DEPARTMENT OF ELECTRONICS ENGINEERING

VLSI & COMMUNICATION SYSTEMS LAB

Lab Manual

Course code: ECC 583

[VLSI LAB: ROOM NO. 202,

COMMUNICATION LAB: ROOM NO. 215,

NEW ACADEMIC BUILDING]

VLSI MODULAR LAB EXPERIMENTS

COMMUNICATION MODULAR LAB EXPERIMENTS

EXPERIMENT 1: CMOS INVERTER

Objective: Design a schematic of a CMOS Inverter in Cadence Virtuoso Schematic Editor. Obtain the DC transfer characteristics and transient response for different widths and temperature conditions using parametric analysis.

Theory: A CMOS inverter contains a PMOS and a NMOS transistor connected at the drain and gate terminals, a supply voltage V_{DD} at the PMOS source terminal, and a ground connected at the NMOS source terminal, where V_{IN} is connected to the gate terminals and V_{OUT} is connected to the drain terminals. It is important to notice that the CMOS does not contain any resistors, which makes it more power efficient that a regular resistor-MOSFET inverter.As the voltage at the input of the CMOS device varies between 0 and 5 volts, the state of the NMOS and PMOS varies accordingly.

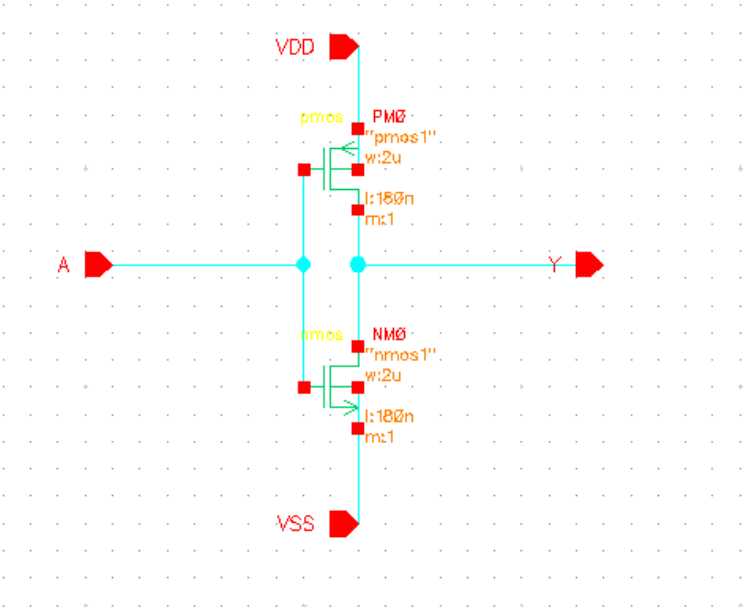


Figure1 (a) Inverter Schematic

Assignment:

Note: Use 180 nm CMOS model from GPDK available in the library. Assume maximum supply voltage for this technology is 1.8 V. The nominal conditions are: $\mathbf{w_n}$ = 1 µm, $\mathbf{w_p}$ = 2 µm and channel length is 180 nm, Temperature is 27 ˚C.

1. Draw the schematic of a CMOS inverter. Obtain its DC transfer characteristics and Transient response for nominal conditions.

2. Do the parametric analysis for different W/L ratios of the inverter circuit. [Hint: Vary $\mathbf{w_p}$ from 2 µm to 10 µm in step size of 1 µm]

3. Do the parametric analysis for different temperature conditions. Take the temperature values as -40˚C, 27˚C, and 100˚C.
    i) Plot the DC transfer characteristics and tabulate transition voltages for nominally sized inverter.
    ii) Tabulate the rising and falling delay for nominally sized inverter. [Hint: Use expressions in calculator]

Comment on the results obtained.

Observation table:

W_p(microns)	V_{inv}(V)

Conclusion:

Practice Questions:

1. What is noise margin of a CMOS inverter?
2. Explain sizing of an inverter?
3. Draw and explain the transfer curve of an inverter?

EXPERIMENT 2: Layout Design

Objective: Perform Layout design of a CMOS Inverter in Cadence Virtuoso Layout Editor. Perform DRC, LVS and RCX. Obtain the DC transfer characteristics and transient response for different widths and temperature conditions and compare with the values obtained in Experiment 1.

Theory: Design rule checking or check(s) (**DRC**) is the area of electronic design automation that determines whether the physical layout of a particular chip layout satisfies a series of recommended parameters called design rules.

The Layout Versus Schematic (**LVS**) is the class of electronic design automation (EDA) verification software that determines whether a particular integrated circuit layout corresponds to the original schematic or circuit diagram of the design.

Pre Layout simulation: It is the functional verification of the design without including the parasitics, which gets added to the design when the design is physically implemented or laid out.

Post Layout Simulation: The parasitic capacitances extracted according to how layout is designed might be critical in affecting the actual performance of the design. In order to get an idea of how the design would work from the layout, you should perform a post-layout simulationfrom the extracted view.

LAYOUT VIEW OF AN INVERTER

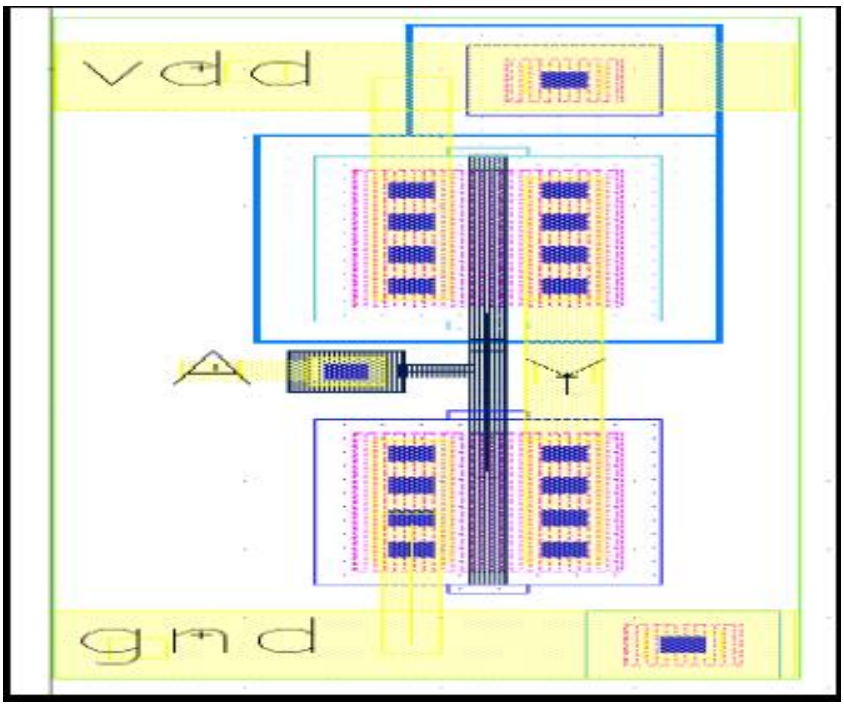


Figure 2: Inverter layout Schematic

Observation:

Calculation:

Pre-Layout Simulation value of rise and fall time=_____ps

Post- Layout Simulation value of rise and fall time= ____ps

Pre-Layout Simulation value of propagation Delay = ____ps

Post-Layout Simulation value of propagation Delay = ____ps

.

Conclusion:

Practice Questions:

1. Give five important design technique to follow when doing a layout design of digital circuit?
2.What do you mean by FEOL and BEOL process?
3. What should be the n-diffusion and p-diffusion layer in lambda rule?

EXPERIMENT 3:

Objective: Extraction of Logical effort and parasitic delay of a CMOS inverter.

Theory: Logical effort of a logic gate is defined as the ratio of its input capacitance to that of an inverter that delivers equal output current. The method of logic effort is an easy way to estimate delay in a CMOS circuit. We can select the fastest candidate by comparing delay estimates of different logic structures. The method also specifies the proper number of logic stages on a path and the best transistor sizes for the logic gates.

