GOVT. MODEL ENGINEERING COLLEGE, THRIKKAKARA

"Evolve into academy of excellence to serve the knowledge society"

(Managed by IHRD, A Govt. of Kerala Undertaking)

DEPARTMENT OF ELECTRONICS ENGINEERING

ECL 332 COMMUNICATION ENGINEERING LAB MANUAL

2019 SCHEME

GOVT. MODEL ENGINEERING COLLEGE, THRIKKAKARA

"Evolve into academy of excellence to serve the knowledge society"

(Managed by IHRD, A Govt. of Kerala Undertaking)

DEPARTMENT OF ELECTRONICS ENGINEERING

ECL 332 COMMUNICATION ENGINEERING LAB

NAME……

BRANCH……..

SEMESTER……ROLL NO……

Certified that this is the Bonafide work done by

……

Staff-in-Charge ***Head of the Department***

Register No…… Thrikkakara

Year & Month…….. Date……..

CONTENTS

Date: 13-02-2024 Experiment No : 1

PART A Hardware Experiments

BPSK GENERATION AND DETECTION

AIM:

To design and set up a Binary Phase Shift Keying (BPSK) generator.

COMPONENTS AND EQUIPMENTS REQUIRED

Analog switch CD4016, IC 741, IC 7404, signal generator, resistor, power supply, breadboard, CRO etc.

THEORY

In the BPSK modulation system the phase of the carrier wave is inverted according to the logic level of the input data. When the data is at logic one level, the sinusoid has one fixed phase and when the data is at the other level, the phase of the sinusoid changes. BPSK and BFSK signals have a constant envelope and hence they are less susceptible to noise.

Two switches inside the quad analog switch CD4016 are used in the circuit. Op-amp is used to invert the phase of the input sine wave.

PROCEDURE

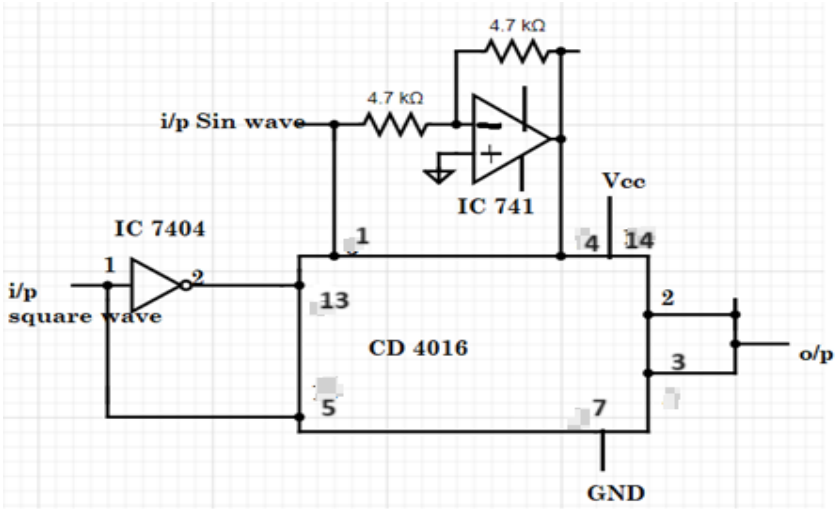
1. Set up the circuit on bread board and switch on power supply and signal generators.
2. Feed the sine wave and clock from the signal generator.
3. Keep the clock frequency lower than the sine wave frequency and observe the output.

DESIGN

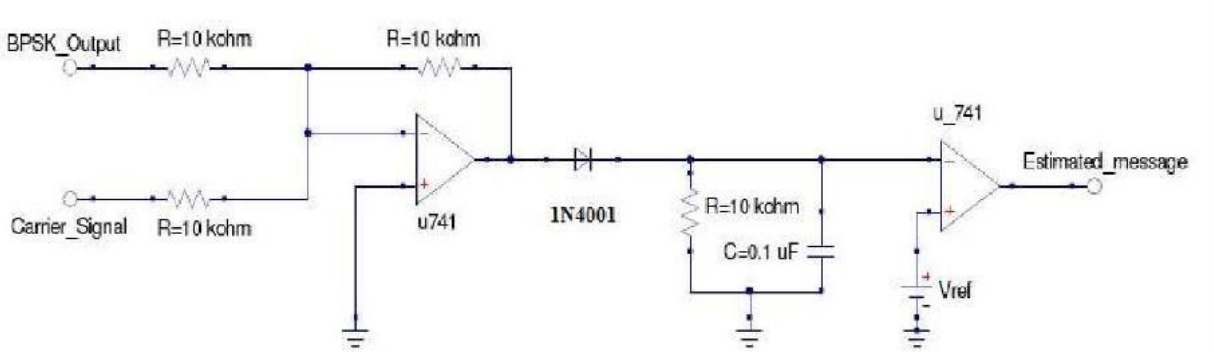
Gain of inverting amplifier, A=-Rf/R1

Let the gain be -1, so that the ratio Rf/R1=1.Take R1=Rf=4.7K

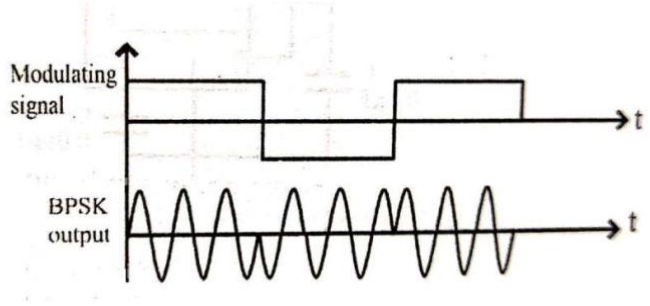
Modulator Circuit



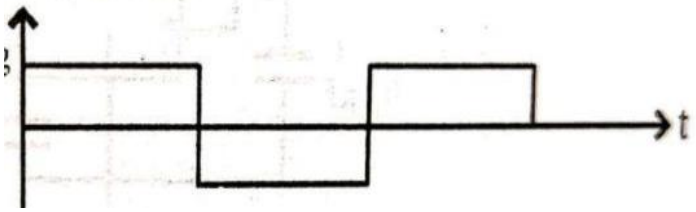
Demodulator Circuit



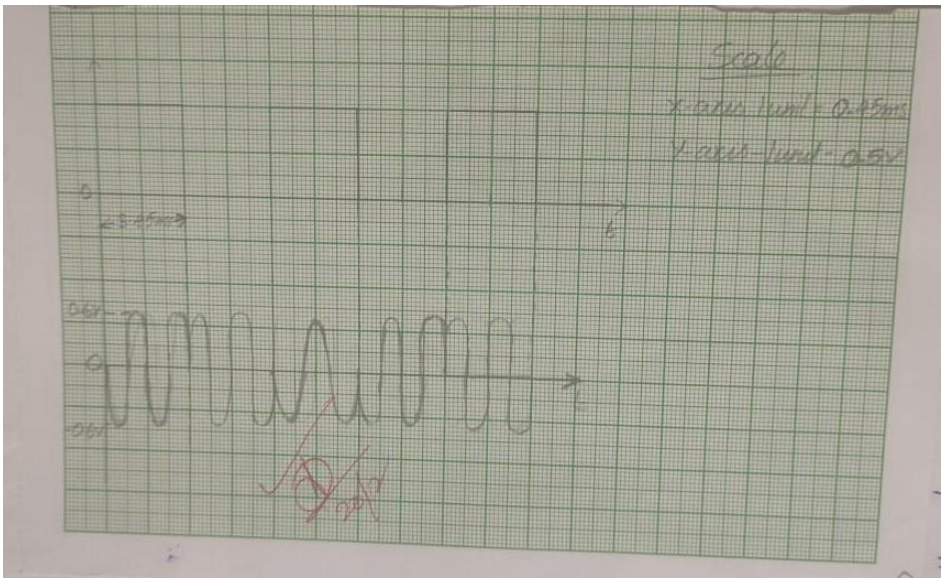
WAVEFORMS



Demodulated signal



OUTPUT WAVEFORMS:



RESULT:

BPSK modulator and detector were implemented.

Date :31-01-2024 Experiment No :2

STUDY OF CMOS PLL

AIM:
To familiarize with CMOS PLL IC CD4046A and study its functional characteristics.

COMPONENTS REQUIRED:
PLL IC, Capacitors, Oscilloscope, Function Generator, resistors, Bread board, DC Power supply, Connector wires.

THEORY:
If a voice or music (i.e., modulating signal) is applied to the VCO instead of digital data, the oscillator's frequency will move or modulate with the voice or music, this is frequency modulation "FM". It's simply moving the frequency in relation to some input voltage which also represents a voltage to frequency conversion.

PLL is a circuit designed to synchronize with an incoming signal and remain in synchronization despite the incoming signal variations. PLL mainly consists of the phase detector, a LPF and a VCO. Phase detector provides a DC Voltage proportional to the difference between the inputs. LPF removes high frequency noise. The DC voltage controls the VCO frequency which is then fed back and compared with input frequency and automatically gets itself equal to input frequency.

IC 4046-A is a silicon gate CMOS device which has pins compatible with 4046-B. It consists of a VCO and a three-phase gate. Large signals can be connected directly to the signal inputs of the PLL while series capacitor is used to couple the signal inputs to connect the small signals.

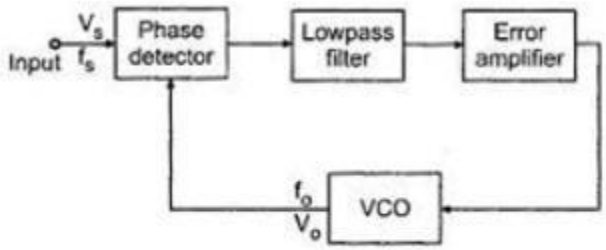


Fig. 2.123 Block diagram of PLL

DESIGN:

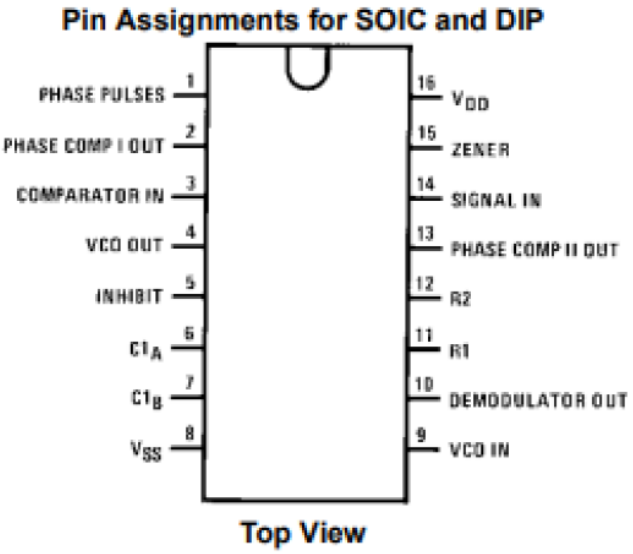
Centre frequency is given by fo = 1/2R1(C1+32PF) at VDD 5V

Let the required fo= 10 kHz. Let R1= 100kHz.

Then C1= 500 pF use 470 pF.

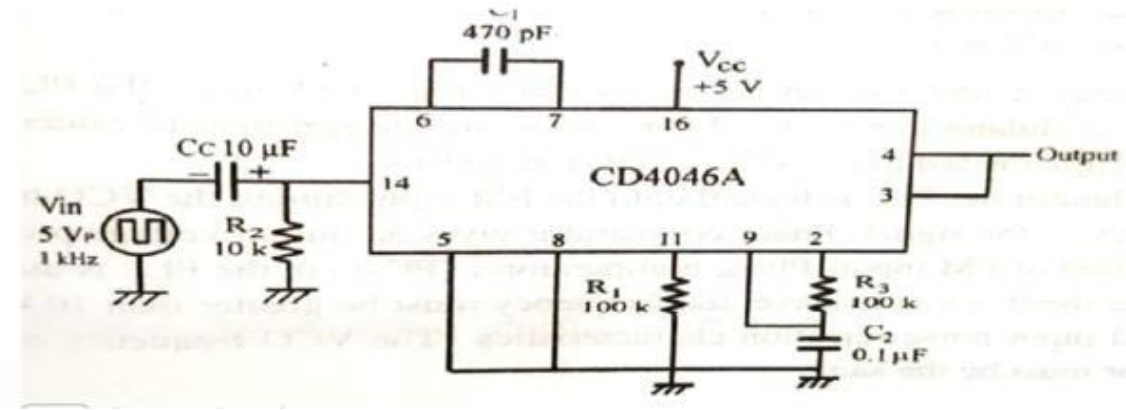
Capture range 2fc= (1/ π) *(2 πf2/R3C2)1/2

Let the captured range be 800 Hz. Take R3= 100kΩ. Then C2= 0.1 µF

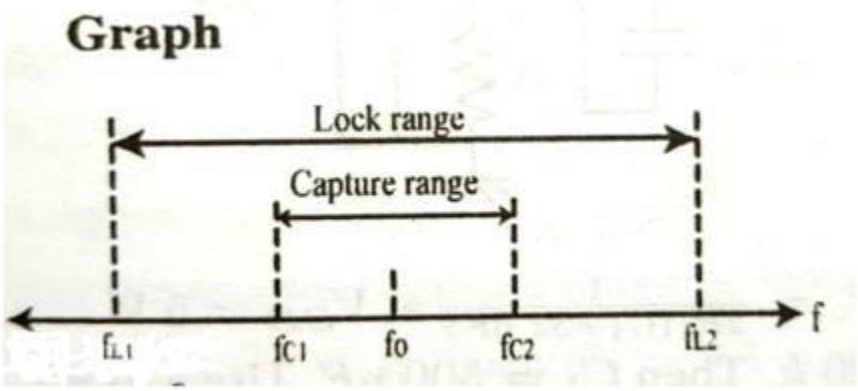


Pin#	Pin Name	Pin Description
1	Phase Pulse	Phase pulse applied to the IC
2	Phase Comp I Out	An output of Phase Comparator I
3	Comparator IN	Input at the Comparator
4	VCO OUT	Output Signal at VCO
5	INHIBIT	Allows to electronically turn on or off the output voltage power supply
6	C1A	Capacitor 1 connected to VCO
7	C1B	Capacitor 2 connected to VCO
8	VSS	Ground Pin
9	VCO IN	Input Signal at VCO
10	Demodulator OUT	Extracting the original signal
11	R1	Resistor 1 connected between VCO and Supply Voltage
12	R2	Resistor 2 connected between VCO and Supply Voltage
13	Phase Comp II OUT	Generated oscillated output at Phase II Comparator
14	Signal IN	Input Signal applied to the Phase Comparator I
15	Zener	5.2 V Zener diode for voltage regulation
16	VDD	Voltage supply pin

CIRCUIT:



WAVEFORM:



PROCEDURE:

1. Let the circuit and observe the free running frequency f0.
2. Feed a square pulse input signal and its frequency from 100 Hz to 1 MHz and note down fc1 and fL2.
3. Decrease the frequency from high value to low value and note down fc2 and fL1.
4. Calculate capture range fc = fc2 - fc1 and lock range fL= fL2 – fL1.

RESULT:

Familiarized with CMOS PLL IC 4046A and studied its functional characteristics.

