where the files will be saved. Set "Verilog" as the preferred language.
4. In the new window just opened, Select Verilog module from the left side and give it a name and click "Next".
5. In this window we can set a few parameters of our design as the type modeling we are going to perform, the entity name and the list of ports. Set them appropriately and Click Next.
6. Now the Verilog module we just created will open. Edit this file (actual design entry), enter the entity (already if not done) and the architecture.

Simulation

1. First select the "Verilog Test Fixture" in the design Hierarchy window.
2. Then in the process window expand ISIM Simulator and double click on "Simulate Behavioral Window Model". Wait till the simulation is complete and the simulation result window will open.

THEORY

A pseudorandom binary sequence (PRBS) is a binary sequence that, while generated with a deterministic algorithm, is difficult to predict and exhibits statistical behavior similar to a truly random sequence. PRBS generators are used in telecommunication, but also in encryption, simulation, correlation technique and time-of-flight spectroscopy.

Verilog Code for Accumulator

```
moduleupaccm(clk,reset,q,d);
input reset;input clk;input [5:0]d;output reg [5:0]q;
always @(posedgeclk or negedge reset)
begin
if(~reset)
q=4'b0000;
else
q=q+d;
end
endmodule
```

Testbench

```
moduletestaccum;
regclk;reg reset;reg [5:0] d;
wire [5:0] q;
upaccmuut ( .clk(clk), .reset(reset), .q(q), .d(d));
initial begin
clk = 0;
reset = 0;
d = 0;
#100;
reset = 1;
# 50 d=4'b0010;
#200 reset = 0;
#50 reset = 1;
end
always #50 clk = ~clk;
endmodule
```

Verilog Code for PRBS Generator

```
moduleprbsGenerator(rand,clk,reset);
inputclk,reset;output rand;wire rand;reg[3:0]temp;
always@(posedge reset)
temp<=4'hf;
always@(posedgeclk)
```

```
begin
if(~reset)
temp<={temp[0]^temp[1],temp[3],temp[2],temp[1]};
end
assign rand=temp[0];
endmodule
```

Testbench

```
modulemainprbs;
regclk,reset;wire rand;
prbsGeneratorpr(rand,clk,reset);
initial begin
forever begin
forever begin
clk<=0;#5clk<=1;#5clk<=0;
end
end
end
initial begin
reset=1;#12reset=0;#90reset=1;#12reset=0;
end
endmodule
```

RESULT

Exp No : 14	DESIGN AND SIMULATE A TRANSMISSION AND PASS TRANSISTOR GATE
Date :	

AIM:

To design and simulate a transmission gate and pass transmission gate using Mentor Graphics EDA tool. Analyze Gain, Bandwidth and CMRR by performing Schematic Simulations.

APPARATUS REQUIRED:

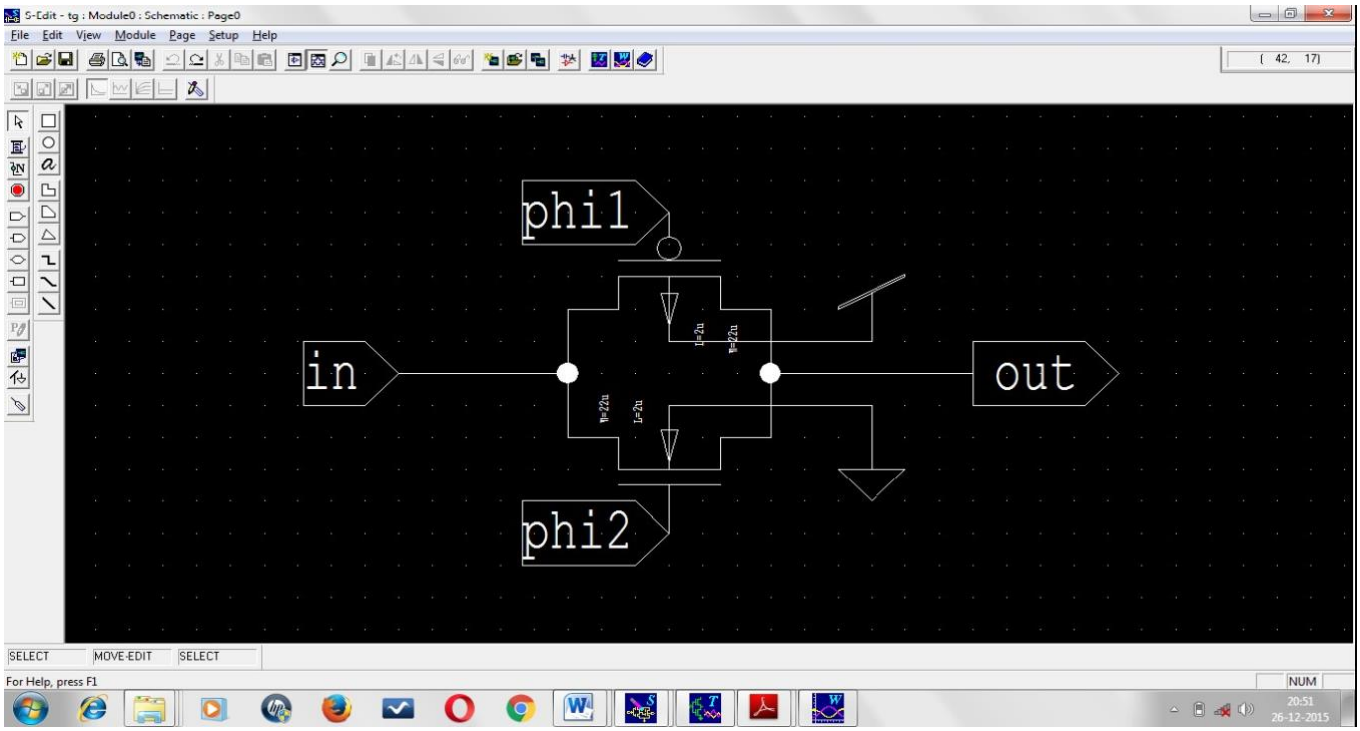
1. PC with windows XP
2. Mentor Graphics EDA Tool