Parasitic delay is the delay due to intrinsic delay of gate mostly the drain capacitance. It is independent of output load and sizing.
The delay incurred by a logic gate is comprised of two components:

- A fixed part called as the parasitic delay, **p**.
- A part that is proportional to the load on the gate's output, called the effort delay or stage effort, **f**.

d = f + p

The effort delay depends on the load and on properties of the logic gate driving the load. We introduce two related terms for these effects:

- The logical effort, **g** captures properties of the logic gate.
- The electrical effort **h** characterizes the load.

f = gh

Therefore, **d = gh +p**.

Multi-stage Inverter with a FAN-OUT-4

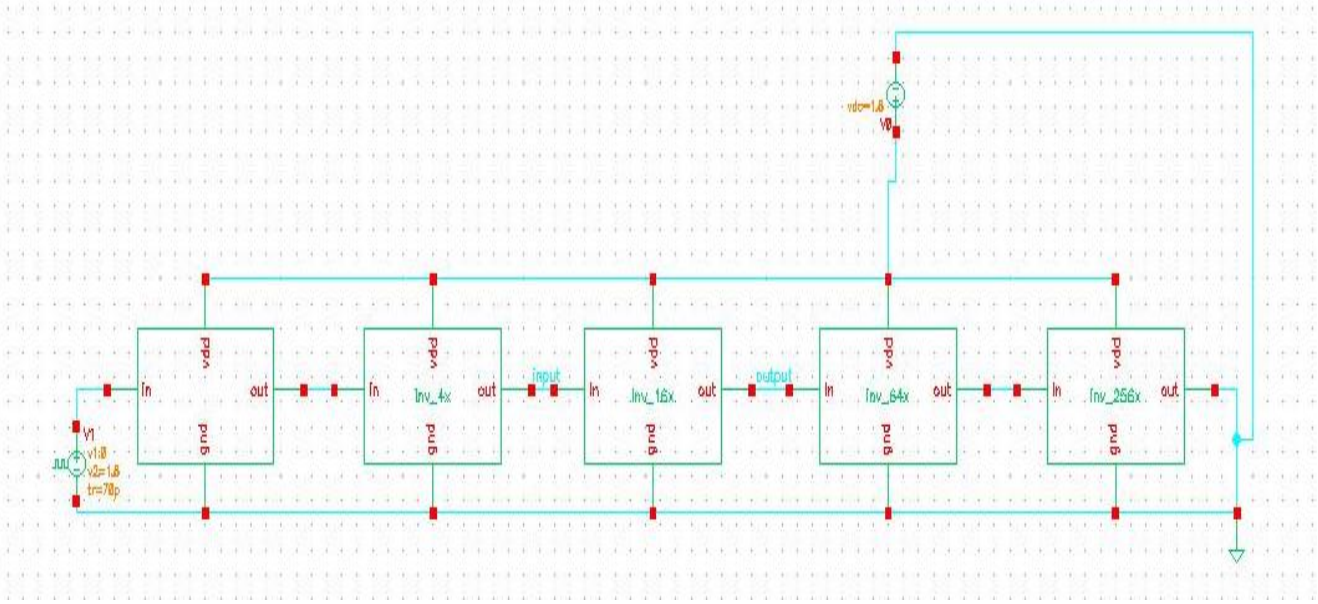


Figure 3(a): Schematic Representation of Multi-stage inverter logic network with a fan-out- 4.

Observation:

Rising propagation delay time (t_{pdr1}) = ____ps
Falling propagation delay time (t_{pdf1}) = ____ps

Calculation:

FO4 delay = $(t_{pdr1} + t_{pdf1})/2 = 5\tau$

τ= time constant in ps

Multi-stage Inverter with a FAN-OUT-1 :

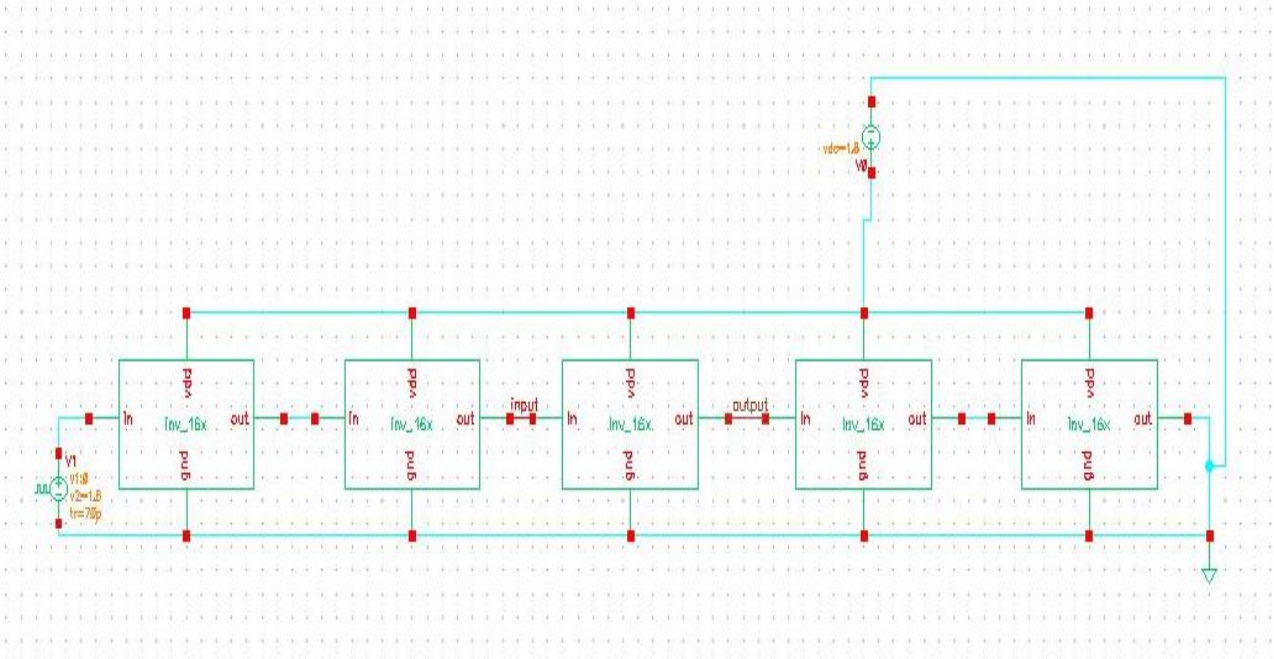


Figure 3(b): Schematic Representation of Multi-stage inverter logic network with a fanout-of-1

Observation:

Rising propagation delay time (t_{pdr2}) = ___ps
Falling propagation delay time (t_{pdf2}) = ___ps

Calculation:

According to **d = gh +p**, now we have two equations,

$$\frac{t_{pdr1} + t_{pdf1}}{2} = gx4 + p$$

$$\frac{t_{pdr2} + t_{pdf2}}{2} = gx1 + p$$

By solving above 2 equations,

g =
p =

Conclusion:

Practice Questions:

1. What is propagation delay?
2.What happens to delay if load capacitance is increased?
3.Why is rise time greater than fall time?

EXPERIMENT 4: D-latch and D-Flip Flop.

Objective: Design and characterization of D-Latch and D-FlipFlop in CMOS Technology.

Theory:

D-latch: Latch is an electronic device that can be used to store one bit of information. The D latch is used to capture, or 'latch' the logic level which is present on the Data line when the clock input is high or low depending on the type of level triggering.

High level triggering: If the data on the D line changes state while the clock pulse is high, then the output, Q, follows the input, D. When the CLK input falls to logic 0, the last state of the D input is trapped and held in the latch.

Low level triggering: If the data on the D line changes state while the clock pulse is low, then the output, Q, follows the input, D. When the CLK input rises to logic 1, the last state of the D input is trapped and held in the latch.

D-flipflop: The working of D flip flop is similar to the D latch except that the output of D Flip Flop takes the state of the D input at the moment of a positive edge at the clock pin (or negative edge if the clock input is active low) and delays it by one clock cycle. That's why, it is commonly known as a delay flip flop. The D flipflop can be interpreted as a delay line or zero order hold. The advantage of the D flip-flop over the D-type "transparent latch" is that the signal on the D input pin is captured the moment the flip-flop is clocked, and subsequent changes on the D input will be ignored until the next clock event.