Free running frequency, f0 = 10.4kHz

Lock range = 15.3kHz

Capture range = 2.4kHz

Date: 06-01-2024 Experiment No: 3

FM MODULATION AND DETECTION USING PLL

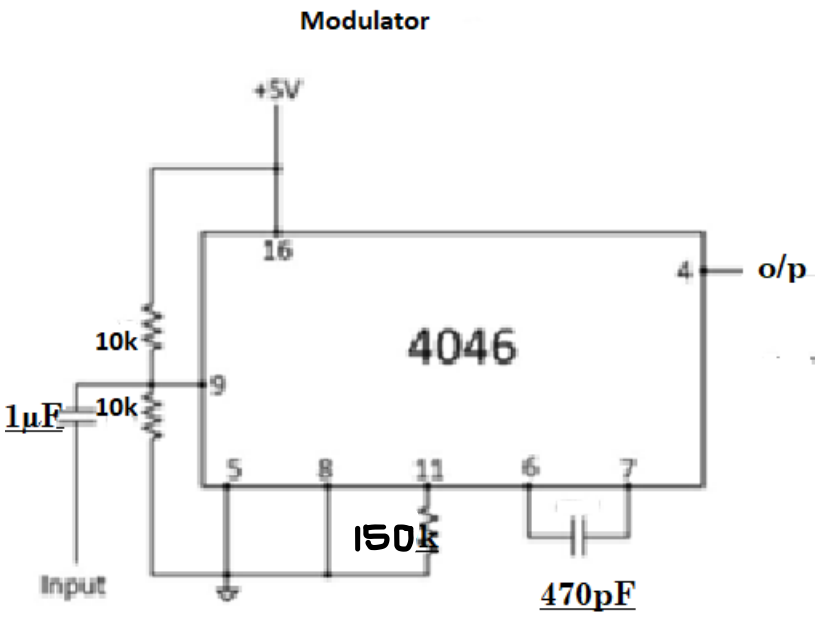
AIM:

To design and set up FM Modulation and Detection using PLL ICs.

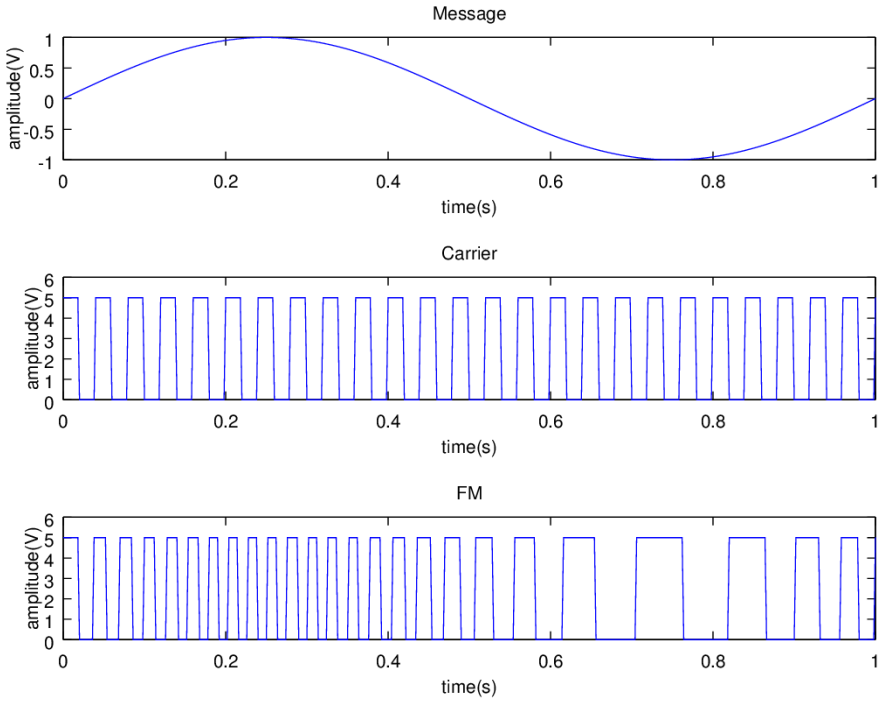
COMPONENTS AND EQUIPMENTS REQUIRED:

PLL IC, Capacitors, Oscilloscope, Function Generator, resistors, Bread board, Connector wires

CIRCUIT DIAGRAM



WAVEFORMS



THEORY:

A phase-locked loop or phase lock loop (PLL) is a control system that tries to generate an output signal whose phase is related to the phase of the input "reference" signal. It is an electronic circuit consisting of a variable frequency oscillator and a phase detector. This circuit compares the phase of the input signal with the phase of the signal derived from its output oscillator and adjusts the frequency of its oscillator to keep the phases matched. The signal from the phase detector is used to control the oscillator in a feedback loop. Frequency is the derivative of phase. Keeping the input and output phase in lock step implies keeping the input and output frequencies in lock step. Consequently, a phase-locked loop can track an input frequency, or it can generate a frequency that is a multiple of the input frequency. The former property is used for demodulation, and the latter property is used for indirect frequency synthesis. Phase-locked loops are widely used in radio, telecommunications, computers and other electronic applications. They may generate stable frequencies,

recover a signal from a noisy communication channel, or distribute clock timing pulses in digital logic designs such as microprocessors. Since a single integrated circuit can provide a complete phase-locked-loop building block, the technique is widely used in modern electronic devices, with output frequencies from a fraction of a Hertz up to many Giga Hertz. When the input frequency is less than fL1, PLL is neither in lock nor in capture, and will be in free running state generating centre frequency fo. When input frequency reaches fC1, VCO frequency becomes equal to input frequency, or VCO captures input frequency. If the input frequency increases, VCO frequency follows the input frequency upto the limit of fL2. If input frequency further increases, VCO frequency becomes centre frequency f0. If the input frequency reduced, VCO frequency becomes equal to input frequency only at fC2. If input frequency further If input frequency further decreased, VCO frequency follows input frequency only up to fL1. If input frequency is further decreased, VCO frequency retains original centre frequency fo. The frequency range fL2 – fL1 can be defined as the lock range, in which PLL keeps lock with input frequency. The frequency range fC2 – fC1 is called capture range, in which PLL able to capture the input frequency.

If a voice or music (ie, modulating signal) is applied to the VCO instead of digital data, the oscillator's frequency will move or modulate with the voice or music, this is frequency modulation "FM". It's simply moving the frequency in relation to some input voltage which it also represents a voltage to frequency conversion.

Demodulation

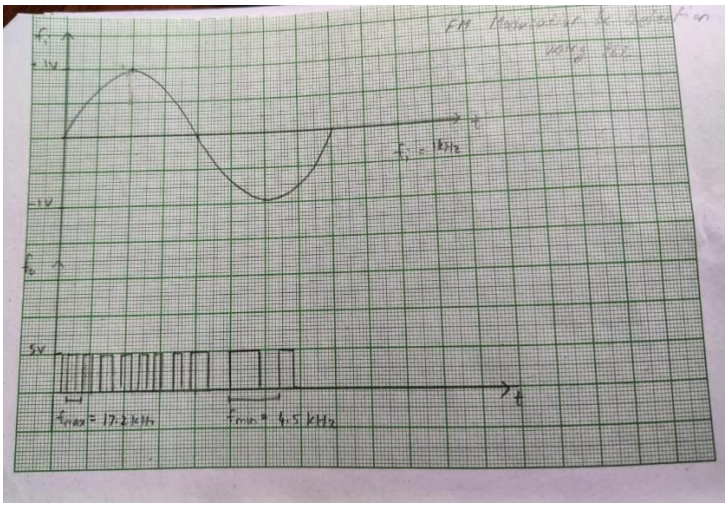
The VCO part of this IC is configured for the same free running frequency as that of the modulator IC. One of the phase detector input is fed with the modulated FM signal and the other input of the phase detector is fed with the VCO output after filtering out high frequency components. The phase variation between the two will be corresponding to the message which was used for modulation. The PD output is passed through an emitter follower internally to the demodulated output pin. The

output from this pin may contain high frequency ripples which may be eliminated by proper filtering to obtain the actual message.

PROCEDURE:

1. Find the lock & capture range of each IC and then do the connections.
2. Give the input modulating signal. Note it.
3. Plot the output waveforms of each ICs

OUTPUT WAVEFORMS:



RESULT:

An FM modulator & detector circuits are set up and the waveforms are plotted.

Centre frequency	10.4kHz
Lower lock range	310Hz
Lower capture range	9.3kHz
Upper capture range	12.5kHz
Upper lock range	15.6kHz
Modulation index (beta)	6.35kHz

Date: 20-02-2024 Experiment No : 4

Part B (Simulation Experiments)

ERROR PERFORMANCE OF BPSK

AIM:

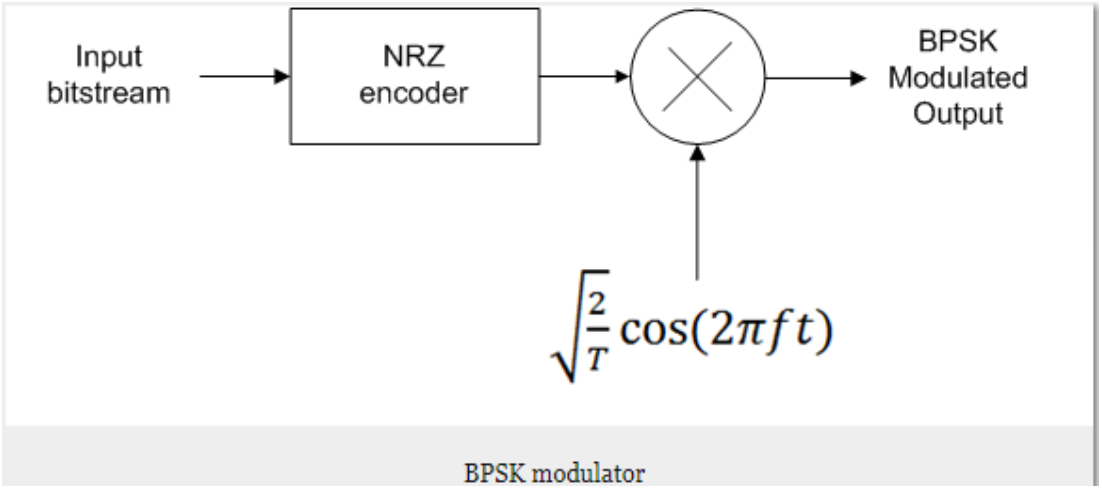
1. Generate a random binary signal of particular frequency
2. Generate a carrier signal
3. Modulate the carrier with binary signal
4. Display the output BPSK waveforms.
5. Encode using BPSK with energy per bit as E_b and represent it using points in a signal-space.
6. Simulate transmission of the BPSK modulated signal via an AWGN channel with variance $N_0/2$.
7. Detect using an ML decoder and plot the probability of error as a function of SNR per bit E_b/N_0.

THEORY:

Phase Shift **Keying is** the digital modulation technique in which the phase of the carrier signal is changed by varying the sine and cosine inputs at a particular time. PSK is widely used for wireless LANs, bio-metric, contactless operations, along with RFID and Bluetooth communications.

Binary Phase Shift Keying BPSK is also called as 2-phase PSK or Phase Reversal Keying. In this technique, the sine wave carrier takes two phase reversals such as 0° and 180°. BPSK is basically a Double Side Band Suppressed Carrier DSBSC modulation scheme, for message being the digital information. Figure 1 shows the block diagram of BPSK modulator. Figure 2. Shows the BPSK in

time domain and frequency domain. Figure 3 shows the block diagram of BPSK receiver.



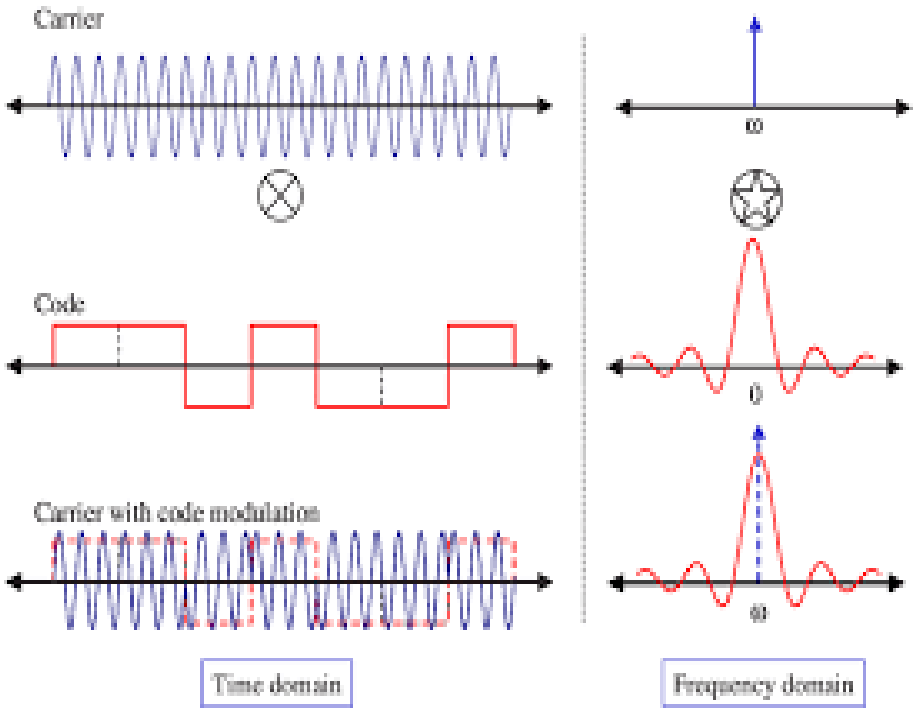


Fig 2. Waveforms

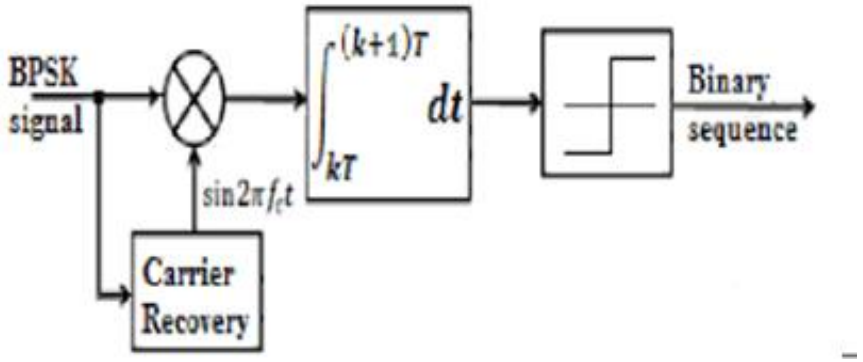


Fig 3. BPSK detector

Bit error rate (BER) of a communication system is defined as the ratio of number of error bits and total number of bits transmitted during a specific period. It is the likelihood that a single error bit will occur within received bits, independent of rate of transmission. For any given modulation, the BER is normally expressed in terms of signal to noise ratio (SNR). The bit error probability is given by

$$P_b = \frac{1}{2} erfc\left(\sqrt{\frac{E_b}{N_0}}\right)$$

Bit error probability depends on the energy contents of the signal. It does not depend on the signal shape. As the energy increases, the error reduces , so bit error probability also decreases.

The simulations are done as follows:

A randomly generated bit stream is generated and converted to BPSK waveforms. BPSK waveforms are transmitted through an AWGN channel with a fixed SNR. The received symbols are converted again to bits. The received bits are compared with transmitted bits and error is calculated. The signal to ratio is then varied and the process is repeated. Monte Carlo simulation is used to calculate Bit error probability.

Monte Carlo simulation describes a simulation in which a parameter of a system, such as the bit error rate (BER), is estimated using Monte Carlo techniques. Monte

Carlo estimation is the process of estimating the value of a parameter by performing an underlying stochastic, or random, experiment.

