GOVERNMENT COLLEGE OF ENGINEERING
KANNUR

ECL332 COMMUNICATION LAB

SEMESTER VI

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TABLE OF CONTENTS

1. FM generation and demodulation using PLL

AIM: To generate frequency modulated signal and to demodulate the Frequency Modulated signal using PLL.

COMPONENTS REQUIRED: ICs 565, resistors, capacitors, function generator, DSO, DC source and bread board.

THEORY:

In FM, frequency of the carrier signal is varied in accordance with the instantaneous amplitude of the modulating signal. PLL is widely used for FM generation. The frequency generated by the VCO of the PLL is varied by the input signal applied. The resistor R_1 and capacitor C_1 decides the carrier frequency of the FM. AF input is capacitively coupled to pin 7 of the IC using R_3-C_c network.

For the demodulation of the FM signal, the center frequency of the VCO of the demodulator is made the same as that of the modulator. The variation of the input frequency from the center frequency is converted into the variation of the voltage by the phase comparator of the demodulator. FM input is coupled to the pin 2 using R_3-C_c network as shown in figure.2

Major advantage of FM demodulator using PLL is linearity. Linearity is governed by the voltage to frequency characteristics of the VCO within the PLL. As the frequency deviation of the incoming signal swings over a small portion of the PLL bandwidth, and the characteristics of the VCO can be made relatively linear, the distortion levels from PLL demodulators are normally very low.

CIRCUIT DIAGRAM:

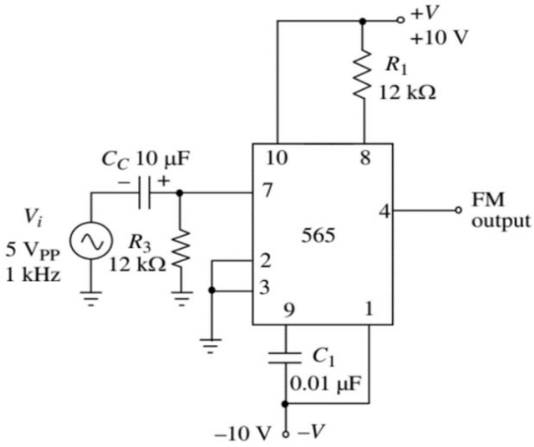


Figure1. FM generation using IC565

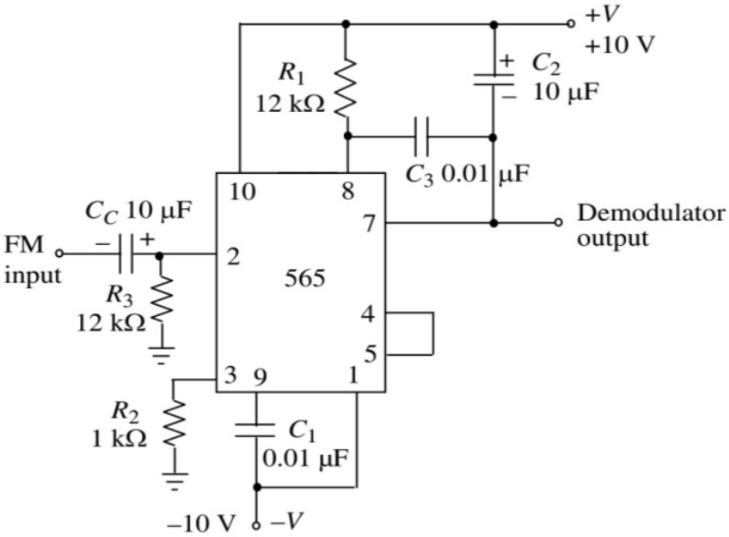


Figure 2. FM Demodulation using IC565

DESIGN

Let V+ =10 V and V- = -10

Let the center frequency of the FM be f_o=(1.2/4*R1*C1) =2.5 kHz

Take C_1=0.01µF Then we get R_1=12 kΩ

The value of R_1 satisfies the required condition 2 kΩ < R_1 < 20 kΩ, as per data sheet. Take C_c=10 µF and R_3=12 kΩ to couple the input to the IC.

For demodulator circuit, since center frequency is same as that of modulator, frequency determining components are the same.

Take same set of coupling capacitor and resistor.

Take R_2=1 kΩ, C_2=10 µF and C_3=0.01 µF

PROCEDURE

1. Set up the FM generator circuit and apply 5 Vpp, 1KHz sinewave input and observe the output.

2. Note maximum and minimum frequencies, f_{max} and f_{min} of FM output. Calculate frequency deviation $\Delta f= f_{max} - f_{min}$. Calculate the modulation index $m_i = \Delta f/f_m$ where f_m is modulating signal frequency.