Assignment:

1. To Design a D latch using transmission gates as shown in fig.4 (a).

2. To Design a D flip-flop using transmission gates as shown in fig.4 (b).

Conditions for Modelling:

- 180 nm CMOS model from GPDK
- w_n = 1μm, w_p = 2μm, channel length = 180 nm, temperature = 27 ˚C.

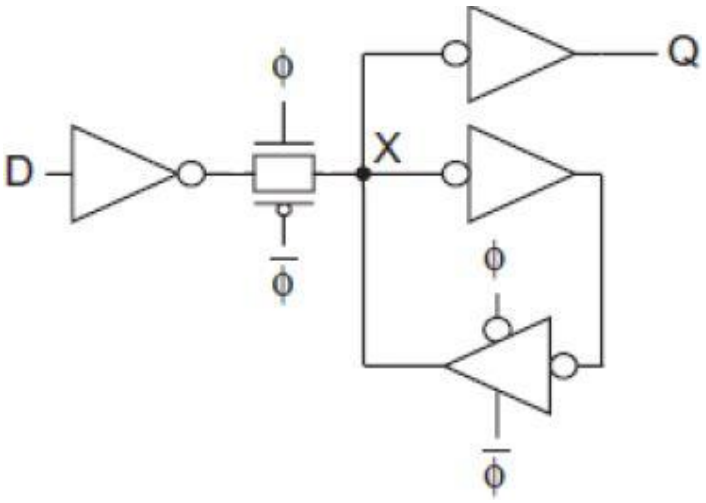


Figure 4(a): D-latch

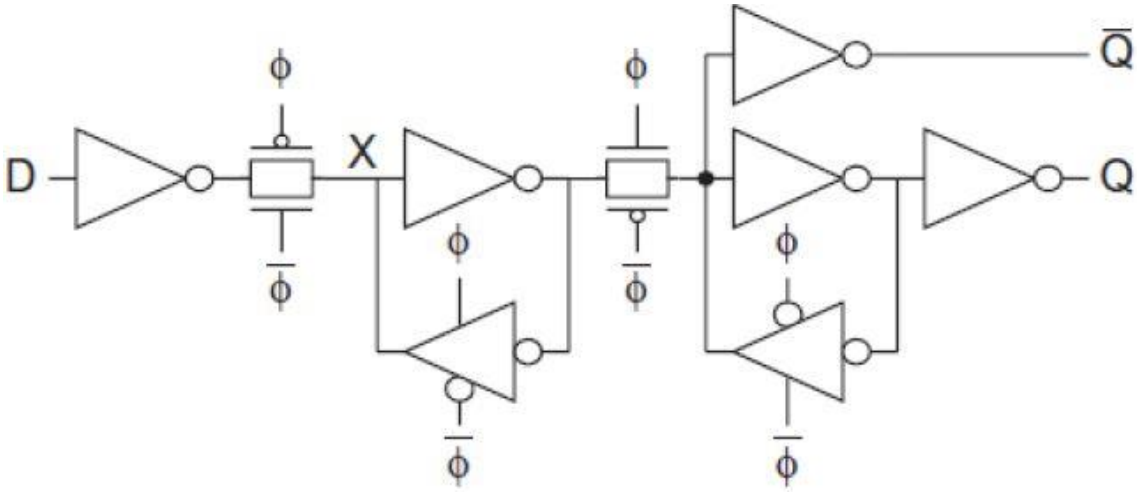


Figure 4(b): D-flipflop

Observation:

Clock Parameters: Rise time =___ ps
Fall time = ___ps,
Pulse width = ___ ns,
Period = ___ ns,

Input Parameters: Rise time = ___ ps
Fall time =___ps,
Pulse width = ___ ns,
Period =___ ns,

Output Parameters: Rise time = ___ ps
Fall time =___ps,
Propagation delay = ___ ns,

Conclusion:

Practice Questions:

1. What is the difference between D latch and D flip-flop?
2. Write down the characteristic equation of D flip-flop?
3. Describe the operation of a negative edge triggered D flip-flop?

Experiment. 5: Small signal parameters of MOSFET.

Objective: Extraction of small signal parameters of MOSFET using CADENCE Virtuoso Schematic Editor.

Theory: Applying design equation to every possible device in a circuit is difficult, particularly when there is a large number of variables involved and more devices are present. So, in order to derive such analytic design equations without losing much accuracy and to develop qualitative understanding of the circuit behavior, circuit designers use a special technique called small signal modeling. This modeling also takes care of the short channel effects of MOSFET, like Channel length modulation, body effect, sub-threshold V_{ds} conduction. Figure 5(a) shows the small-signal model for an n-channel MOSFET.

In this model, the complex MOSFET structure is represented as a set of transconductances (g_m, g_{mb}, and $1/r_o$) and the parasitic junction capacitances between each terminal. The dependent current source g_mV_{gs} reflects how drain current is controlled by gate source voltage. The channel length modulation effect, which corresponds to the dependence on V_{ds} voltage source, is represented as r_o resistance. The current-source $g_{mb}V_{bs}$ reflects the body effect on the MOSFET.

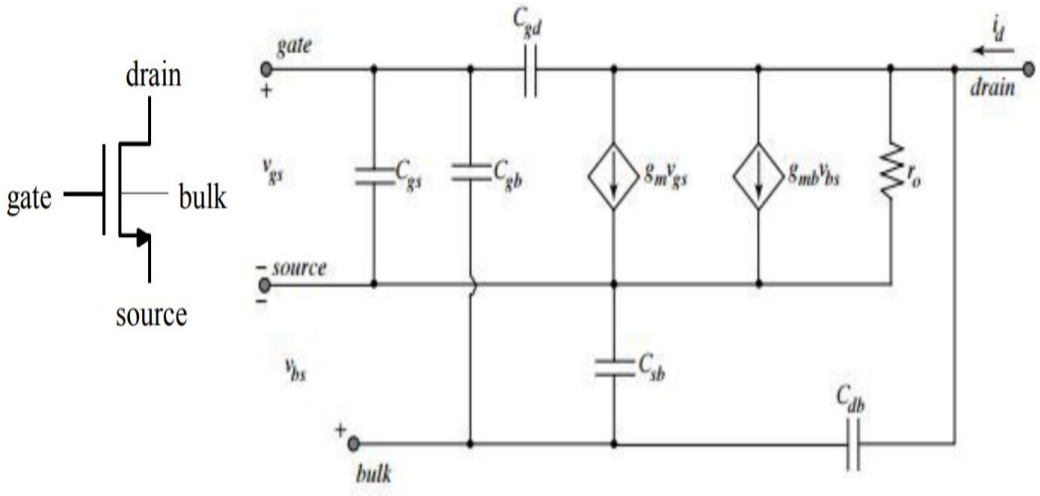


Figure 5(a): small signal model of NMOS

The parasitic junction capacitances, shown in Fig 5(a) affects the behavior of the MOSFET, mainly on high-frequency operation. Similar to this model, the p-channel MOSFET is modeled as shown in Figure 5(b).