ALGORITHM:

1. Define message signal, carrier frequency and amplitude
2. Define sampling frequency
3. Define the time period for which the waveform has to be plotted
4. Generate a random binary data stream (0's and 1's)
5. If binary bit is 0 then multiply the carrier signal by -1 otherwise use same carrier
6. Plot the waveform

Monte Carlo simulation

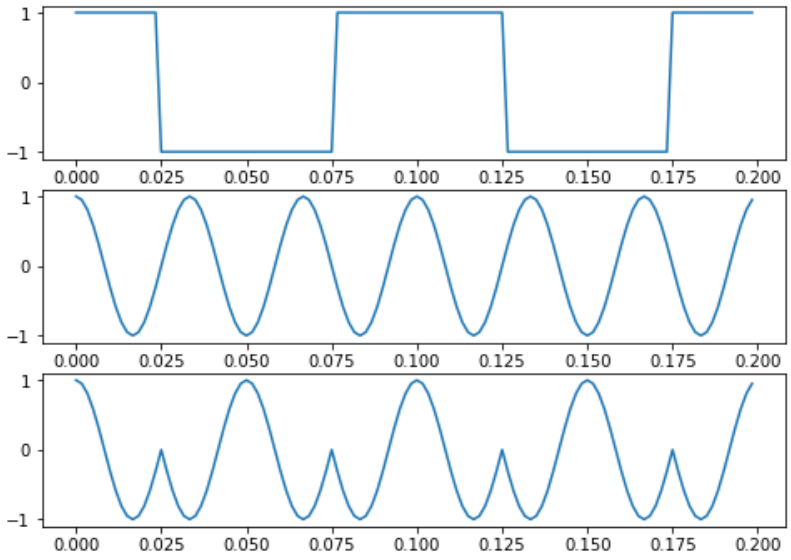
1. Mention the number of samples taken
2. Specify the range Eb/N0
3. Simulate BPSK modulator, AWGN channel and detector
4. Find errors
5. Find BER for each value of Eb/N0 and store it in an array
6. Plot the graph BER vs Eb/N0

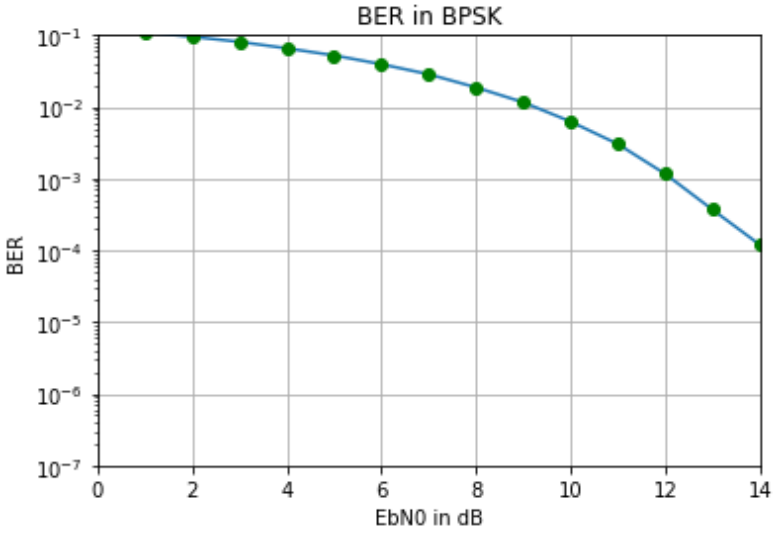
PROGRAM

```
# # Generation of BPSK signal
import numpy as np
import matplotlib.pyplot as plt
message_frequency = 10
carrier_frequency = 20
sampling_frequency = 30 * carrier_frequency
t = np.arange(0, 4/carrier_frequency, 1/sampling_frequency)
message = np.sign(np.cos(2 * np.pi * message_frequency * t) +
np.random.normal(scale = 0.01, size = len(t)))
carrier = np.cos(2 * np.pi * sampling_frequency/carrier_frequency * t)
modulated_signal = carrier * message
plt.figure(figsize=(8, 6))
```

```
plt.subplot(3, 1, 1)
plt.plot(t, message)
plt.subplot(3, 1, 2)
plt.plot(t, carrier)
plt.subplot(3, 1, 3)
plt.plot(t, modulated_signal)
plt.show()
plt.plot(t, message)
plt.plot(t, modulated_signal, "--")
plt.plot(t, carrier, "-")
plt.show()
# # Monte Carlo Simulation
N = 500000
EbN0dB_list = np.arange(0, 50)
BER = []
for i in range(len(EbN0dB_list)):
    EbN0dB = EbN0dB_list[i]
    EbN0 = 10**(EbN0dB/10)
    x = 2 * (np.random.rand(N) >= 0.5) - 1
    noise = 1/np.sqrt(2 * EbN0)
    channel = x + np.random.randn(N) * noise
    received_x = 2 * (channel >= 0.5) - 1
    errors = (x != received_x).sum()
    BER.append(errors/N)
plt.plot(EbN0dB_list, BER, "-", EbN0dB_list, BER, "go")
plt.axis([0, 14, 1e-7, 0.1])
plt.xscale('linear')
plt.yscale('log')
plt.grid()
plt.xlabel("EbN0 in dB")
plt.ylabel("BER")
plt.title("BER in BPSK")
plt.show()
```

Output waveforms





RESULT:
Simulated the code and verified the output waveforms.

Date: 27-02-2024 Experiment No: 5

ERROR PERFORMANCE OF QPSK

AIM:

1. To generate a string of message bits.

2. 2.To encode using QPSK with energy per symbol Es and represent it using points in a signal- space.

3. 3.To simulate transmission of the QPSK modulated signal via an AWGN channel with variance N0/2 in both I-channel and Q-channel.

4. To detect using an ML decoder and plot the probability of error as a function of SNR per bit Eb/N0 where Es = 2Eb.

THEORY:

Quadrature Phase Shift Keying QPSK is a variation of BPSK. The QPSK scheme is the most- widely used digital modulation technique, as it is used in wireless communications, such as WLANs/Wi- Fi and WiMAX standards, as well as many digital cellular mobile systems and TV broadcast satellite systems. Channel bandwidth depends on signaling rate .Two or more bits are combined then signal rate is reduced, the bandwidth required will be less. If two bits per symbol are transmitted, i.e. in each symbol we have 2 bits and if Phase shift keying is used then it is Quadrature Phase shift keying. QPSK the total 360-degree phase is divided into four phases.

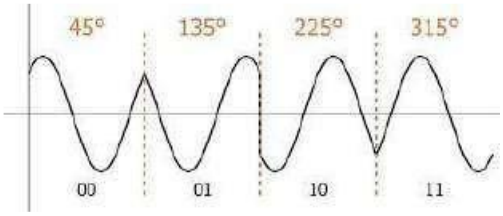
i.e., 360/4 =90 degrees.

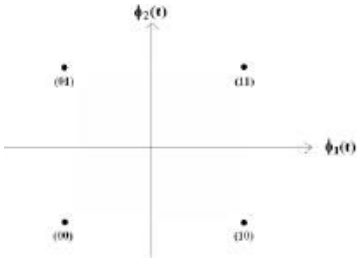
The four phases used are: 45°,135°,225°,315°.

Mathematically QPSK can be represented as:

$si(t) = \sqrt{(2\ s/T)}\cos(2\Pi fct+(2n-1)\Pi/4)$, $0 \le t \le T$ *where* $i = 1,2,3,4$

Wave forms and Signal space diagram are shown below.





Bit Error Probability:

BER has been measured by comparing the transmitted signal with the received signal and computing the error count over the total number of bits. For any given modulation, the BER is normally expressed in terms of signal to noise ratio (SNR).

BER for QPSK is = $erfc(\sqrt{E / 2N0})$

Since the symbol energy is twice the bit energy. E= 2Eb

$Pe = erfc(\sqrt{E / N0})$

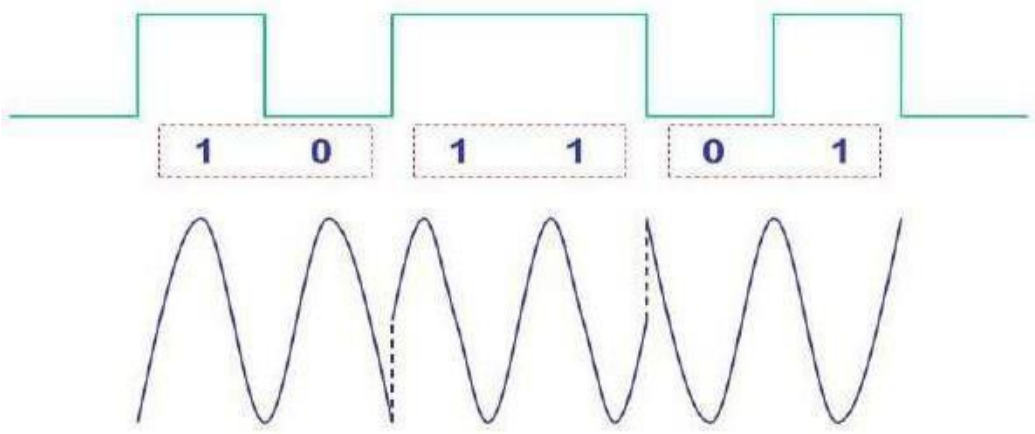
ALGORITHM:

QPSK generation

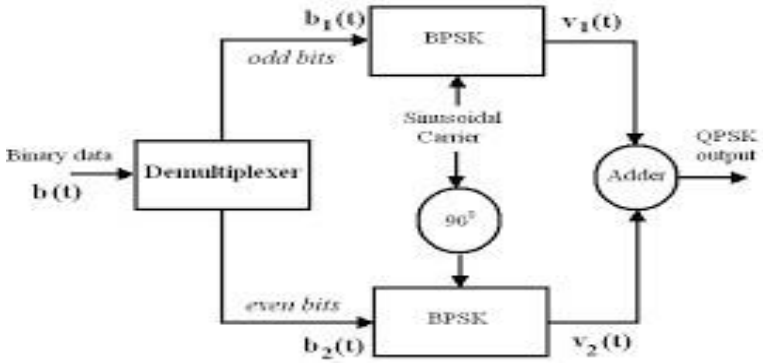
1. Generate random bit streams
2. Start FOR loop
3. Generate corresponding message signal. (bipolar form)
4. Multiply carrier 1 with odd bits of message signal and carrier 2 with even bits of message signal
5. Perform addition of odd and even modulated signals to get the QPSK modulated signal
6. Plot QPSK modulated signal.
7. End FOR loop.
8. Plot the binary data and carriers.
9. Add random noise to signal.
10. Based on the waveform decode the output.
11. Plot the input and output waveforms

Bit Error Probability

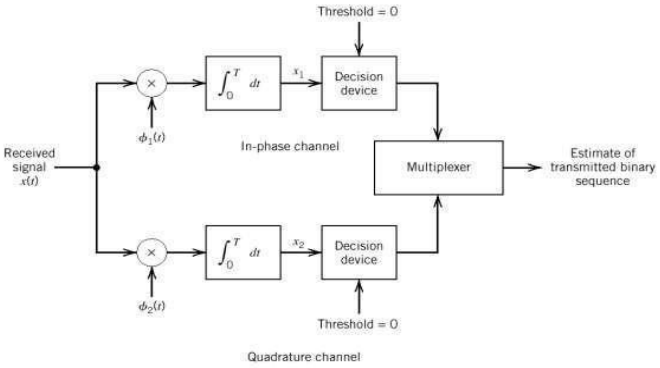
1. Mention the no. of samples taken
2. Specify the range of Eb/N0
3. Find BER for each value of Eb/No and store it as an array.
4. Plot the graph BER vs Eb/No.



QPSK Transmitter and receiver Block diagram



QPSK transmitter



QPSK receiver

PROGRAM:

```
# QPSK Modulation
import numpy as np
from numpy import pi
import matplotlib.pyplot as plt
def cosineWave(f, overSamplingRate, nCycles, phase):
 fs = overSamplingRate * f
 t = np.arange(0, nCycles*1/f, 1/fs)
 g = np.cos(2 * np.pi * f * t +
 phase)
 return list(g)
fm = 10
fc = 30
overSamplingRate = 20
fs = overSamplingRate * fc
x = np.random.rand(30) >= 0.5
str_x = [str(int(i)) for i in x]
x = "".join(str_x)
print("Message string : {}".format(x))
message = [x[2*i : 2*(i+1)] for i in range(int(len(x)/2))]
print("Message string grouped as combinations of 2 bits each
:{}".format(message))
mod_00 = cosineWave(fc, overSamplingRate, fc/fm, 3*pi/4)
mod_01 = cosineWave(fc, overSamplingRate, fc/fm, pi/4)
mod_10 = cosineWave(fc, overSamplingRate, fc/fm, -3*pi/4)
mod_11 = cosineWave(fc, overSamplingRate, fc/fm, -pi/4)
modulated_signal =[]
for i in message:
 if i == '00':
```

```
     modulated_signal = modulated_signal + mod_00
 if i == '01':
     modulated_signal = modulated_signal + mod_01
 if i == '10':
     modulated_signal = modulated_signal + mod_10
     21
 if i == '11':
     modulated_signal = modulated_signal + mod_11
t = np.arange(0, (len(x)/2) * 1/fm, 1/fs)
print(len(t), len(modulated_signal))
plt.figure(figsize = (28, 6))
plt.plot(t,
modulated_signal)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Modulated signal")
plt.grid(True)
plt.show()
# Error performance of QPSK
N = 500000
EbN0dB_list = np.arange(0, 50)
BER = []
for i in range(len(EbN0dB_list)):
    EbN0dB = EbN0dB_list[i]
    EbN0 = 10**(EbN0dB/10)
    x = np.random.rand(N) >= 0.5
    x_str = [str(int(i)) for i in x]
    x_str = "".join(x_str)
    message = [x_str[2*i : 2*(i+1)] for i in range(int(len(x)/2))]
```

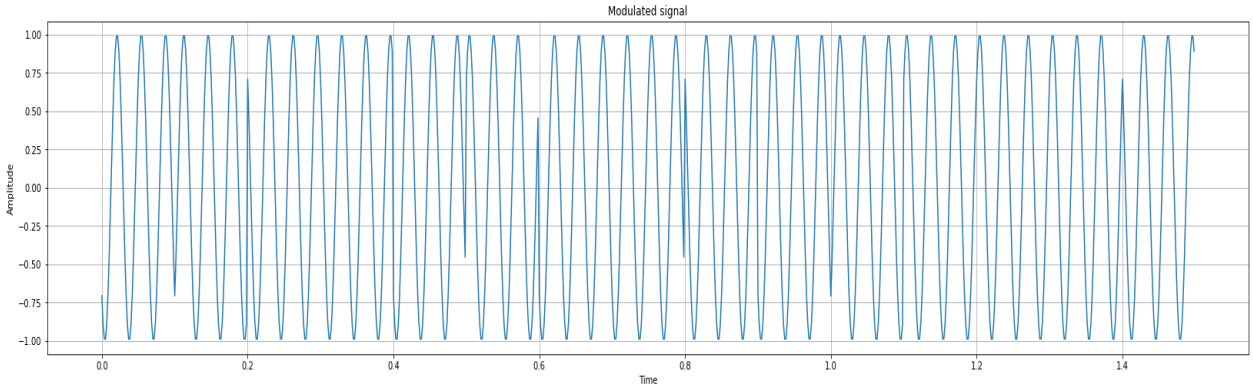
```
    noise = 1/np.sqrt(2 * EbN0)
    channel = x + np.random.randn(N) * noise
    received_x = channel >= 0.5
    xReceived_str = [str(int(i)) for i in received_x]
    xReceived_str = "".join(xReceived_str)
    messageReceived = [xReceived_str[2*i : 2*(i+1)] for i in
    range(int(len(x)/2))]
    message = np.array(message)
    messageReceived = np.array(messageReceived)
    errors = (message != messageReceived).sum()
    BER.append(errors/N)
print(BER)
plt.plot(EbN0dB_list, BER, "-", EbN0dB_list, BER, "go")
plt.xscale('linear')
plt.yscale('log')
plt.grid()
plt.xlabel("EbN0 andB")
plt.ylabel("BER")
plt.title("BER in BPSK")
plt.show()
```

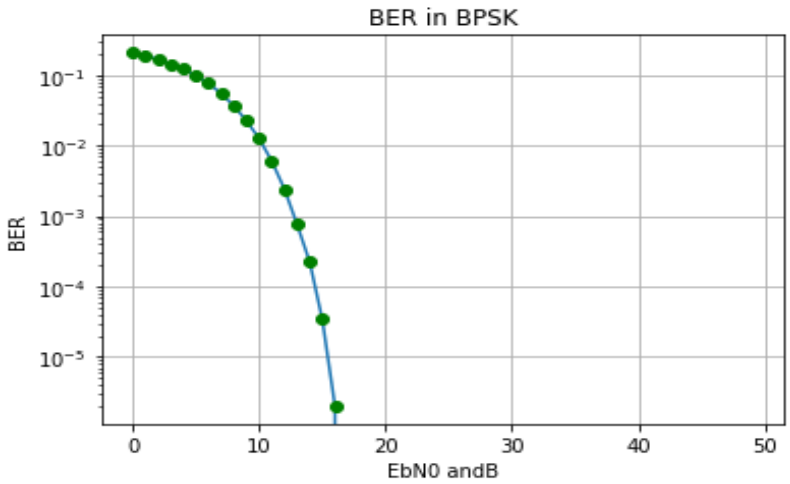
RESULT:

The simulations corresponding to QPSK modulator, detector and BER plot corresponding to QPSK are plotted herewith.