3. Set up FM demodulator and apply the FM signal to it. Observe the demodulated output.

SAMPLE WAVEFORMS

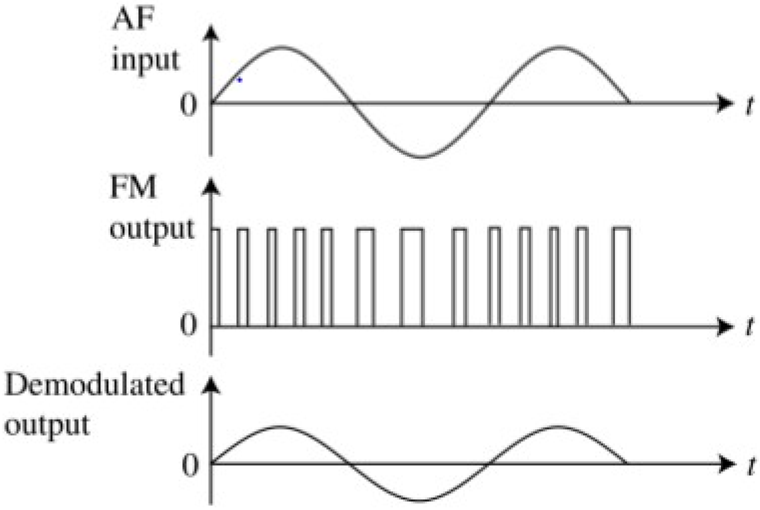


Figure 3: Modulated and demodulated output waveforms.

RESULT:

2 Binary Phase Shift Keying

AIM

To design and set up a BPSK modulator

COMPONENTS AND EQUIPMENTS REQUIRED

Equipments / Components	Specification / Range	Quantity
Resistors ICs Transistor Signal generators DC supplies Bread board CRO		

THEORY

In binary phase shift keying, the binary symbols 1 and 0 modulate the phase of the carrier. Let the carrier signal for transmitting the binary symbol ‘1’ be $s_1(t)=A\cos(2лf_ct)$. For the transmission of symbol ‘0’, phase of the carrier is shifted by an amount of 180° (p radians). So $s_2\ (t) = -A\ \cos(2лf_ct) = A\ \cos(2лf_ct + л)$. From the above equations, BPSK signal can be defined as $s(t) = b(t)\ A\cos(2лf_ct)$, where $b(t) = +1$ to transmit symbol ‘1’ and $b(t) = –1$ to transmit symbol ‘0’.

Modulator: This circuit mainly consists of a unity gain op-amp inverting amplifier and two switches. Inverting amplifier provides 180° phase-shift to carrier signal s(t). The binary symbols are applied to the control inputs of two switches. An op-amp inverting Zero Crossing Detector (ZCD) enables to apply symbol ‘1’ at the control input of one switch while ‘0’ at the control input of the other switch. BPSK signal has constant amplitude as in the case of BFSK signal. Therefore, the noise can be removed easily.

Demodulator: BPSK signal can be demodulated non-coherently by converting it to BASK signal and then by using an envelope detector. The original carrier is added with BPSK signal using a summing amplifier to obtain BASK signal. BASK signal is further demodulated using an envelope detector. The output of the envelope detector may not be square-shaped. Therefore, the demodulated signal is converted to a square-shaped signal with the help of a comparator