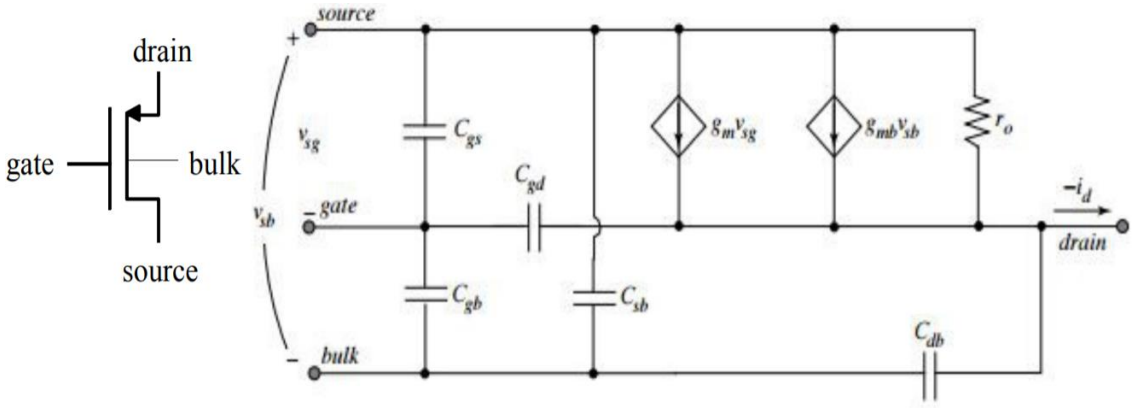


Figure 5(b): small signal model of PMOS

Assignment:

1. Perform the DC analysis of MOSFET (PMOS and NMOS).
2. Obtained the value of Small signal parameters from result browser.

Observation:

Mention all the small signal parameters along with their value obtained.

Conclusion:

Practice Questions:

1. Why do we use small signal model?
2.What is the small signal transconductance of MOSFET?
3.What is small signal gain?

Experiment No.-1

Name of the Experiment : Generation of Pseudo Random sequence using LFSR

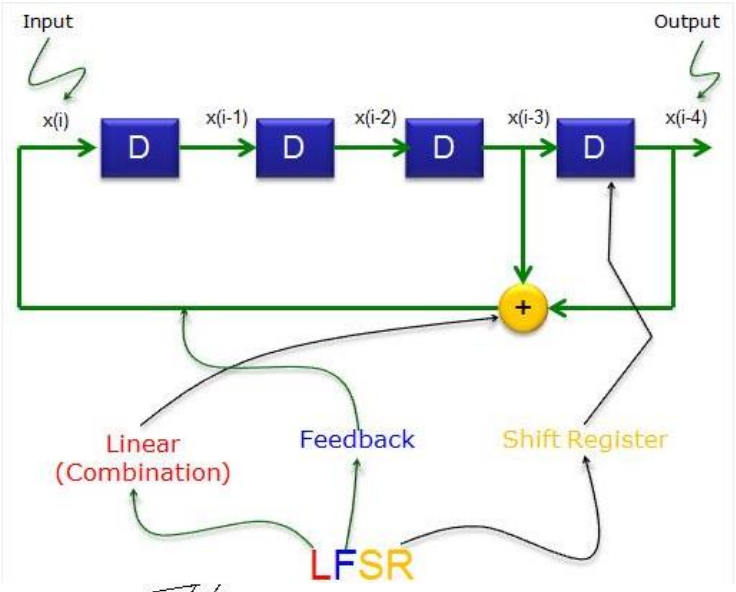
Objective:

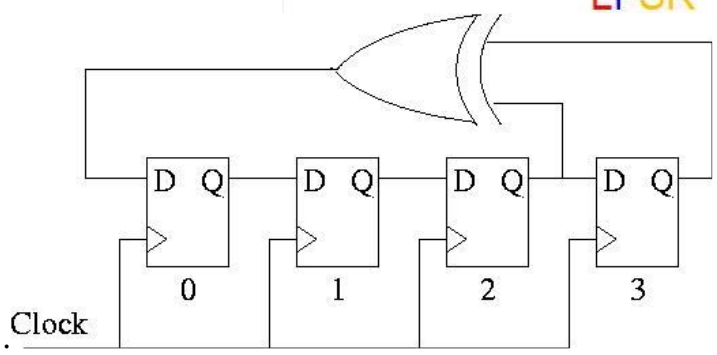
(i) Understanding the mathematics behind the theory of PN sequences
(ii) Verification of properties of maximal length PN sequences
(iii) Analysing effects of seed and tap-positions.

Theoretical concepts :

Pseudo-Noise (PN) or Pseudo-Random (PR) sequences are a stream of periodic binary sequences in a known pattern used in simulation and testing of communication systems. It is a key element in spread spectrum based wireless systems used in cellular phone technologies, cordless phones, WLAN, Bluetooth and GPS. They are also commonly used to generate random noise that is approximately "white", and has applications in scrambling and cryptography as well. The qualifier "pseudo" implies that the sequence is not truly random, but it is periodic with a (possibly large) period, and exhibits some characteristics of a random white sequence within that period.

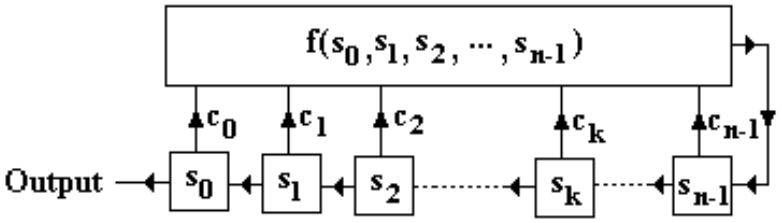
PN sequences are conventionally generated from Linear Feedback ShiftRegister (LFSR) which is a system that generates bits from a register and a feedback function . Herer the output of two or more intermediate steps is linearly combined and fed back to the input value. After several iterations, the register returns to a previous state already known and stats again in a loop , the number of iterations of which is called a period. When the period is long enough, it allows the generation of PN sequences. The plus here denotes the XOR operation.





The background mathematics:
The initial state of registers is known as the seed or the "starting value". For an n-stage register, D(n-1) stores the LSB of the seed, and D(0) stores the MSB of the seed. For example, for n = 9, if the seed is 10 (binary form 1010), the initial state in register is {0 0 0 0 0 1 0 1 0}. We can characterize the LFSRs by defining its ***characteristic polynomial*** with coefficients c_0 to c_{n-1}

$$f(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} + x^n = \sum_{i=0}^{n} c_i x^i$$



For instance, if the polynomial is x^9+x^5+1, therefore, n=9 and, $c_9=1$, $c_5=1$ and all others are zero.

- If the state of the shift register is all zero at any time, it remains so for all time. We need to ensure that this never happens (we start with a non-zero value). (See how you do this in the circuit)
- If the state ever remains unchanged from one clock cycle to the next, it remains the same forever.
- The sequence must be periodic (since there are at most 2^n -1 states). Since all zero state is not allowed, the period of the output sequence can be at most 2^n -1. A feedback shift register that generates a sequence of this period is said to be of maximal length.

Some Facts and Definitions From Algebra

1. Every polynomial f(x) with coefficients in GF(2) having f(0) = 1 divides $x^m + 1$ for some m. The smallest m for which this is true is called the ***period*** of f(x). see the mathematical definition of Galois Field (GF)
2. An ***irreducible*** (can not be factored) polynomial of degree n has a period which divides $2^n - 1$.
3. An irreducible polynomial of degree n whose period is $2^n - 1$ is called a ***primitive polynomial***

Theorem: *A LFSR produces a PN-sequence if and only if its characteristic polynomial is a primitive polynomial*

Ex: Let us say that the characteristic polynomial of an LFSR with n = 4 is: $f(x) = x^4 + x^3 + x^2 + 1 = (x + 1)(x^3 + x + 1)$ and so is not irreducible and therefore not primitive.

Ex: $f(x) = x^4 + x^3 + x^2 + x + 1$ is a monic irreducible polynomial since it has no linear factors and remainder x + 1 when divided by $x^2 + x + 1$. However, $x^5 + 1 = (x + 1) f(x)$ and so it has period 5 and is not primitive.