Message string : 01100010011000101000111100000 0

Message string grouped as combinations of 2 bits each :['01', '10', '00', '10', '01', '10', '00', '10', '10', '00', '11', '11', '00', '00', '00']


Modulated signal
Amplitude
Time
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4


BER in BPSK
BER
EbN0 andB
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
10
20
30
40
50

Date: 05-03-2024 Experiment No : 6

PERFORMANCE OF WAVEFORM CODING USING PCM

AIM:

To:

1. Generate a sinusoidal waveform with a DC offset so that it takes only positive amplitude value.

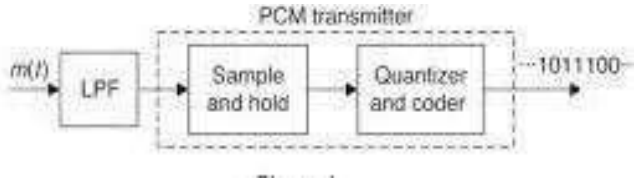
2. Sample and quantize the signal using a uniform quantizer with number of representation levels L. Vary L. Represent each value using decimal to binary encoder.

3. Compute the signal-to-noise ratio in dB.

4. Plot the SNR versus number of bits per symbol. Observe that the SNR increases linearly.

THEORY:

Pulse code modulation is a method that is used to convert an analog signal into a digital signal so that a modified analog signal can be transmitted through the digital communication network. PCM is in binary form, so there will be only two possible states high and low (0 and 1). The Pulse Code Modulation process is done in three steps Sampling, Quantization, and Coding.



Low pass filter is used as an anti-aliasing filter. A Pulse Code Modulator circuit consists of Sample and Hold circuit, Quantizer and Encoder**.** The output will be in te form of '1's' and '0's'. Sample and Hold circuit helps to collect the sample data at instantaneous values. Quantizer reduces the excessive bits and confining the data. The sampled output is fed to Quantizer. Encoder designates each quantized level by a binary code.

For PCM the quantization noise depends on no. of Quantization levels. The signal to quantization ratio of an n bit PCM system is given by

SNR=6n+1.8dB

SNR increases as number of bits n increases

ALGORITHM:

1. Generate samples of a raised sine wave of frequency f = 1 Hz or $2Hz$
$$x(t) = A \times 1 + \sin(2\pi ft)$$
with a sampling rate fs = 16 samples per second. Note that the sampling rate is four times the Nyquist rate.

2. Quantize the samples using different uniform quantizer values. The levels of quantization should be powers of 2 i.e., N= 2L. The resultant quantized signal is denoted by Xx \(nTs).

3. Encode each quantized sample using binary code with number of bits R = log2(L). This results in the pulse-code modulated binary stream of the input.

4. Given the binary string, it is possible to reconstruct back Xxx(nTs) by binary-to-decimal conversion followed by appropriate scaling. Plot original signal $x(t)$ and reconstructed signal $Xxx(nTs)$.

5. Compute the signal-to-noise ratio (SNR) as given below:

6. Signal Power Px = 1 /2 × (A/2)^2

7. Noise Power Pn = $\sum_n \frac{((X_x(nTs) - X(nTs))^2}{N}$

8. SNR in dB = 10 log (Px Pn)

9. Verify that the SNR in dB varies linearly with the number of bits R used by the encoder

PROGRAM

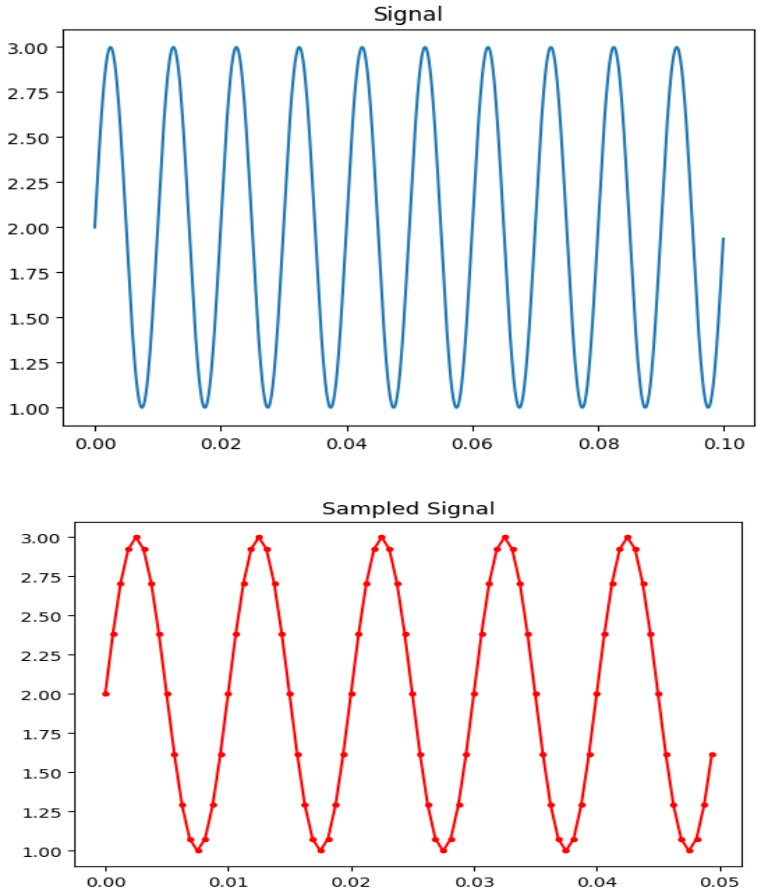
```
import numpy as np
import matplotlib.pyplot as plt
time = np.arange(0,0.1,0.0001)
mssg_f=100
dc = 2
sig = np.sin(2*np.pi*mssg_f*time) + dc
plt.plot(time,sig)
plt.title("Signal")
plt.show()
fs= 16*mssg_f
ts = np.arange(0,0.05,1/fs)
sampled_signal = dc+ np.sin(2*np.pi*mssg_f*ts)
plt.plot(ts,sampled_signal,"r.-")
plt.title("Sampled Signal")
```

```
plt.show()

# Quantizing with L levels #-TVE19EC061
L =int(input("Enter no.of quantization levels: "))
sig_min = round(min(sig))
sig_max = round(max(sig))
q_levels1 = np.linspace(sig_min,sig_max,L)
q_sig1 = []
for i in sampled_signal:
 for j in q_levels1:
  if i <= j:
   q_sig1.append(j)
   break
plt.subplot(1,2,1)
plt.plot(ts,q_sig1,"b",ts,q_sig1,"m*")
plt.title("Quantized Signal")
plt.show()
q_level=[]
for i in np.linspace(0.9,3,1000):
 for j in q_levels1:
  if i <= j:
   q_level.append(j)
   break
plt.subplot(1,2,2)
plt.plot(np.linspace(0.725,3,1000),q_level)
plt.title("Quantizer")
plt.show()

# Encoding with L quantization Levels
count = 0
q_level_map1 = {}
for i in q_levels1:
 q_level_map1[i] = q_level_map1.get(i, count)
 count += 1
 binary_code1 = {}
 bit_no = int(np.log2(L))
 for i in range(L):
  val = bin(i).replace("0b", "")
  if len(val) != bit_no:
   bin_str = "0" * (bit_no - len(val))
   bin_str += val
  else:
   bin_str = val
  binary_code1[i] = binary_code1.get(i, val)
print("Quantization Levels Mapping:", q_level_map1)
```

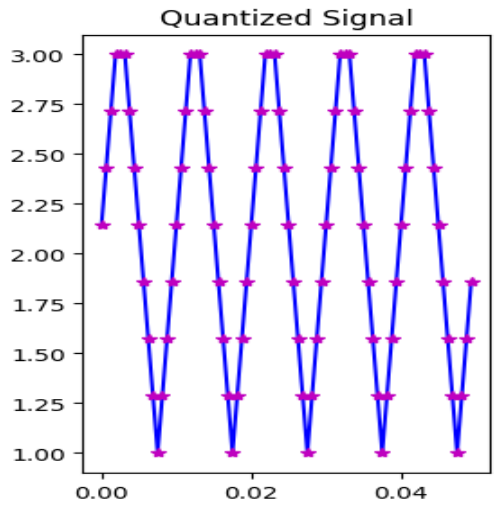
```
print("\nBinary Code:", binary_code1)
encoded_signal1 = []
for k in q_sig1:
 encoded_signal1.append(q_level_map1[k])
plt.plot(ts, encoded_signal1, "b", ts, encoded_signal1, "g*")
plt.title("Encoded Signal")
plt.show()
binary_coded_signal1 = []
for k in encoded_signal1:
 binary_coded_signal1.append(binary_code1[k])
print("Binary Coded Signal:", binary_coded_signal1)

# Quantization Noise # -
def power(s):
 p = 0
 for i in s:
  p += i**2
 P = p/len(s)
 return P
q_noise1 = q_sig1-sampled_signal
plt.subplot(2,1,1)
plt.plot(ts,q_noise1)
plt.title("Quantization Noise")
plt.show()
'''plt.subplot(2,1,2)
plt.hist(q_noise1)
plt.show()

#SNR
p_signal = power(sig)
p_noise = power(q_noise1)
snr = p_signal/p_noise
snr_db = 20*np.log10(snr)
print("Signal-to-Noise ratio in dB: ", snr_db)
snr_db=[]
s_min = round(min(sig))
s_max = round(max(sig))
power_signal = power(sig)
for i in range(1,11):
 R = i
 L = 2**R
 step_size = (s_max-s_min)/L
 power_noise = (step_size**2)/3
 snr = power_signal/power_noise
 snr_db.append(20*np.log10(snr))
```

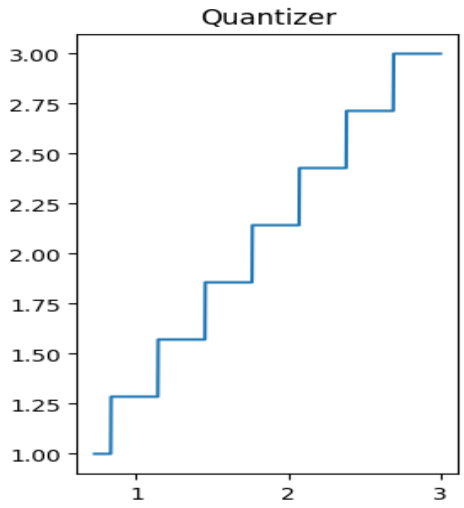
```
plt.plot(range(1,11),snr_db,"r*-")
plt.xlabel("No.of Bits per symbol")
plt.ylabel("SNR in dB")
plt.title("SNR vs No.of bits per symbol")
plt.show()
```

RESULT

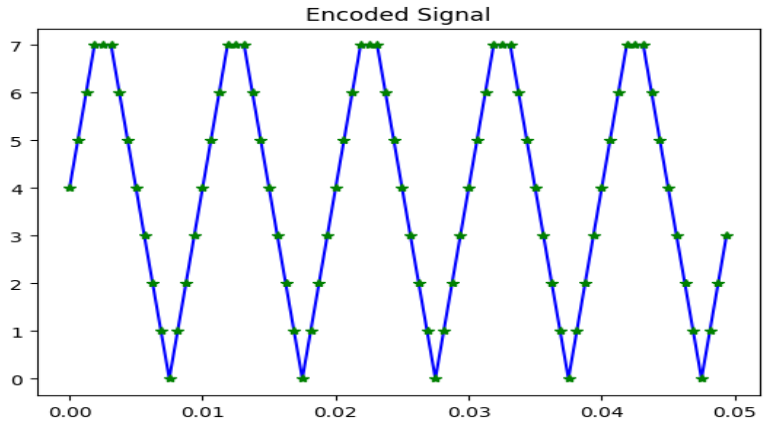



```
Enter no.of quantization levels: 8
```

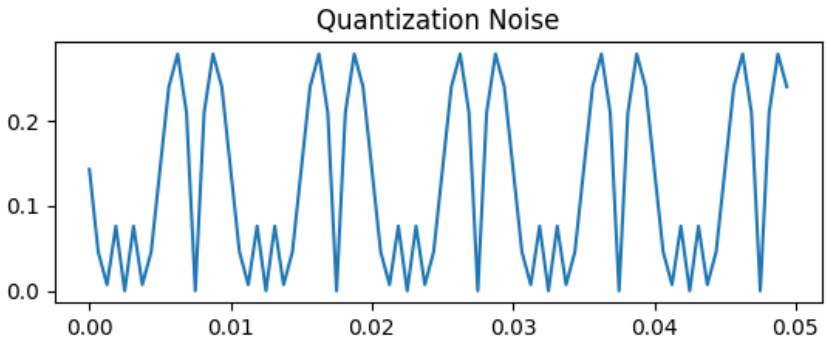




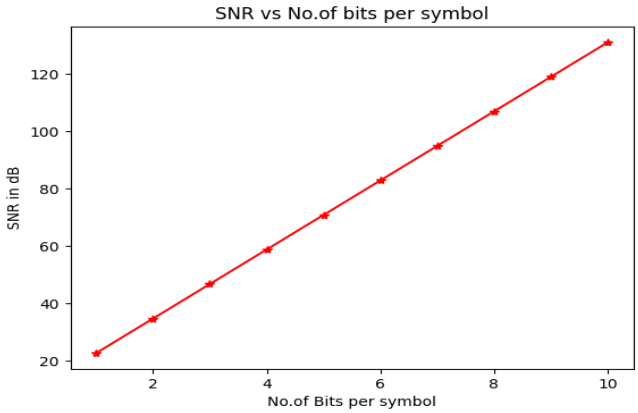

```
Quantization Levels Mapping: {1.0: 0, 1.2857142857142856: 1,
1.5714285714285714: 2, 1.8571428571428572: 3, 2.142857142857143: 4,
2.4285714285714284: 5, 2.7142857142857144: 6, 3.0: 7}

Binary Code: {0: '0', 1: '1', 2: '10', 3: '11', 4: '100', 5: '101', 6:
'110', 7: '111'}
```




```
Binary Coded Signal: ['100', '101', '110', '111', '111', '111', '110',
'101', '100', '11', '10', '1', '0', '1', '10', '11', '100', '101', '110',
'111', '111', '111', '110', '101', '100', '11', '10', '1', '0', '1', '10',
'11', '100', '101', '110', '111', '111', '111', '110', '101', '100', '11',
'10', '1', '0', '1', '10', '11', '100', '101', '110', '111', '111', '111',
'110', '101', '100', '11', '10', '1', '0', '1', '10', '11', '100', '101',
'110', '111', '111', '111', '110', '101', '100', '11', '10', '1', '0',
'1', '10', '11']
```




```
Signal-to-Noise ratio in dB:  44.79038196417986
```



The PCM simulation corresponding to an analog signal is done and quantization is done for various levels. The effect of quantization on SNR is also studied.

Date: 09-04-2024 Experiment No: 7

PULSE SHAPING AND MATCHED FILTERS

AIM:

1. To Generate a string of message bits.
2. Use root raised cosine pulse p(t) as the shaping pulse, and generate the corresponding baseband signal with a fixed bit duration Tb. You may use roll-off factor as α = 0.4.
3. Simulate transmission of baseband signal via an AWGN channel
4. Apply matched filter with frequency response $P_r(f) = P^*(f)$ to the received signal.
5. Sample the signal at mT_b and compare it against the message sequence.