THEORY:

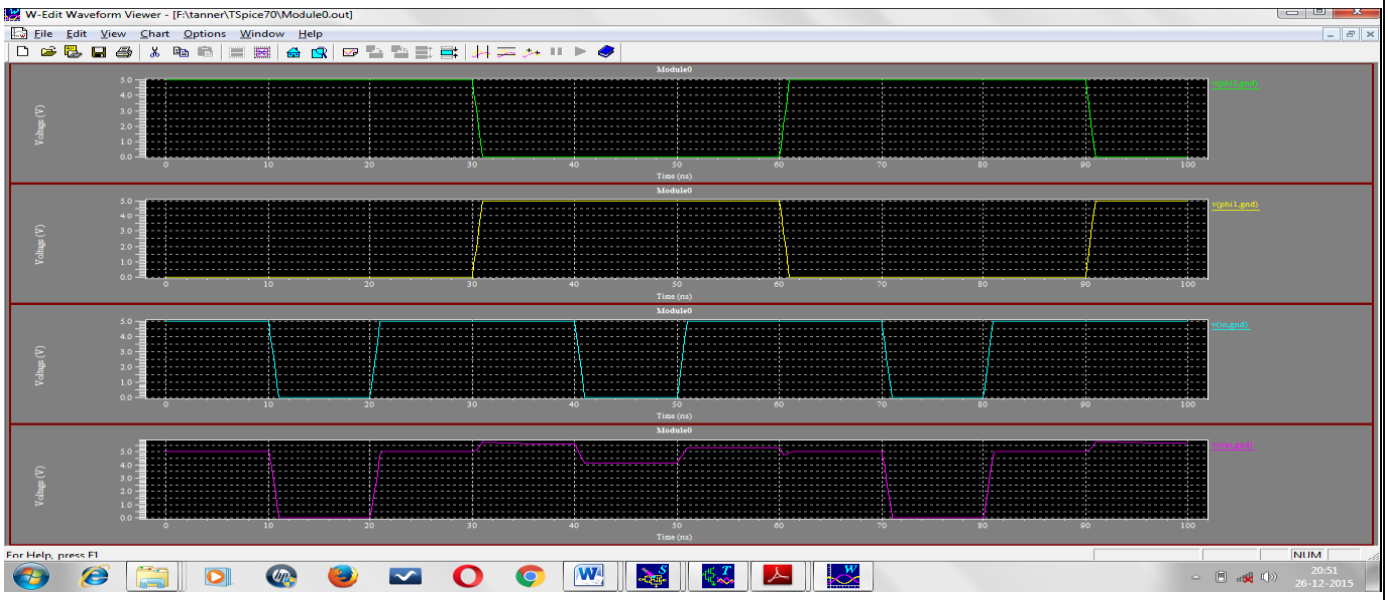
Transmission Gate

It's a parallel combination of pmos and nmos transistor with the gates connected to a complementary input. The disadvantages weak 0 and weak 1 can be overcome by using a TG instead of pass transistors. Working of transmission gate can be explained better with the following equation. An important advantage of TGs is that the reduction in the resistance because two transistors will come in parallel. When phi ='0' n and p device off, Vin=0 or 1, Vo='Z' When phi ='1' n and p device on, Vin=0 or 1, Vo=0 or 1 , where 'Z' is high impedance.