Ex: $f(x) = x^4 + x^3 + 1$ is a monic irreducible polynomial over GF(2). To find its period, we have to determine the smallest m so that f(x) divides $x^m + 1$. Clearly, m > 4, also, by 2) above, the period divides $2^4 - 1 = 15$, thus it must be either 5 or 15. By trying the possibilities we get $x^5 + 1 = (x+1)(x^4 + x^3 + 1) + (x^3 + x)$
$x^{15} + 1 = (x^{11} + x^{10} + x^9 + x^8 + x^6 + x^4 + x^3 + 1)(x^4 + x^3 + 1)$

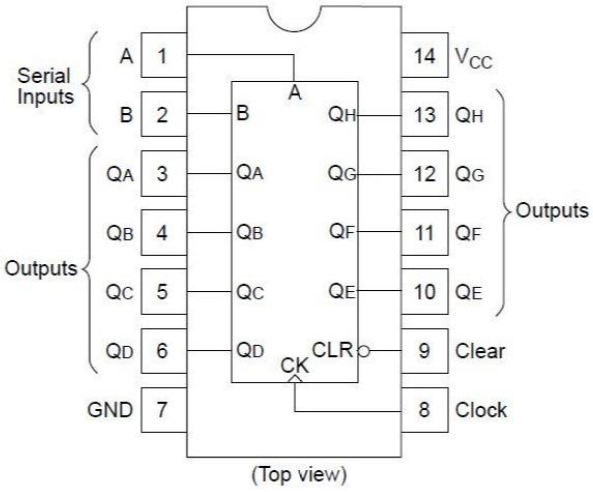
Thus, f(x) has period 15 and so, is a primitive polynomial.

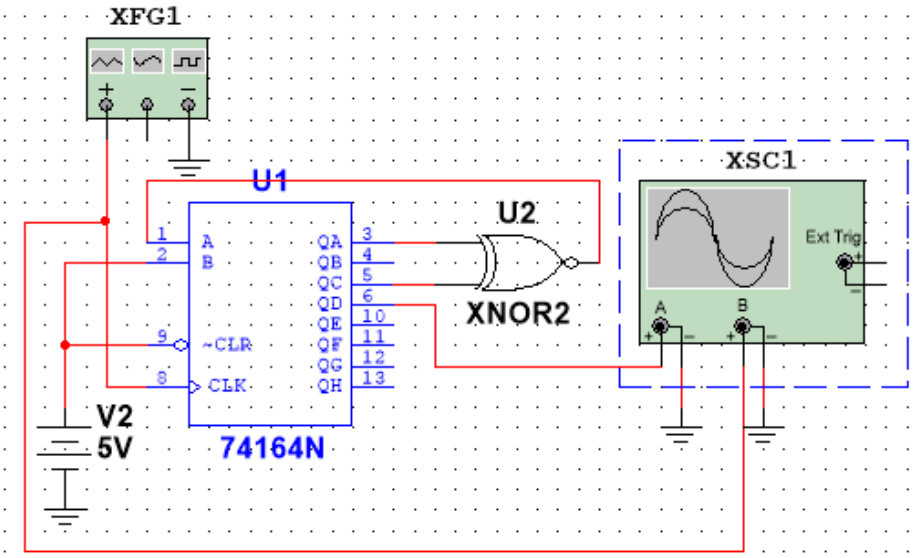
Search the internet for tables of primitive polynomials and try to interpret their meaning.

Procedure of the Experiment

Use any circuit simulation software or hardware to realize an LFSR selecting a suitable IC such as 74164N from the component Library and making the connections as shown . See the uploaded video demonstration of this. When the LFSR is clocked, it will generate a pseudorandom pattern of 1s and 0s. Note that the only signal necessary to generate the test patterns is the clock.

Pin Configuration of IC74164N:





Circuit diagram to generate required PN sequence.

Observation Table:

1. Take initial sequence as 0100, 0101 and 0110.
2. Find the output after every clock.
3. Verify if the output is repeated to the starting value after 2^n-1 clock.

Table I

clock	Q_A	Q_B	Q_C	Q_D	Output

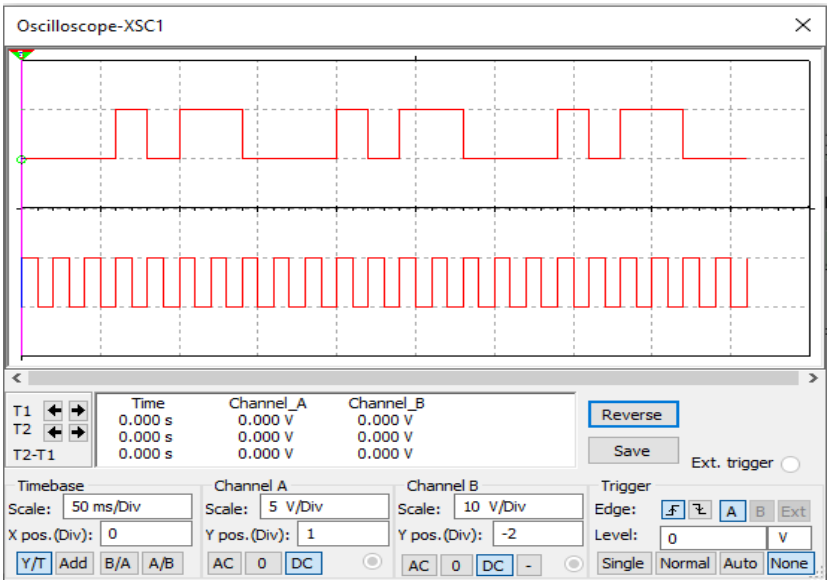


Figure 6: Generated PN sequence in Multisim software.

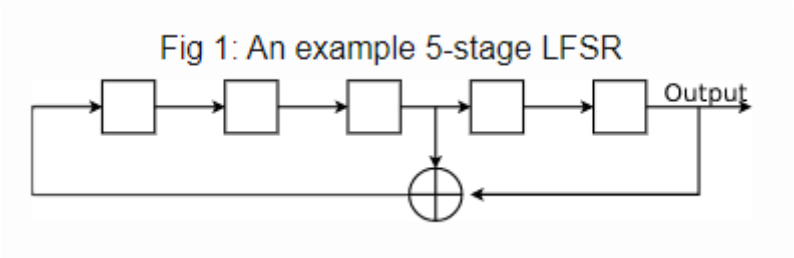
Task #1. Work out the PN sequence output generated with the following conditions : (Verify both by calculation and circuit if they are maximal length)

(i) n = 4, $c_0 = c_2 = c_3 = 1$, $c_1 = 0$ with initial state (0,1,1,0)

(ii) n = 4, $c_0 = c_3 = 1$, $c_1 = c_2 = 0$ and starting state 0,0,0,1

Note that you are XOR'ing the tap and the last bit, to produce the next bit that will be input back into the register.

Task #2 For the 5 stage LFSR shown, what will be the tap sequence? (binary as well as polynomial?) will it produce a maximal length sequence? Show all the states with initial seed as 0,0,0,0,1 . Does the initial stae decide if a ML sequence will be produced?



Task # 3 Find out some interesting applications of PN sequences and LFSRs other than what is mentioned in this note and write a short not within 10 sentences on that application.

Experiment No 2

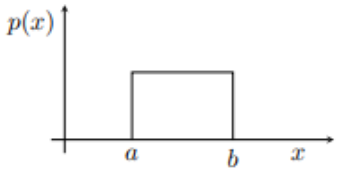
Random numbers and Noise generating functions in MATLAB

For use in numerical computing applications and communication simulations Matlab has built-in functions capable of producing pseudorandom numbers . The basic suite of random-number-generating functions includes rand randi and randn

The function rand generates pseudorandom numbers with a **uniform distribution** over the range of (0, 1). This is different from the function **randi** which generates pseudorandom integers drawn from a discrete uniform distribution on a specified interval min to max. Uniform distributions imply that all numbers are equally probable.

Uniform Distribution Let x be a random variable with PDF

$$p(x) = \begin{cases} \frac{1}{b-a} & a \le x \le b \\ 0 & \text{otherwise} \end{cases}$$



TASK #1 : Try to generate using rand a (i) 5 X 5 random matrix of channel gains (MIMO channel) a (ii) 1 X 5 matrix of different multipath channel gain) : can you make them complex gains? `Hint:` Try the command rand +1i*rand

Is there any difference between rand (3) and rand (1, 3). (iii) How can you generate them in a given interval: say between 5 to 10? See if you can do this

Generate a RV X uniformly distributed between [0,1], how can you change it to another RV which is uniformly distributed between [-1,1],

Using the rand function generate numbers between [-5 5]

Hint: Use the fact that distribution of a uniform random variable does not change under addition and multiplication of constants.