THEORY:

Inter symbol interference (ISI) is a form of distortion of a signal in which the symbol interferes with subsequent symbol. The spreading of the pulse beyond its allotted time interval causes it to interfere with neighboring pulses.ISI degrade the bit and symbol error rate performance in the presence of noise. The causes of ISI are multipath propagation and dispersion of channels. The baseband transmission system is as shown in figure 1.

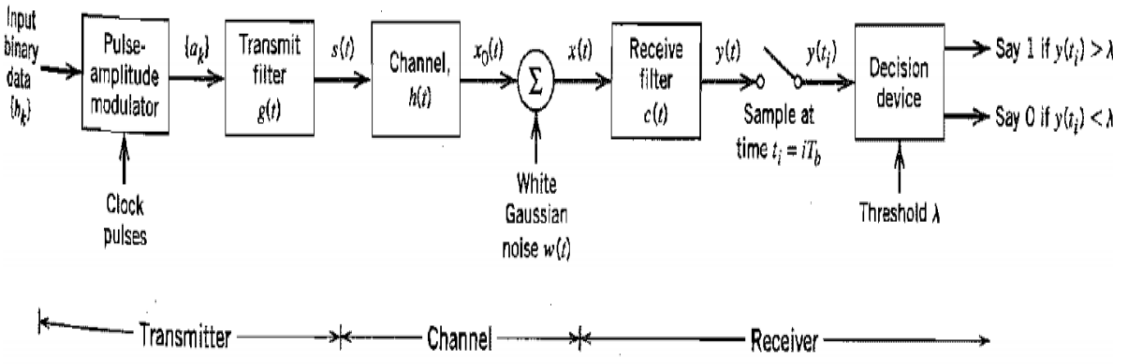


FIGURE 4.7 Baseband binary data transmission system.

Here,

$$ak = \begin{cases} +1, if\ the\ symbol\ b_k\ is\ 1 \\ -1, if\ the\ symbol\ b_k\ is\ 0 \end{cases}$$

$$s(t) = \sum_k a_k g(t - kT_b)$$

The received filter output is written as

$$y(t) = \mu \sum_k a_k p(t - kT_b) + n(t).$$

Where μ- scaling factor and pulse p(t) is to be defined.

At ith instant

$$y(t_i) = \mu \sum_{k=-\alpha} a_k p[(i-k)T_b] + n(t_i)$$

$$y(t_i) = \mu a_i + \sum_{\substack{k=-\alpha, \\ k \neq i}} a_k p[(i-k)T_b] + n(t_i)$$

The second term is due to ISI.

To avoid ISI Pulse shaping filters are used. Pulse shaping filter must be chosen carefully not to introduce inter symbol interference. The commonly used pulse shaping filters are –

(i) Rectangular pulse shape: This pulse shape has poor spectral properties with high sidelobes.

(ii) Sinc pulse shape: Theoretically, the sinc filter has ideal spectral properties, as the Fourier transform of a sinc function is an ideal lowpass spectrum. However, a sinc pulse is non-causal, hence not realizable.

(iii) Raised-cosine pulse: This is a pulse widely used in practice. The pulse shape and the excess bandwidth can be controlled by changing the roll-off factor ($0 \leq \alpha \leq 1$, where 0 means no excess bandwidth, and 1 means maximum excess bandwidth)

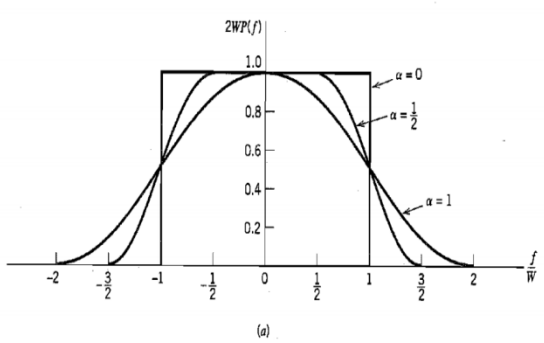


Figure 2. Raised cosine spectrum- frequency spectrum for different α values

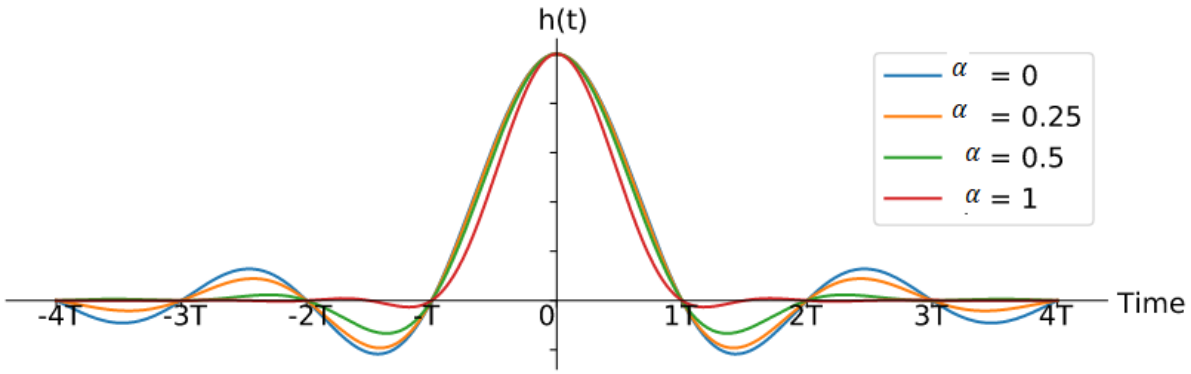


Figure 3. Impulse response of Raised cosine filter with various α values

$$h(t) = \begin{cases} \frac{\pi}{4T} Sinc\left(\frac{1}{2\alpha}\right), & t = \pm\frac{T}{2\alpha} \\ \frac{1}{T} sinc\left(\frac{t}{T}\right) \frac{Cos\left(\frac{\pi\alpha t}{T}\right)}{1 - (\frac{2\alpha t}{T})^2}, & otherwise \end{cases}$$

(iv) Root raised cosine pulse: It has a transfer function equal to square root of raised cosine filter. This filter satisfies Nyquist criteria. These filters are real valued and symmetric. It has its own matched filter.

(v) Gaussian filter: The impulse response of this filter is a Gaussian function. Gaussian pulses have good spectral properties.

Matched filter:

A matched filter is a filter to provide maximum signal to noise ratio at the output. The Characteristic of the matched filter at the receiver should be complex conjugate of the one at the transmitter in order to fulfill Nyquist criteria. If an RRC filter used at the transmitter, the same filter can be used as it is in the receiver since RRC filter is its own matched filter.

The impulse response of matched filter is $h(t)= s(t-\tau)$. Where s(t) is the input.

ALGORITHM:

1. Specify the no. of symbols transmitted
2. Specify the no. of samples of transmitted signal.
3. Generate random binary data and convert it to NRZ format.
4. Oversample each bit by adding 8 samples.
5. Assume the number of taps, roll off rate (alpha) and sample period.
6. Generate a Raised Cosine filter.
7. Perform convolution between raised cosine filter and input signal.
8. Simulate an AWGN channel.
9. Send the convoluted signals through AWGN channel.
10. Generate output after convolution of noise affected signal with matched filter response.
11. Plot the output and compare with transmitted bits.

PROGRAM:

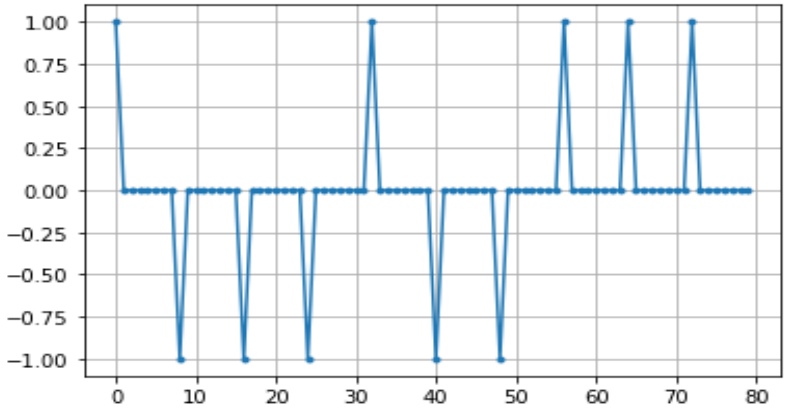
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
num_symbols = 10
sps = 8
num_taps = 101
beta = 0.35
```

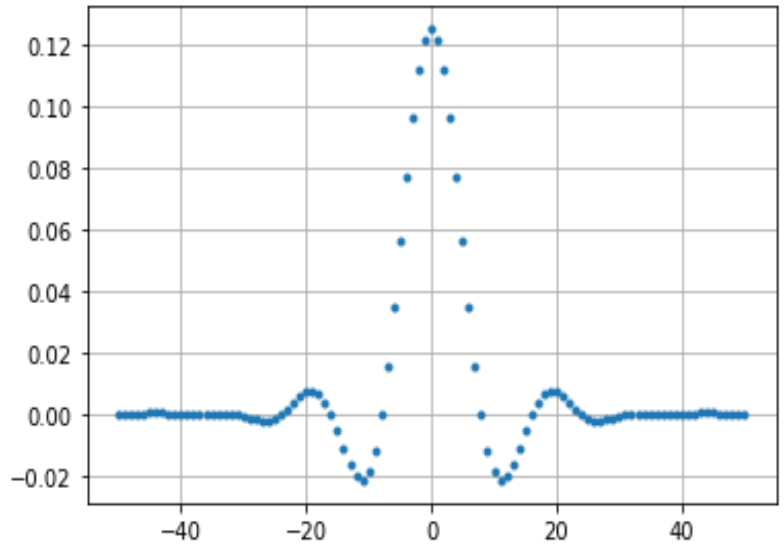
```
output_length = 80  # Desired output length


# Generate random bits

bits = np.random.randint(0, 2, num_symbols)


# Create the pulse

x = np.array([np.concatenate(([bit*2-1], np.zeros(sps-1))) for bit in
bits]).flatten()


# Plot the pulse

plt.figure(0)

plt.plot(x, '.-')

plt.grid(True)

plt.show()


# Create the raised-cosine filter

Ts = sps

t = np.arange(-50, 51)

h = 1/Ts * np.sinc(t/Ts) * np.cos(np.pi*beta*t/Ts) / (1 -
(2*beta*t/Ts)**2)


# Plot the filter

plt.figure(1)

plt.plot(t, h, '.')

plt.grid(True)

plt.show()


# Convolve the signal with the filter

x_shaped = np.convolve(x, h)
```

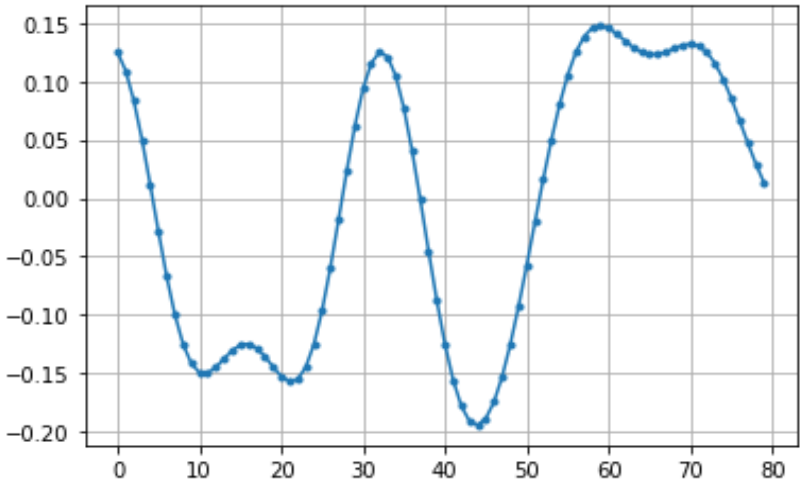
```
# Trim the convolution result to match the desired output length
start_index = (len(x_shaped) - output_length) // 2
end_index = start_index + output_length
x_shaped_trimmed = x_shaped[start_index:end_index]


# Plot the convolved signal
plt.figure(2)
plt.plot(x_shaped_trimmed, '.-')
plt.grid(True)
plt.show()


SNR_dB = 10
# Add Gaussian noise
power_signal = np.sum(np.abs(x_shaped_trimmed) ** 2) /
len(x_shaped_trimmed)
power_noise = power_signal / (10 ** (SNR_dB / 10))
noise = np.random.normal(0, np.sqrt(power_noise),
len(x_shaped_trimmed))
x_noisy = x_shaped_trimmed + noise


# Plot the convolved signal with noise
plt.figure(3)
plt.plot(x_noisy, '.-')
plt.grid(True)
plt.show()


t = np.arange(0, len(x_noisy))  # Assuming each sample corresponds to
one second


# Sample the signal
```

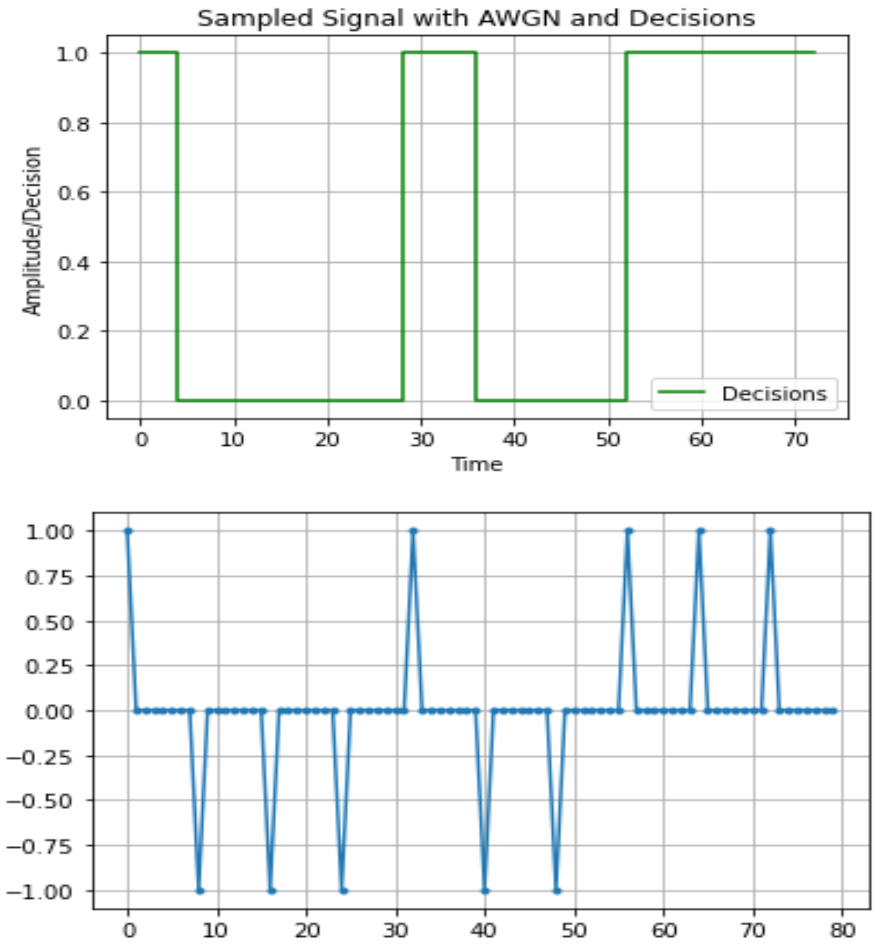
```
sampled_indices = np.arange(0, len(x_noisy), sps)  # Sample every sps-
th sample
x_sampled = x_noisy[sampled_indices]