Transmission Gate Schematic Diagram



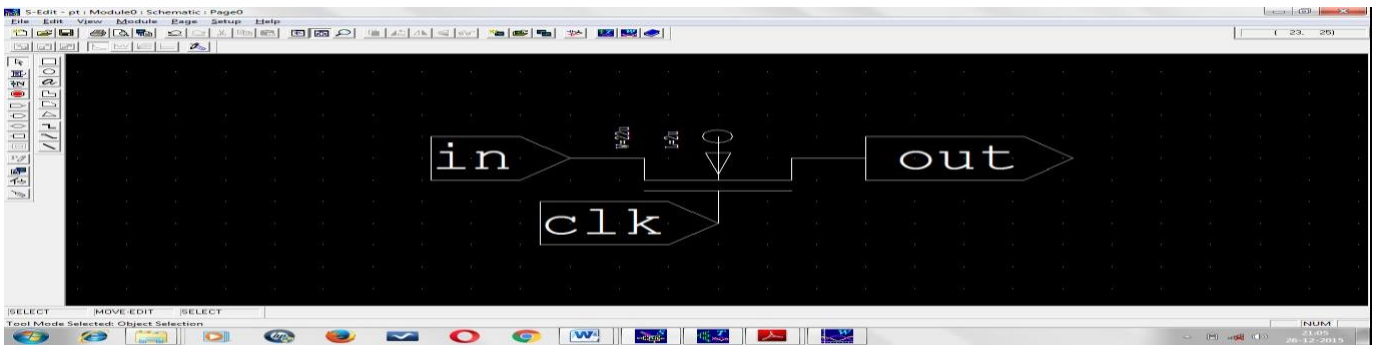
Output Waveform



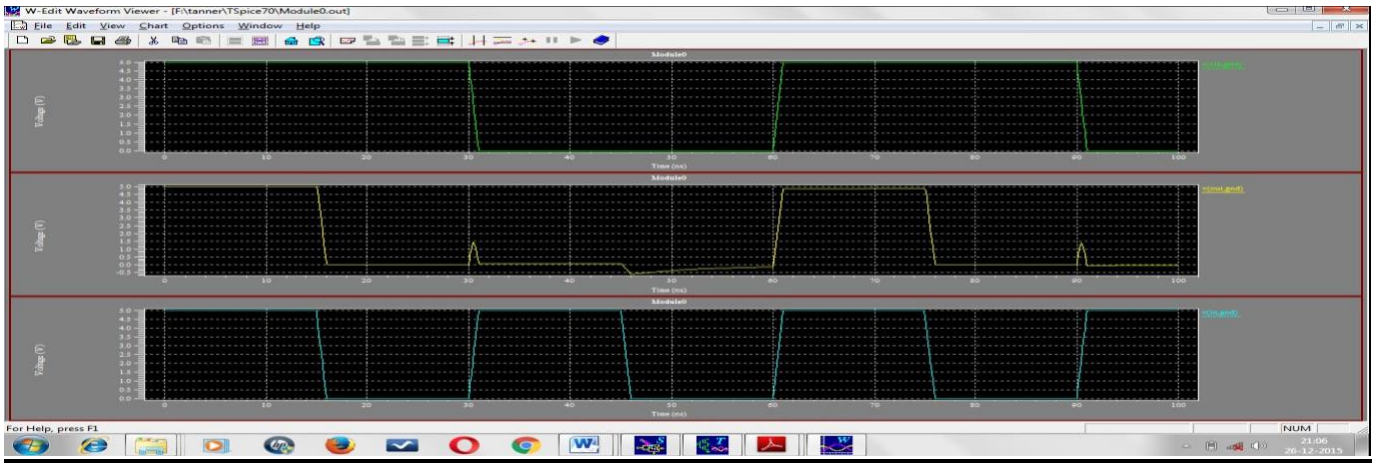
Pass Transistor Logic

When an nMOS or pMOS is used alone as an imperfect switch, then it is called a pass transistor. An nMOS transistor is an almost perfect switch when passing a 0 and thus we say it passes a strong 0. However, the nMOS transistor is imperfect at passing a 1. The high voltage level is somewhat less than VDD. We say it passes a degraded or weak 1. A pMOS transistor again has the opposite behavior, passing strong 1s but degraded 0s. The disadvantage with the pass transistors is that, they will not be able to transfer the logic levels properly.

Pass Transistor Logic Schematic Diagram



Output Waveform



PROCEDURE:

Schematic Design

1. Open in Terminal using csh, source /home/Setting/cshrc/hep.cshrc, sedit &.
2. Create new design, library path and name.
3. Right click on design name and add library.
4. Click on cell and select new view.
5. Type the name of the cell and select view type as schematic.
6. Add nMOS and pMOS transistors and connect using wires.
7. Click on cell and generate the symbol.
8. Click on cell and select new view.
9. Type the name of the cell and select view type as schematic. Press "I" to instance the symbol.
10. Make connect between Vin, Vout, Vdd, Gnd and Vdc
11. Click on Setup Simulation Icon and add libraries.
12. Enable the Transient Fourier analysis. Provide simulation stop time 100 ns, Step time 1ns, and print time 1ns.
13. Save, run the simulation and check the Functionality of the Design in waveform window.

RESULT