Can you realize the function of randi using rand? Hint: use the floor function.

Generate 15 random integers from 5 to 20 using randi function and also using rand and floor.

The randi function generates a matrix of pseudorandom integers over a specified range. The command **randi(x,m,n)** creates a matrix of random integers of size m x n in a range from 1 to x. Unlike rand and randn, a parameter specifying the range must be entered before the dimensions of the matrix. Try out generating a 3 x 5 matrix of random integers in the range of [1, 20]

The function randn generates psueudorandom numbers with a normal (Gaussian) distribution with mean zero and unit variance, abbreviated as $N(0,1)$. This distribution is quite common in a wide variety of scientific, mathematical, and engineering applications.

The theoretical PDF of Gaussian random variable is given by

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

Matlab's randn function is designed so that the mean is always (approximately) zero and the variance is (approximately) unity. Check the difference between randn, randn(n) and randn(n,m)

Consider a signal defined by w = a randn + b; that is, the output of randn is scaled and offset. What are the mean and variance of w? A common use of this function is to create a vector of normally distributed values with a specified mean and variance. What values must a and b have to create a signal that has mean 1.0 and variance 4.0?

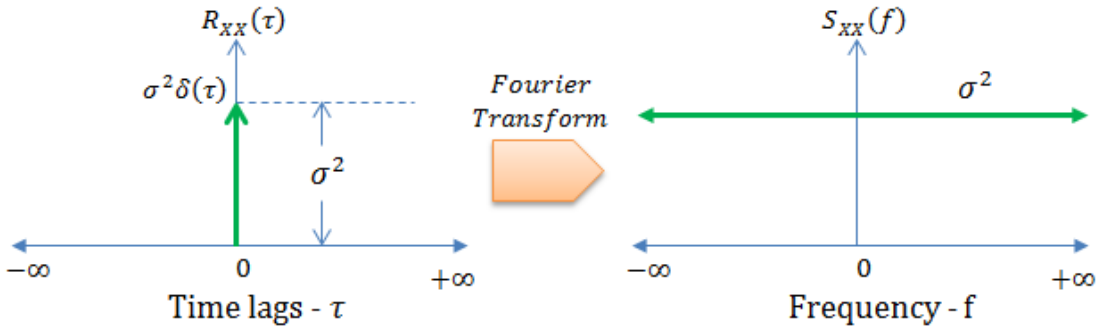
Note: If X is a random variable with zero mean and unity variance, then (aX + b) is a random variable with mean b and variance a^2. This follows from basic probability theory.

TASK #2 Generate 500 numbers which are normal distributed having variance 9 and mean 5

Test the actual value of the mean and variance (or standard deviation) using the command var or std. Does it depend on the length of the sequences generated? Test by generating 600, 700, 800 …1000 data.

Learn how to use the awgn function in MATLAB . This is a very useful function for BER analysis and channel modelling AWGN (Additive Gaussian White Noise): make sure you understand the meaning of the acronym: A W and G very clearly. A white noise signal (process) is constituted by a set of independent and identically distributed (i.i.d) random variables. A white process is seen as a random process composing several random variables following the same Probability Distribution Function (PDF). This condition is called "identically distributed" condition The individual samples are "independent" of each other In discrete sense, the white noise signal constitutes a series of samples that are independent and generated from the same probability distribution. For example, you can generate a white noise signal using a random number generator in which all the samples follow a given Gaussian distribution. This

is called White Gaussian Noise (WGN) or Gaussian White Noise. Why Additive?



The spectral properties of ACF and PSD of White Noise

TASK #3

Generate a Gaussian white noise signal of length 10,000 using the randn function in Matlab and plot it. Let's assume that the pdf is a Gaussian pdf with mean 0 and standard deviation $\sigma = 2$. Plot the histogram of the generated noise signal and verify the histogram by plotting against the theoretical pdf of the Gaussian random variable.

Matlab communication toolbox has an inbuilt function named **- awgn()** with which one can add an Additive Gaussian White Noise to obtain the desired Signal −to − NoiseRatio (SNR). The main usage of this function is to add AWGN to a clean signal (infinite SNR) in order to get a resultant signal with a given SNR (usually specified in dB). This usage is often found in signal processing/digital communication applications. For example, in Monte Carlo simulations involving modeling of modulation/demodulation systems, the modulated symbols at the transmitter are added with a random noise of specific strength, in order to simulate a specific Eb/N0 or Es/N0 point.

See the function y = awgn(x,snr) and explore how the resulting signal y is guaranteed to have the specified SNR.

TASK #4

Test whether the generated random numbers are actually uniformly/ Normally distributed or not by plotting their histogram; learn how the command hist works for plotting the histogram.

```
subplot(1,2,1);hist(randn(1000,1)); title('Normally Distributed');
subplot(1,2,2);hist(rand(1000,1); title('Uniformly Distributed');
```

TASK #5 OPTIONAL: Try to read what SEED of the PR number generators is (how can you generate the same random number sequence every time you call the rand/randn /randi functions

Experiment 2- (Part-B)

Practice the following codes that makes you more familiar to use the random noise functions

- SNR in dB = 10 $\log_{10}$ (σ_s^2/σ_n^2); where σ_s^2:signal variance, σ_n^2: noise variance
- Given signal, s(t), find σ_s^2
- Compute required σ_n^2
- Generate noise signal, n(t) = σ_nN(0,1), where N(0,1) is a Normally (Gaussian) distributed random variable with Zero mean and Unit variance
- Message signal with desired SNR, m(t) = s(t) + n(t)
- Synthesize a 1 second duration Sharp sinusoidal tone (466.16 Hz) sampled at 8 kHz.
- Plot this waveform; observe and listen using the *sound* command in Matlab.
- Corrupt this signal with a Gaussian noise source to get an SNR of 10 dB. Observe, listen and save waveform as before.
- Synthesize 1 cycle of the noisy waveform.

```
%Specify SNR
snr=10;

%Generate A sharp signal
t=[0:1/8e3:1.0]';
s = 0.5*sin(2*pi*466.16*t);
sound(s);

%Compute signal variance
var_s = cov(s);

%Calculate required noise variance
var_noise=var_s/(10^(snr/10));

%Generate noise
n=sqrt(var_noise)*randn(length(s),1);
sound(n);

%Add signal to noise and generate message
m=s+n;
sound(m);
```

Experiment 3: Signal Modulation and Reconstruction Using Hilbert Transform

Objective: Modulate a signal and recover the baseband signal from the Hilbert transform of the modulated signal.

```
fs = 600; %sampling frequency in Hz
t = 0:1/fs:1-1/fs; %time base
a_t = 1.0 + 0.7 * sin(2.0*pi*3.0*t) ; %information signal
c_t = chirp(t,20,t(end),80); %chirp carrier
x = a_t .* c_t; %modulated signal

subplot(2,1,1); plot(x);hold on; %plot the modulated signal

z = hilbert(x); %form the analytical signal
inst_amplitude = abs(z); %envelope extraction
inst_phase = unwrap(angle(z));%inst phase
inst_freq = diff(inst_phase)/(2*pi)*fs;%inst frequency

%Regenerate the carrier from the instantaneous phase
regenerated_carrier = cos(inst_phase);

plot(inst_amplitude,'r'); %overlay the extracted envelope
title('Modulated signal and extracted envelope'); xlabel('n'); ylabel('x(t) and |z(t)|');
subplot(2,1,2); plot(cos(inst_phase));
title('Extracted carrier'); xlabel('n'); ylabel('cos[\omega(t)]');
```

Experiment 4
Estimation of Power Spectral Density (PSD) of Random Processes in MATLAB

1 Aim

To estimate the power spectral density of various random noise signals.