threshold = 0


# Make decisions based on threshold
decisions = np.where(x_sampled > threshold, 1, 0)


# Plot sampled signal with decisions
plt.figure(4)
plt.step(sampled_indices, decisions, 'g', where='mid',
label='Decisions')
plt.xlabel('Time')
plt.ylabel('Amplitude/Decision')
plt.title('Sampled Signal with AWGN and Decisions')
plt.legend()
plt.grid(True)
plt.show()
plt.figure(0)
plt.plot(x, '.-')
plt.grid(True)
plt.show()
```

RESULT:


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60
70
80


0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−40
−20
0
20
40


0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
10
20
30
40
50
60
70
80



The performance of raised cosine pulse shaping and matched filter are simulated.

Date: 26-03-2024 Experiment No: 8

EYE DIAGRAM

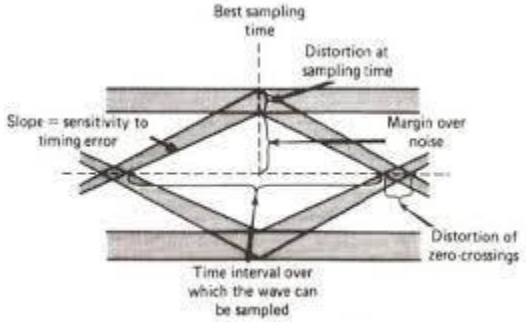
AIM:

To

1. Generate a string of message bits.
2. Use raised cosine pulse p(t) as the shaping pulse, and generate the corresponding baseband signal with a fixed bit duration Tb. You may use roll-off factor as $\alpha = 0.4$.
3. Use various roll off factors and plot the eye diagram in each case for the received signal. Make a comparison study among them.

THEORY:

An eye pattern is a pattern displayed on the screen of a cathode ray oscilloscope (C.R.O.). The shape of this pattern resembles the shape of the human eye and therefore, it is called an eye pattern. The eye pattern is a practical way to study Inter symbol interference (ISI) and its effects on a PCM or data communication system. The interior region of the eye pattern is called the eye-opening. The eye pattern provides a great deal of information about the performance of the system. The height of eye-opening at a specified sampling time defines the margin over the noise.



Interpretation of eye pattern

An eye diagram is used to evaluate high speed data quality. An eye diagram is measured in the time domain.

ALGORITHM:

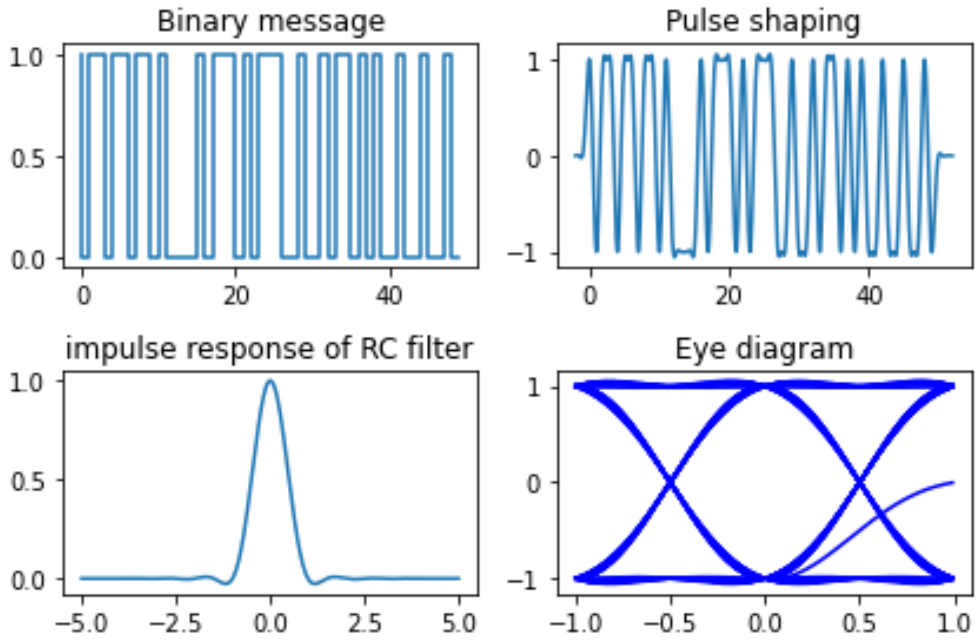
1. Specify the no. of symbols transmitted
2. Specify the no. of samples of transmitted signal.
3. Generate random binary data and convert it to NRZ format.
4. Oversample each bit by adding 8 samples.
5. Assume the number of taps, roll off rate (alpha) and sample period.
6. Create Raised Cosine filter.
7. Perform convolution between raised cosine filter and input signal.
8. Simulate an AWGN channel.
9. Send the convoluted signals through AWGN channel.
10. Generate output after convolution of noise affected signal with matched filter response.
11. Generate eye diagram
12. Obtain eye pattern for various roll off rates.

PROGRAM:

```
import matplotlib.pyplot as plt
import numpy as np
T = 1 #time period
Fs = 100 #sampling frequency
t = np.arange(-3*T, 3*T, 1/Fs)
roll=1.0
g = lambda t: np.sinc(t) * np.cos(np.pi*roll*t) / (1-(2*roll*t)**2)
#Defining a lambda function g(t) that represents a raised cosine
filter with a roll-off factor of 0.5.
binary_sequence = np.array(np.random.randint(2,size=50))
d = 2 * np.array(binary_sequence) - 1 #making the binary sequence NRZ
#to get transmitted signal
t = np.arange(-2*T, (len(d)+2)*T, 1/Fs)
y = np.sum(d[k] * g(t - k*T) for k in range(len(d)))
fig, ax = plt.subplots(2, 2)
```