2 Theory

Power spectral density function (PSD) is a measure of the density of signal power in frequency domain. It is used to calculate the power of a signal in a given frequency interval by integrating the PSD over the frequency interval. Mathematically, if $X(t)$ is a random signal, i.e., at each time $t = t_0$, $X(t_0)$ is a random variable, then, we compute the power spectral density of $X(t)$ as below:

$$S_X(\omega) = \lim_{T\to\infty} \frac{\mathbb{E}\left[\left|\tilde{X}_T(\omega)\right|^2\right]}{T}, \tag{1}$$

where the expectation is taken over the randomness of the $X(t)$ process, and we have defined

$$\tilde{X}_T(\omega) = \mathcal{F}\left\{X_T(t)\right\} = \mathcal{F}\left\{X(t)\text{rect}\left(\frac{t}{T}\right)\right\}, \tag{2}$$

i.e., $\tilde{X}_T(\omega)$ is the Fourier transform of the $X(t)$ process truncated to the time interval $[-T, T]$. The PSD of a random signal is directly related to the auto-correlation function $r_X(\tau) = \mathbb{E}\left[x(t)x(t-\tau)\right]$, which is described by the beautiful *Wiener-Khinchin Theorem*:

$$r_X(\tau) \overset{\mathcal{F}}{\Leftrightarrow} S_X(\omega). \tag{3}$$

A practical tool to calculate Fourier transform of a signal is to use the Fast Fourier Transform (FFT). In this experiment, we will explore the `fft` procedure in MATLAB to compute the PSD from first principle. Also, we will compute the auto-correlation function and take its Fourier Transform using `fft` to verify the Wiener-Khinchin Theorem.

3 Experiments

3.1 Compute the autocorrelation function of a Gaussian random process

Use the following `MATLAB` code to find the auto-correlation of a Gaussian random process.

```
L = 1024; % Length of the random signal

X = randn(L,1); % A random Gaussian vector of length L

figure();

Rxx=1/L*conv(flipud(X),X); % Compute auto-correlation, normalized by length

lags=(-L+1):(L-1);

plot(lags,Rxx);

title('Auto-correlation Function of white noise');

xlabel('Lags')

ylabel('Correlation')

grid on;
```

Tasks:

1. What do you expect as the output of the code and why?
2. What happens if the `rand` function is used instead of `randn`?
3. How about `randi`?
4. Is there a special reason behind taking such particular value of L? What type of values of L are suitable?

3.2 Compute the PSD of white noise by computing FFT of the auto-correlation

Compute the PSD by applying FFT to the auto-correlation using the Wiener-Khinchin theorem. The following code can be appended at the end of the code before to obtain this PSD.

```
Sxx = fft(Rxx);

Sxx = fftshift(Sxx); % Shifts the zero-frequency components to center the spectrum

normFreq = [-L+1:L-1]/L; % normalized frequencies

figure;

plot(normFreq, Sxx, '-r', 'Linewidth', 2);

axis([-0.5 0.5 0 10]); grid on;

ylabel('PSD(dB/HZ)(Wiener)');

xlabel('Normalized Frequency');
```

3.3 Compute PSD from first principle

Compute the PSD from first principle using Eq. (1). However, note that this definition assumes the signal to be of infinite duration, which is impossible for practical signals. So, we must use large signal length to get good approximation. See what the following code yields. (Append the following at the end of the first code)

```
Z = 1/sqrt(L) * fft(X); % Fourier transform of X

Pavg = (abs(Z))^2;
```

Tasks:

1. Append the above code at the end of the first code and plot this estimate of PSD as before (follow the plot code at the end of the second code)
2. Compare the PSD estimates obtained from the codes 2 and 3.
3. Repeat the experiments above with different suitable values of L.
4. Describe your observations.

3.4 Do a Monte-carlo Simualtion to obtain smooth curves

Observe that whatever we have doe till now considers a single random vector $\boldsymbol{X}$ and so the calculated quantities are all random. We have to run the experiment many times and average the final obtained result. This is called the *Monte-Carlo* procedure of estimation. Use the following code to obtain smooth curves of the auto-correlation, and the PSD's obtained from both the Wiener-Khinchin theorem as well as the first principle approach.

```
clear all;

clc;

L = 1024; % Length of the random signal

sigma = 2;

iter = 1000;

Rxx_avg = zeros(2 * L - 1,iter);

Sxx_avg = zeros(2 * L - 1,iter);

P_avg = zeros(L,iter);

for i = 1 :  iter

X = sigma * randn(L,1); % A zero mean random Gaussian vector of length L with variance
    sigma^2

Rxx=1/L*conv(flipud(X),X); % Compute auto-correlation, normalized by length

Rxx_avg(:,i) = Rxx;

Sxx = fft(Rxx);
```

```
Sxx = fftshift(Sxx); % Shifts the zero-frequency components to center the spectrum
Sxx_avg(:,i) = Sxx;
Z = 1/sqrt(L) * fft(X); % Fourier transform of X
PSD = (abs(Z)).^2; % PSD estimate
P_avg(:,i) = PSD;
i
end
Rxxavg = mean(Rxx_avg, 2);
Sxxavg = mean(Sxx_avg, 2);
Pavg = mean(P_avg, 2);

figure();

lags=(-L+1):(L-1);
plot(lags,Rxxavg,'Linewidth',2);
title('Auto-correlation Function of white noise');
xlabel('Lags')
ylabel('Correlation')
grid on;

normFreq = [-L + 1:L - 1]/L; % normalized frequencies
figure;
plot(normFreq, 10*log10(abs(Sxxavg)), '-r', 'Linewidth', 2);
axis([-0.5 0.5 0 10]); grid on;
ylabel('PSD(dB/HZ)(Wiener)');
xlabel('Normalized Frequency');
title('PSD using Wiener-Khinchin');


Freqvec = [-L/2 :  L/2 - 1] / L;
```

```
figure;
plot(Freqvec, 10*log10(Pavg), '-r', 'Linewidth', 2);
axis([-0.5 0.5 0 10]); grid on;
ylabel('PSD(dB/HZ)');
xlabel('Normalized Frequency');
title('PSD from first principle');
```

Tasks:

1. Vary the number of iterations and see the effect on the smoothness of the outputs obtained.
2. Why do we not get the constant value of the PSD?

EXPERIMENT-5A

BPSK Modulation, Demodulation and PSD and BER analysis

AIM :-

a) To perform Binary Phase Shift keying (BPSK) – Modulation and Demodulation.

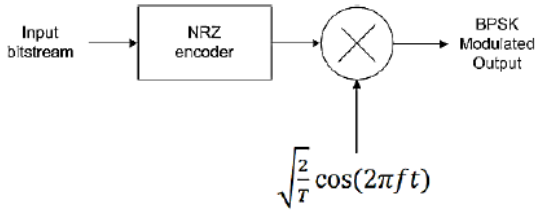
b) To obtain the PSD of the BPSK modulated signal

c) To calculate the Bit Error Rate (BER) of BPSK

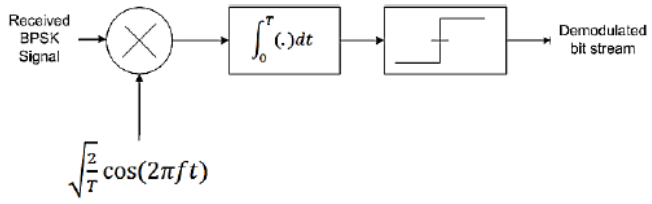
THEORY:-

Binary Phase Shift Keying (BPSK) is a two phase modulation scheme, where the 0's and 1's in a binary message are represented by two different phase states in the carrier signal:
θ = 0 for binary 1 ;
θ = \pi for binary 0.
The basis function in BPSK, is taken as a sinusoidal signal. Modulation is achieved by varying the phase of the sinusoid depending on the message bits. Therefore, within a bit duration Tb, different phase states of the carrier signal are represented as
S1 (t) = Ac cos(2πfct) ; 0 ≤ t ≤Tb for binary 1
S0(t) = Ac cos(2πfct+π) 0≤ t ≤Tb for binary 0
where, Ac is the amplitude of the sinusoidal signal, fc is the carrier frequency(Hz), t being the instantaneous time in seconds, Tb is the bit period in seconds. The signal S0(t) stands for the carrier signal when information bit ak=0 was transmitted and the signal S_1(t) denotes the carrier signal when information bit ak=1 was transmitted.

BPSK Transmitter :



BPSK Receiver:



Bit Error Rate (BER): Consider that the transmitted signal is affected with additive white Gaussian noise (AWGN), so that the received signal is expressed as:

$$R(t) = \sqrt{\frac{2E_b}{T_b}}\cos\left(2\pi ft + \phi(t)\right) + w(t),$$

where $\phi(t)$ is the phase (0 or π) of BPSK modulated carrier and w(t) is the AWGN noise. Then, the corresponding signal space representation of the transmitted and the received signals can be expressed as:

The transmitted vector: **t** = $\sqrt{E_b}$ **or** $-\sqrt{E_b}$

The received vector: **r=t + n,**

Where **n** is the Gaussian noise vector, which can perturb the actual transmitted vector.

The optimal detection rule is to choose the symbol 1 if the received vector is positive, else choose the symbol 0

The BER is the probability of wrong detection, and therefore is equal to the probability that the noise vector n is such that it changes the sign of the received vector from the transmitted vector.

The probability can be evaluated to be:

$$BER = 0.5 * erfc(\sqrt{\frac{E_b}{N_0}}),$$

Where $N_0/2$ is the PSD of the white Gaussian noise.

1. **BPSK Modulation and Demodulation**

Experiment with the following code and investigate:

Tasks:

1. Observe the effects of changing the values of f_c,f_b and f_s.
2. What happens if the basis function is changed? Observe.
3. What happens if the carrier used at the receiver has phase error? Observe the effect of this *asynchronous* recovery by taking different values of the pghase offset at the receiver. What happens if the phase offset is a uniform random variable in [0,2π]?

```
clc
clear all;

%%% Data
 dat = binornd(1 , 0.5, 1 , 10);
 polardat = 2 * dat - 1;

%%% Data and carrier timings
 f_b = 1000;
 T_b = 1 / f_b;
 n_c = 3;
 f_c = n_c * f_b;
```

```
T_c = 1 / f_c;
samp = 100;
f_s = samp * f_c;
T_s = 1 / f_s;

t = [0 : T_s : length(dat) * T_b]';
%% Generate carrier
E_b = 1;
carr =  cos(2 * pi * f_c * t);

%% Data Pulse Generation
dur = n_c * samp;
pulsedat = zeros(length(t), 1);
for i = 1 : length(dat)
    pulsedat((i - 1) * dur + 1 : i * dur) = polardat(i) * ones(dur,
1);
end

 %% Modulation
modwave = pulsedat .* carr;

%% Channel
SNR = 2;
sigma = sqrt(E_b / (2 * 10^(SNR/10)));
receive = modwave + sigma *
randn(length(modwave),1);%awgn(modwave, SNR);
figure;

%% Demodulation
demod = receive .* carr;
decode = zeros(length(dat), 1);
corr = zeros(length(dat), 1);
for i = 1 : length(dat)
   corr(i) = sum(demod((i - 1) * dur + 1: i * dur));
   decode(i) = (corr(i) >= 0);
end

subplot(4,1,1); plot(t,pulsedat,'-k',t, carr , '-b','Linewidth',2);
title('Binary input sequence');
grid on;
```

```
subplot(4,1,2); plot(t, modwave, '-k', 'Linewidth', 2); title('BPSK
modulated carrier');
grid on;
subplot(4,1,3); plot(t, receive, '-k', 'Linewidth', 2);
title('Received signal (corrupted by AWGN noise)');
grid on;
subplot(4,1,4); plot(t, demod, '-k', 'Linewidth', 2); title('Result of
Product Modulator');
grid on;
```

2. **BPSK Modulated Signal PSD:**

Find the PSD of the BPSK modulated signal using Monte-Carlo simulation.

Tasks:

1. Observe the effect of f_c and f_b in the spectrum.
2. Is there any effect of the length of the input sequence?
3. What happens if the input random Bernoulli sequence has a bias?

```
clc
clear all;

%%% Data and carrier timings
f_b = 100;
T_b = 1 / f_b;
n_c = 20;
f_c = n_c * f_b;
T_c = 1 / f_c;
samp = 100;
f_s = samp * f_c;
T_s = 1 / f_s;

bitnum = 10^2;
t = [0 : T_s : bitnum * T_b]';

iter = 300;
spectrum = 0;


for i = 1 : iter
    dat = binornd(1 , 0.5, 1 , bitnum);
    polardat = 2 * dat - 1;
```

```
    %% Generate carriers
    E_b = T_b/2;
    carr =  sqrt(2 * E_b/T_b) * cos(2 * pi * f_c * t);

    %% Data Pulse Generation
    dur = n_c * samp;
    pulsedat = zeros(length(t), 1);
    for j = 1 : length(dat)
        pulsedat((j - 1) * dur + 1 : j * dur) = polardat(j) * ones(dur, 1);
    end
    %% Modulation
    modwave = pulsedat .* carr;

    %% PSD
    modfft = fftshift(fft(modwave));
    L = length(modfft);
    sx = abs(modfft)/(2*E_b*L);
%     sx_one_side = sx(1 : floor(n_bin / 2) - 1);
    spectrum = spectrum + sx;
    i
end
spectrum = spectrum / iter;
figure;
plot((-L/2 : L/2 - 1)/L * T_b / T_s, spectrum);
axis([-40,40, 0, inf]);
grid on;
```

4. BER calculation

Use the following code for BER calculation and investigate:

1. The effect of the length of the input sequence.
2. The effect of bias in the input random sequence.

```
clear all;
clc;

%number of bits
bitnum=10^6;
%generating random bits
```

```
data=binornd(1, 0.5, 1, bitnum);
%generating BPSK signal
bpsk_data=2*data-1;
%generating noise with zero mean and var. equal to 1.
noise=1/sqrt(2)*(randn(1,bitnum)+1i*randn(1,bitnum));
mean(abs(noise.^2)) %test the power of the noise
SNR=0:9;   %set SNR in dB
snr_lin=10.^(SNR/10); %calculate linear snr from dB SNR.
y=zeros(length(SNR),bitnum);
 %multiply sqrt of snr to signal and add noise:
for i=1:length(SNR)
   y(i,:)=real(sqrt(snr_lin(i))*bpsk_data+noise);
end
%reciever and ber count
err=zeros(length(SNR),bitnum); Err=zeros(10,2);
for i=1:length(SNR)
   for j=1:bitnum
     y(i,j) = y(i,j) > 0;
   end
    err(i,:)=abs(y(i,:)-data);
    Err(i,:)=size(find(err(i,:)));
end
 %calculating BER
ber=zeros(length(SNR),1);
for i=1:length(SNR)
   ber(i)=Err(i,2)/bitnum;
end
%theoretical BER calculation
theoryBer = 0.5*erfc(sqrt(snr_lin));
semilogy(SNR,ber,'b*-','linewidth',1); grid on;  hold on;
semilogy(SNR,theoryBer,'r+-','linewidth',1); grid on;
xlabel('Eb/N0'); ylabel('BER'); legend('Simulation','Theory') ;
toc;
```

5. **Signal Space representation:**

   See the effect of noise in the signal space.

   Investigate:

   Tasks:

   1. The effect of SNR.

   2. The effect of signal length.

```
%%% BPSK signal space
clc;
clear all;

bitnum = 10^2;
snr = -20;
snr_lin = 10^(snr / 10);
%%%% Tx symbol
tx = sqrt(snr_lin) * (2 * binornd(1, 0.5, 1, bitnum) - 1);
%%%% Noise
noise = 1/sqrt(2) * randn(1, bitnum);
%%%% Received symbol
rx = tx + noise;

figure;
plot(1 : bitnum, tx, 'ob', 1 : bitnum, rx, '*r');

figure;
plot(complex(tx, zeros(1, bitnum)), 'ob', 'Markersize', 12); hold on;
plot(complex(rx, zeros(1, bitnum)), 'r*'); hold on;
```