```
ax[0, 1].plot(t, y)
ax[0, 1].set_title("Pulse shaping")
x = np.arange(-T, T, 1/Fs)
for i in range(2*Fs, len(y)-3*Fs, Fs):
    ax[1, 1].plot(x, y[i:i+2*Fs], 'blue')
ax[1, 1].set_title("Eye diagram")
ax[0, 0].step(np.arange(len(binary_sequence)), binary_sequence)
ax[0, 0].set_title("Binary message")
t = np.arange(-5, 5, 0.01)
ax[1, 0].plot(t, g(t))
ax[1, 0].set_title("impulse response of RC filter")
plt.tight_layout()
plt.show()
```

RESULT:



The eye diagram for various roll off rate is simulated.

Date: 27-03-2024 Experiment: 9

PART C

FAMILIARIZATION WITH SOFTWARE DEFINED RADIO (HARDWARE AND CONTROL SOFTWARE)

AIM:

To:

1. Familiarize with an SDR hardware for reception and transmission of RF signal.
2. Familiarize how it can be interfaced with computer.
3. Familiarize with GNU
4. Familiarize available blocks in GNU Radio. Study how signals can be generated and spectrum (or power spectral density) of signals can be analyzed.

THEORY:

The term "software-defined radio" (SDR) describes a technology in which the functions of a radio system's hardware components are instead defined by software. SDR systems are more flexible than traditional radio systems. Software-defined radio is a type of radio communication in which the components that have traditionally been implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented via software on a computer.

GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. This can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in research, industry, academia, government, and hobbyist environments to support both wireless communications research and real-world radio systems. It is a graphical user interface that comes with a comprehensive library of processing blocks that can be readily combined to make complex signal processing applications.

HARDWARE:

It consists of RTL-SDR dongle, antennas, USB port, connector wires and stands for Antennas.

Figure 1. SDR Hardware Part
Figure 1 shows the parts of RTL SDR Hardware set up.

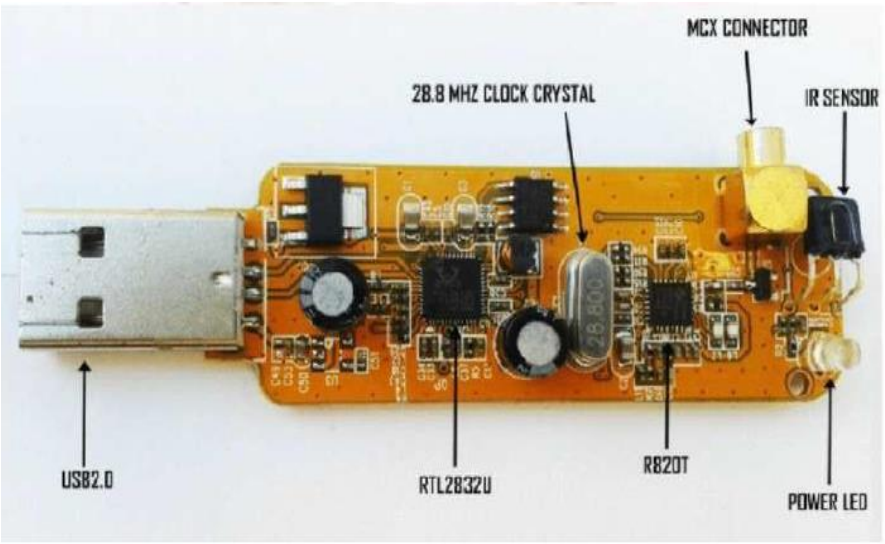


Figure 2. Inside the RTL SDR Dongle

Figure 2 shows the components inside an RTL dongle. The two important ICs are RTL 2832 which is a digital modulator and R820 T is a tuner.

INSTALLING SOFTWARE:

First step is download and install Ubuntu Desktop

Go to terminal

Install the following software Gqrx, GNU radio

Gqrx is a graphical SDR receiver.

Update Ubuntu

sudo apt-get update

To install GNU radio at

sudo apt install gnuradio

To check whether RTL SDR is installed *type rtl tab* in command window and see the list.

To install Gqrx

sudo apt-get install gqrx-sdr

Go to terminal and type *lsusb* to check whether the system has recognized the dongle. Please make sure that the system should recognize the dongle.

Can make sure this by typing the command *dmseg*

To receive a signal

Then type *"gqrx"* in terminal which will pop up a configurable menu Configure the SDR – select the I/O device

Now configure the receiver. Three panes appear in the screen. One is the receiver pane which shows receiver frequency, waterfall diagram etc. Figure 3 shows configurable receiver frequency and figure 4 shows corresponding waterfall diagram.

Figure 3. Frequency configuring window

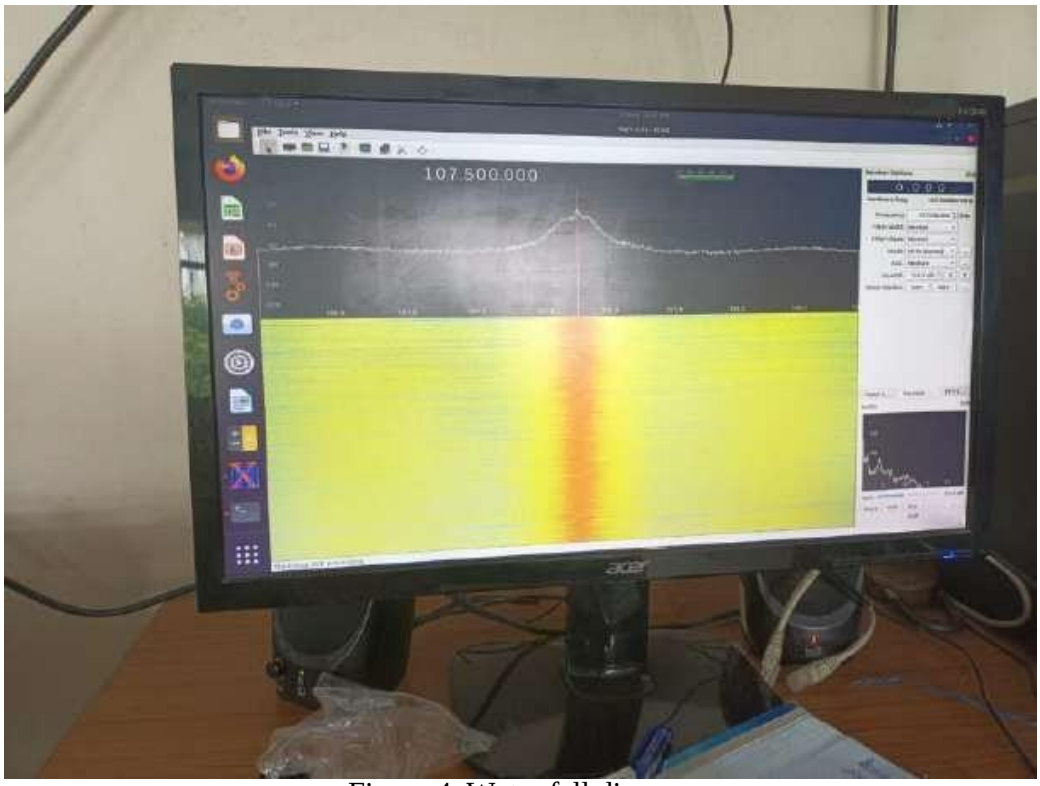


Figure 4 .Water fall diagram

You can set the frequency in receiver pane. The bottom pane shows how audio is processed.
Select frequency, mode as WFM (wideband FM)
Then press the play button.
You can hear the corresponding radio station.
This part of the experiment helps you to verify your dongle and understand software defined radio.
You can try different radio stations:

- Radio Mango 91.9 FM.
- Club 94.3 FM.
- All India Radio Air Akashvani 102.3 FM.
- Radio Mirchi 104 FM.
- Red 93.5 FM.
- Air Rainbow 102.3 FM.
- Radio Gyan Vani 105.6 FM.
- All India Radio AIR Kochi 107.5 FM.

USING GNU RADIO:

To open GNU radio in Ubuntu:
Open a terminal window using: Applications > Accessories > Terminal. At the prompt type: *gnuradio-companion*.
It opens up as shown in figure 5

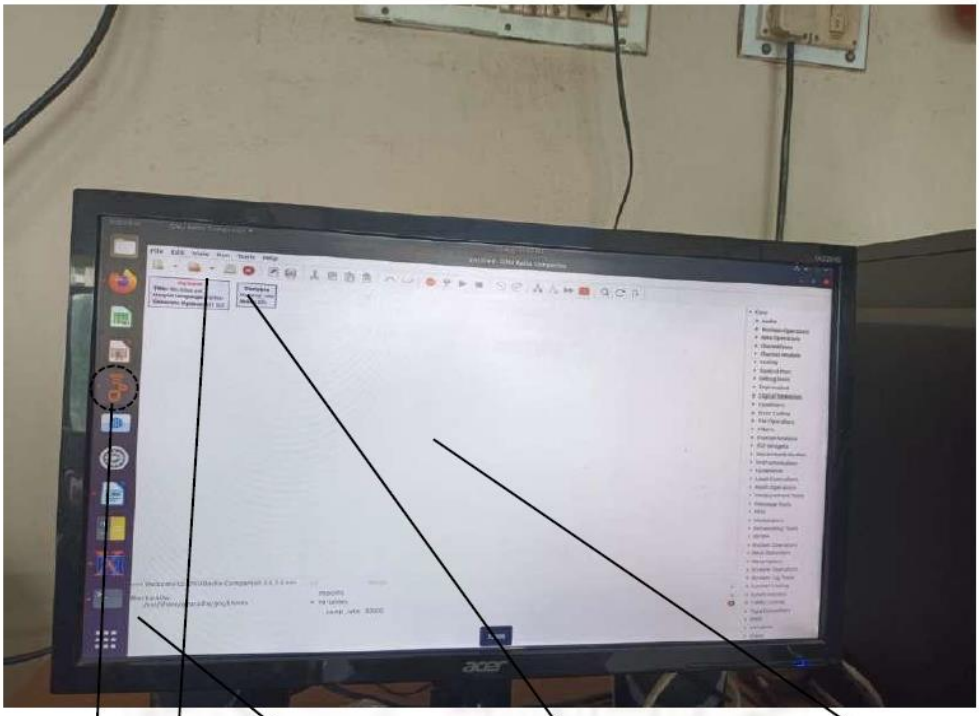


Figure 5. GNU radio

The programs are developed in workspace.

GNU radio programs are called flow graphs. There are two blocks which you will find when you open GNU radio. They are Options block and Variable block

On opening Open Window as shown in figure 6, another window pops up.

In this window you can give title, id, copyright etc. There is a tab called output language. While compiling the flow graph, a program can be generated either in Python or C language by selecting the corresponding output language tab. From generate options select *QT GUI.*

The next block is Variable block. On variable block by default *the id is sampling rate*. We can change the sampling rate to 1 or 2 MHz as we are setting up digital radio.

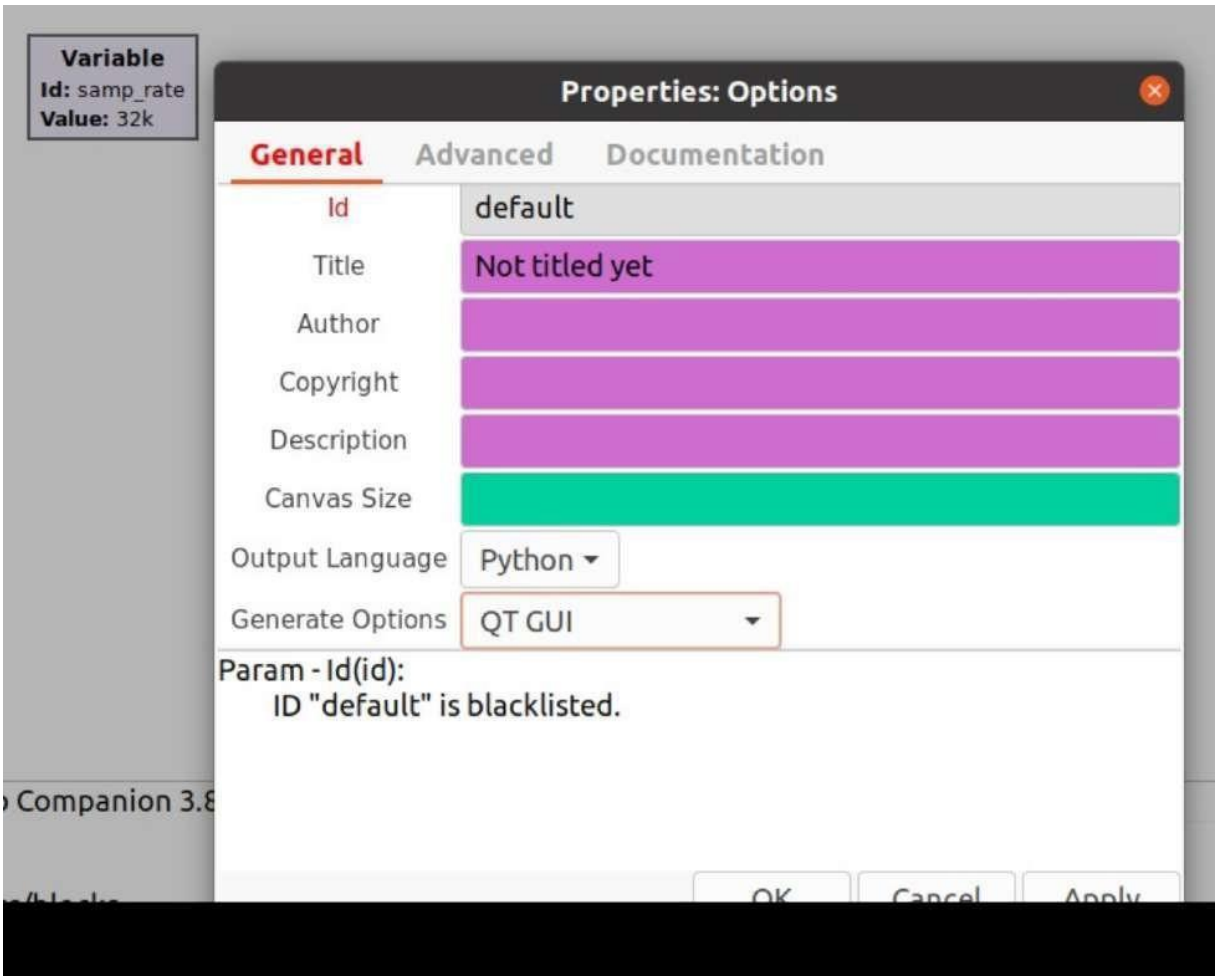


Figure 6. Options window

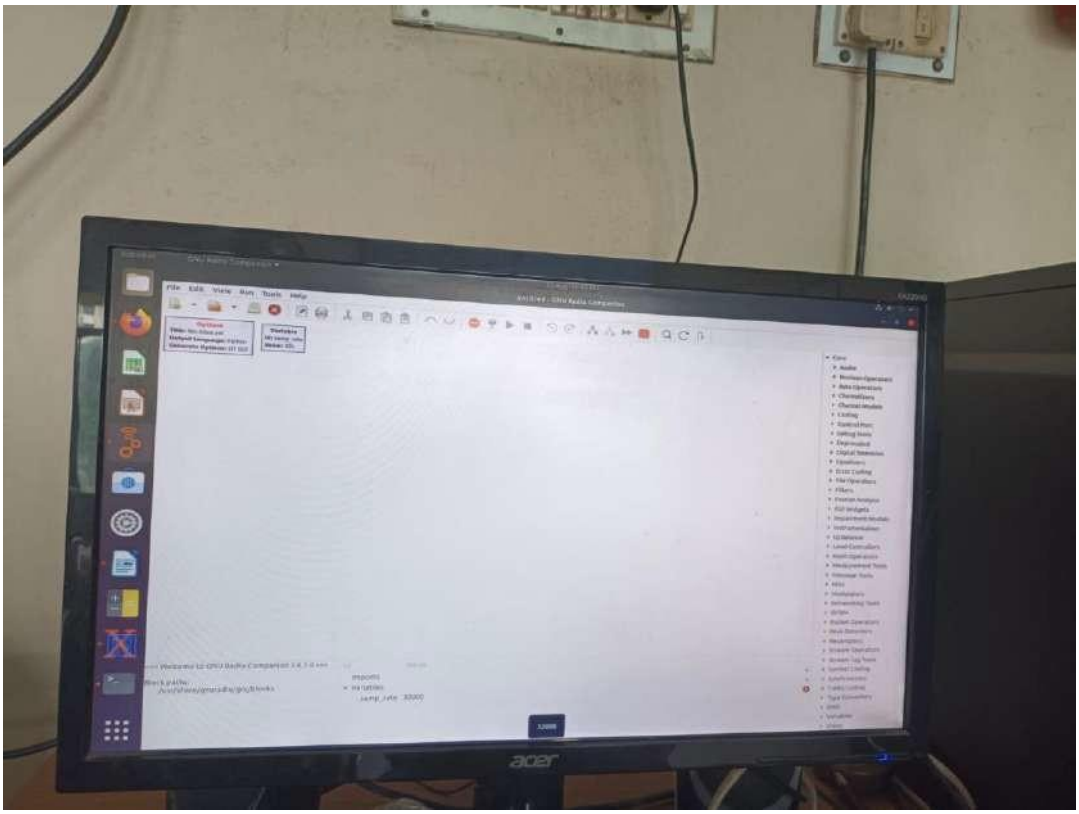


Figure 7

If there is an error in the code, the button marked in figure 7 is active. On clicking this button we will get errors. Then errors have to be rectified, and then only we can compile the program. Press the *RUN* button. There is an option called *GENERATE,* on pressing this, it will save as *.grc file.*

On the right side of the window is a list of the blocks that are available. By expanding any of the categories (click on triangle to the left) you can see the blocks available. Explore each of the categories so that you have an idea of what is available.

From the right side options, open the Waveform generators category and double click on the Signal Source. A Signal Source block will now appear in the main window. Double click on the block and the properties window will open. Adjust the

settings as set to output a real valued 1 KHz sinusoid with amplitude of .5. Change the output type t float. Refer figure 8.

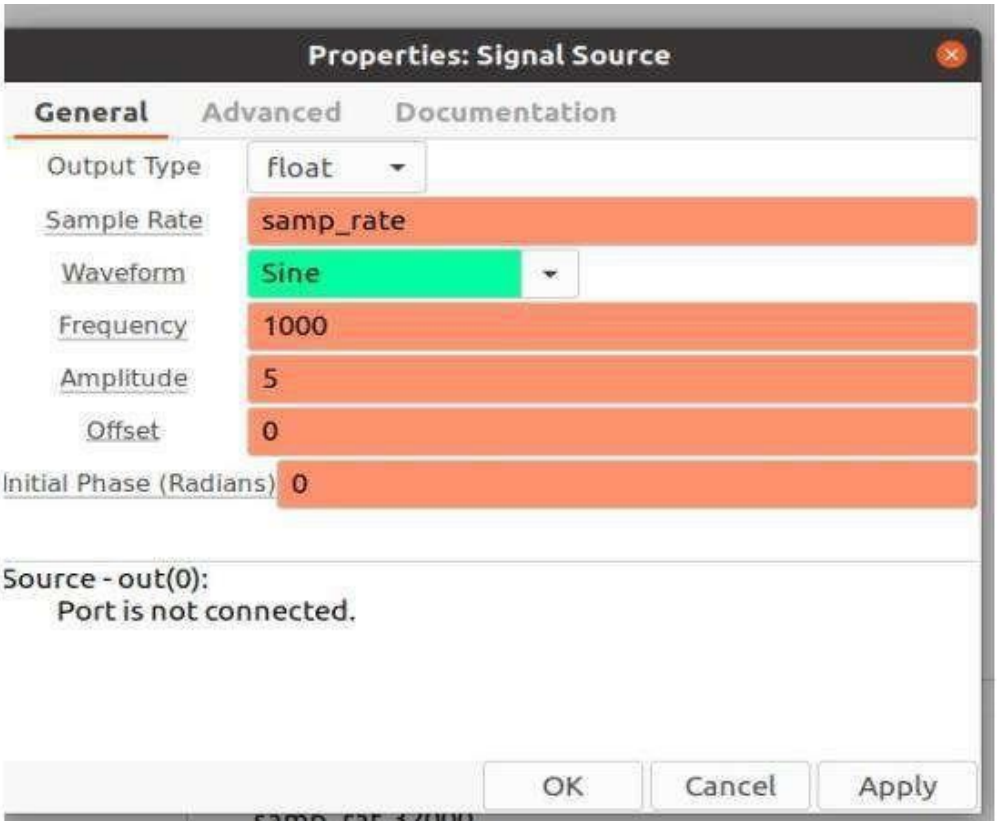


Figure 8

Setting Signal source To view this wave we need one of the graphical sinks. Choose Core→Instrumentation→ QT →QT GUI Time sink as shown in figure 9 and then double click on it.

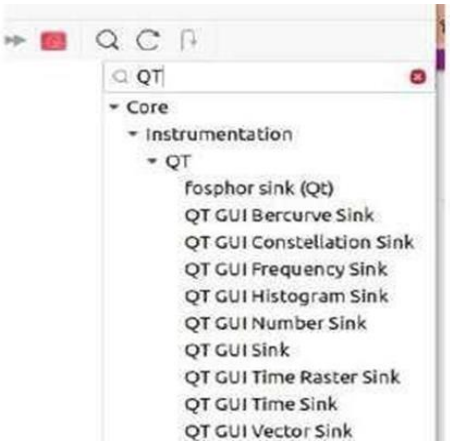


Figure 9 QT Sink

A box will appear. Make connections between them as shown in figure

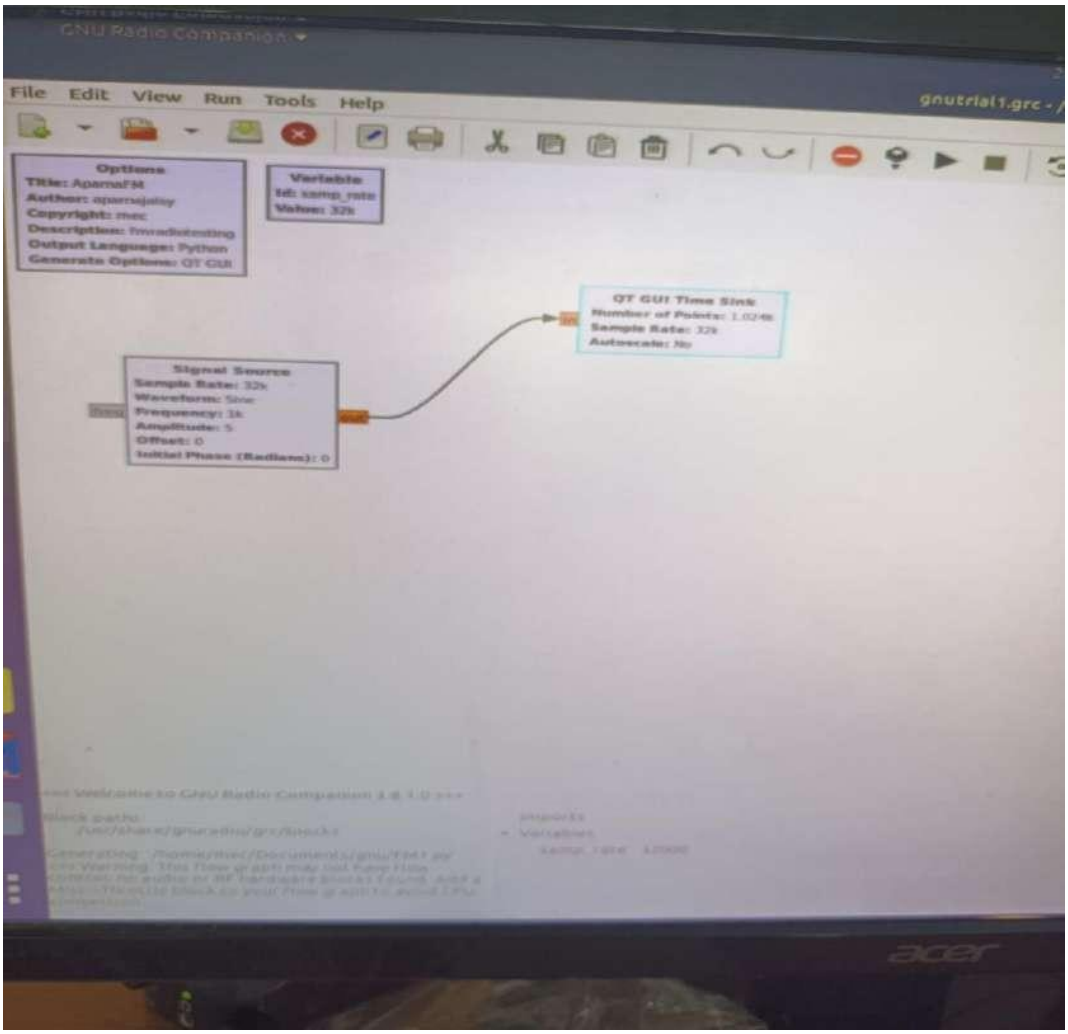


Figure 10.

In order to observe the operation of this simple system we must generate the flow graph and then execute it. Click first on the “Generate the flow graph” icon. A box will come up in which you enter the name of the file. Name this file: test1.grc and save. Click the “Execute the flow graph” icon. A scope plot will open and several cycles of the sine wave are displayed as shown in figure 11.

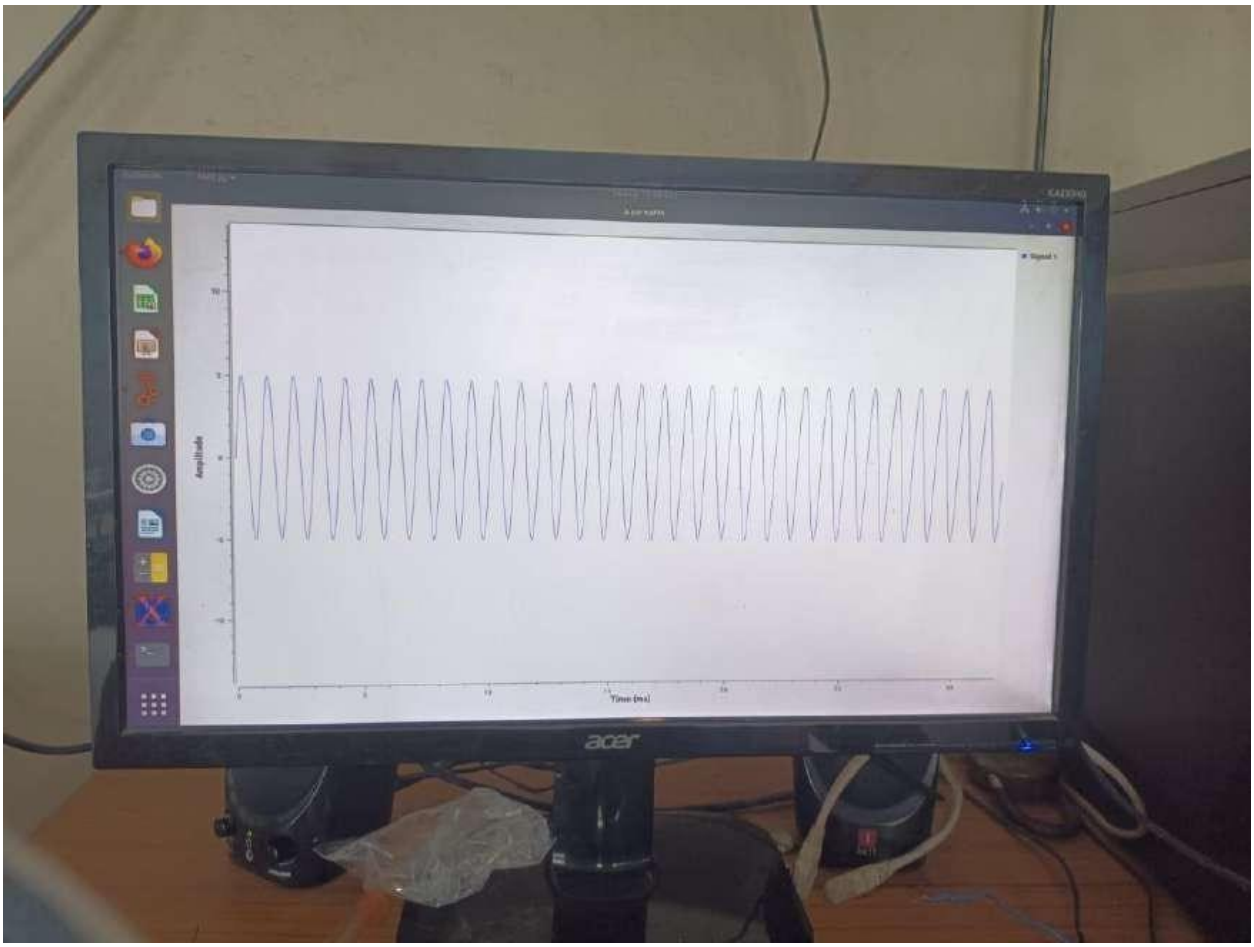


Figure 11.Waveform display

You can also display the frequency spectrum using QTGUI Frequency Sink, Audio sink waterfall Sink etc. The connections are as shown in figure 12 and results in figure 13

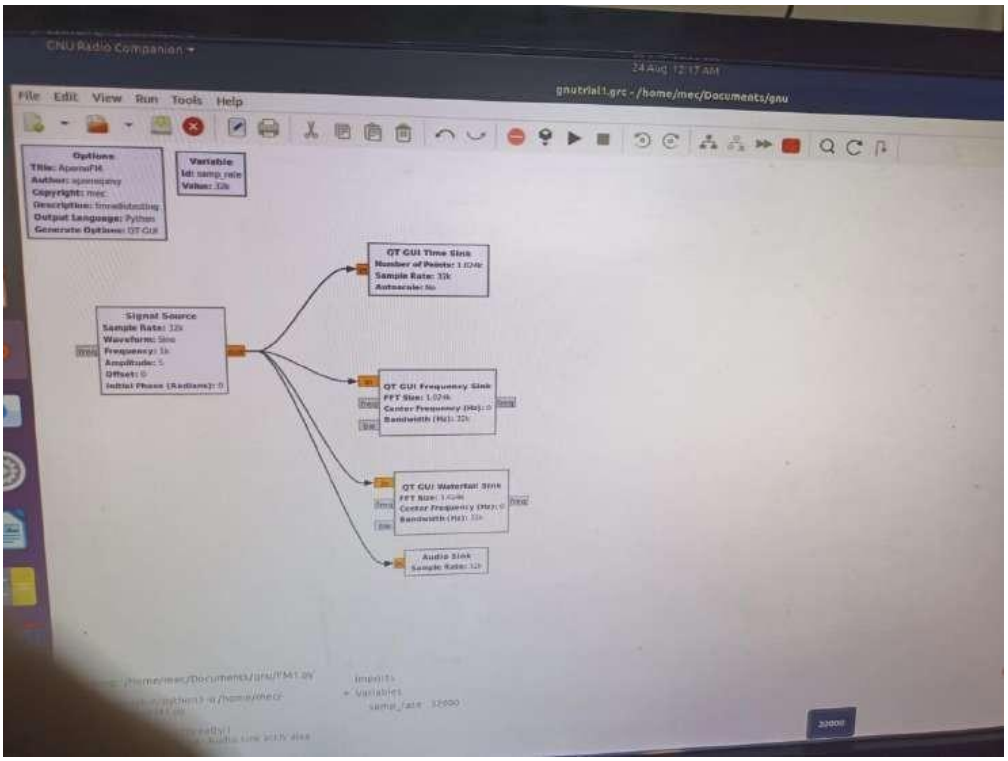


Figure 12

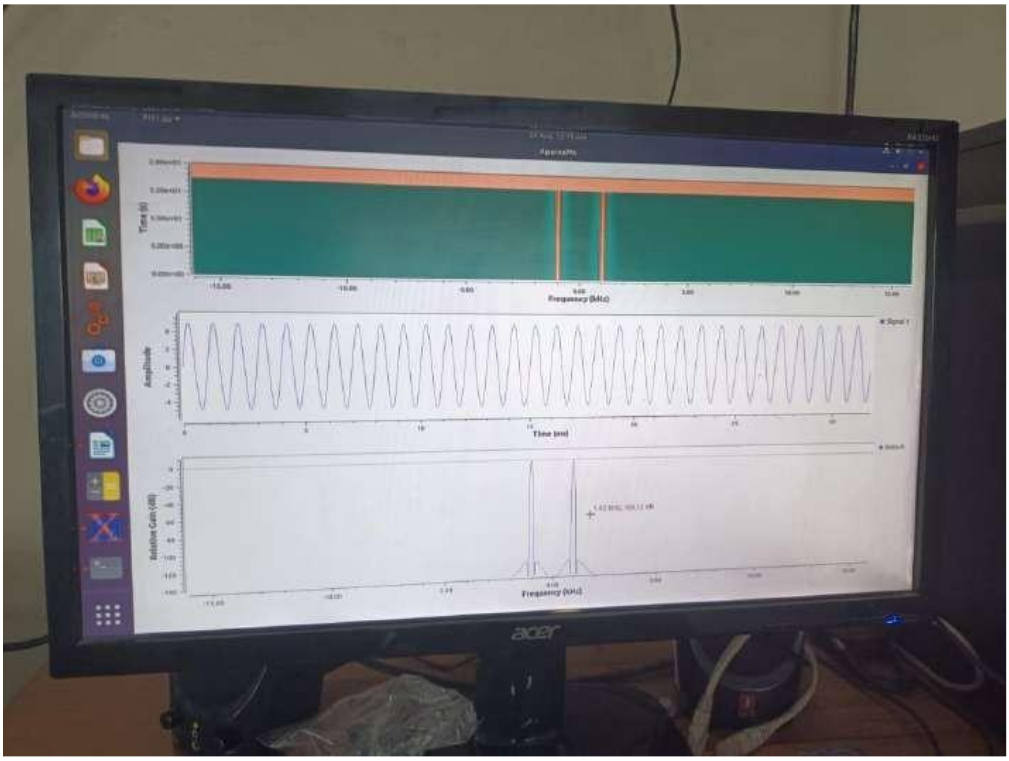


Figure 13

Similarly you can add multiply two different signals and display the result. In addition to saving a ".grc" file with your flow graph, note that there is also a file titled "xxx. py". Double click on this block. You will be given the option to Run or Display this file. Select Display. This is the Python file that is generated by GRC.

RESULT:

Software Defined Radio is familiarized.

Date: 27.03.2024 Experiment No: 10

FM RECEPTION USING SDR

AIM:

To:

1. Receive digitized FM signal (for the clearest channel in the lab) using the SDR board.

2. Set up an LPF and FM receiver using GNU Radio.

3. Use appropriate sink in GNU Radio to display the spectrum of signal.

4. Resample the voice to make it suitable for playing on computer speaker.

PROCEDURE:

Start GNU radio. Set the Options and Variable windows. For Interfacing RTL SDR dongle, insert RTL SDR Source block as shown in Figure 1.

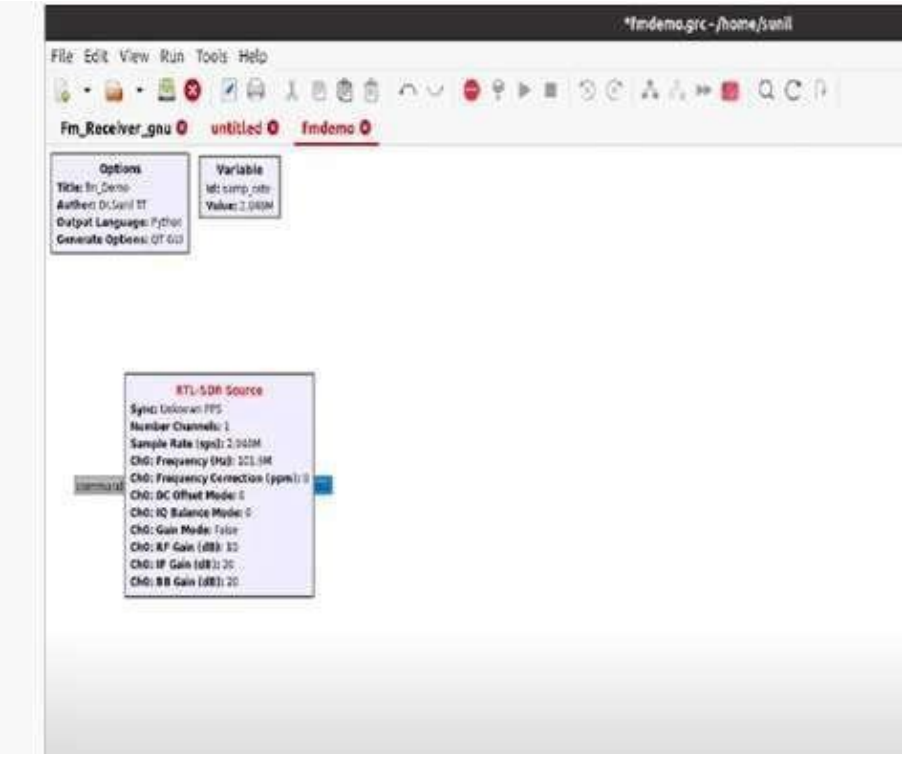


Figure 1

The simplest FM radio consists of few other elements like:

- Rational resampler
- Low pass filter
- WBFM demodulator
- Audio output - your PC's sound card

The setup for FM receiver is shown in Figure 2.

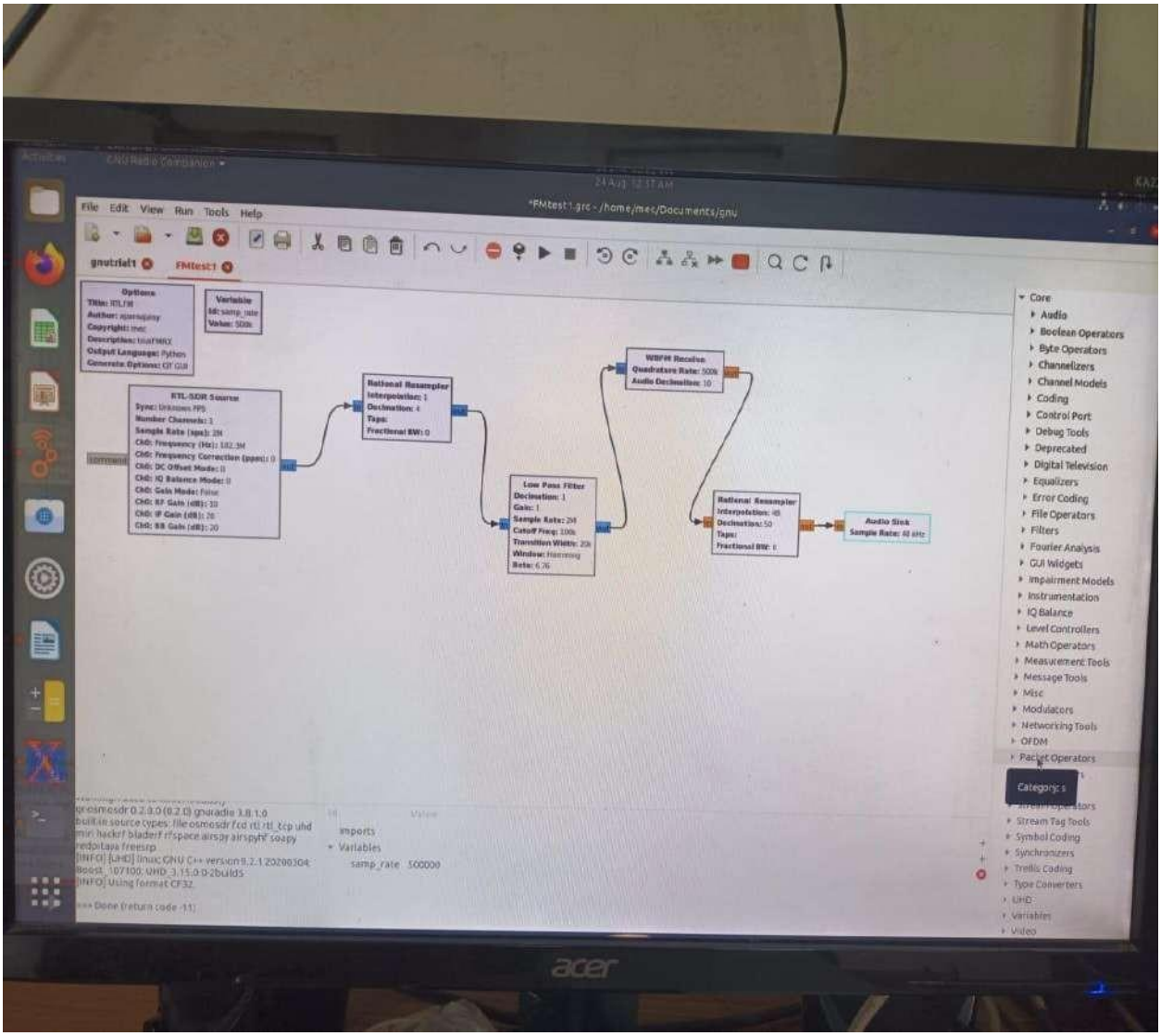


Figure 2

For the first rational resampler, set interpolation factor 1, decimation factor 4.

For Low pass filter, set decimation factor -1 Sample rate -2.04 MHz, Gain -1, Cut off frequency - 100 KHz, Transition Width- 20 K, Window – Hamming, Beta – 6.76.

For WBFM Receive block, set the quadrature rate as 500 KHz and audio decimation as 10. For Second rational resampler, set interpolation factor 48 and decimation factor 50.

Audio sink sample is at 48KHz.

Now run the program. You can hear FM radio station you have set.

We can add slider and water fall diagram to view the spectrum. We can set FM range by QT GUI Range.

RESULT:

FM receiver using Software Defined Radio is set up.