CIRCUIT DIAGRAM

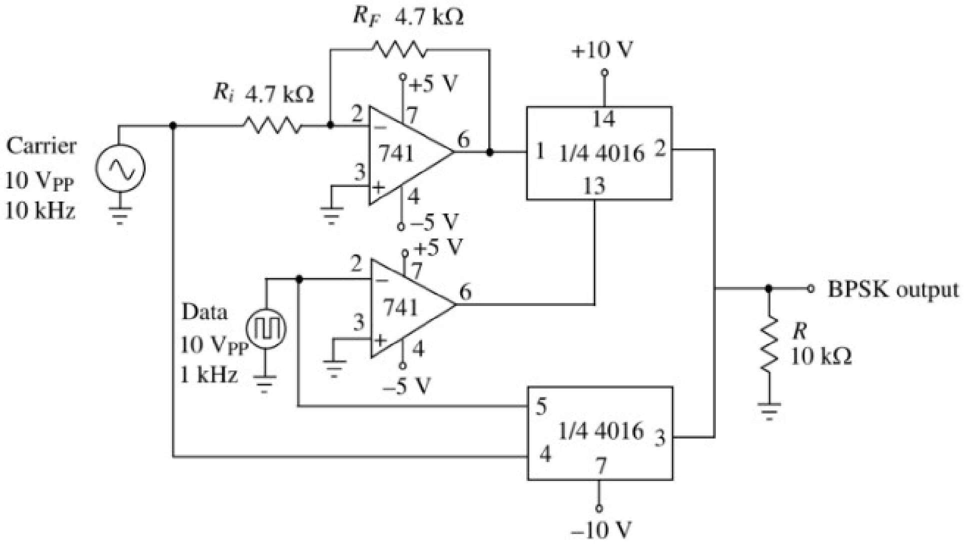


Figure: BPSK Modulator

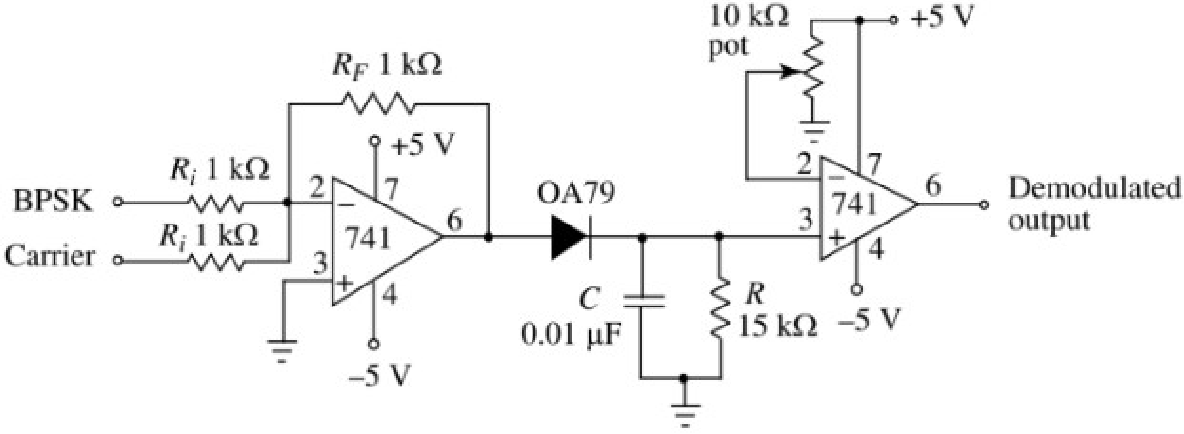


Figure: BPSK Demodulator

DESIGN

Modulator: Gain of inverting amplifier A= -R_F/R_i.
Let the Gain be -1 so that the ratio R_F/R_i=1. Take $R_i = R_F$ =4.7 kΩ

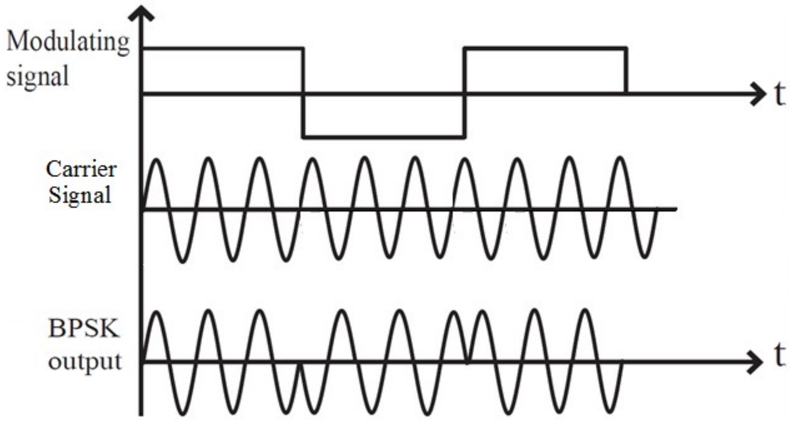
Demodulator: Use 1 kΩ resistor as R_F and R_i to get unity gains for the adder circuit. Carrier frequency is 10 kHz and data input frequency are 1kHz.

We know, $f = \frac{1}{2\pi RC}$ =1 kHz

If C=0.01μF, then R= 15 kΩ

Use a 10kΩ pot for setting the threshold voltage for the comparator.

WAVEFORMS:



-

PROCEDURE:

1. Setup the circuit part by part and verify their function.
2. Connect the parts together and observe the output waveform on CRO.

RESULT:

3 DELTA MODULATION AND DEMODULATION

AIM

To design, setup and study a Delta Modulator and Demodulator circuit.

COMPONENTS AND EQUIPMENTS REQUIRED

Equipment / Components	Specification / Range	Quantity
Resistors Capacitors IC's Signal generators DC Supplies Bread board CRO		

THEORY

Delta Modulation (DM) is a differential PCM scheme in which the difference signal is encoded into a single bit. This single bit is transmitted per sample to indicate whether the signal is larger or smaller than the previous sample. The difference between the input and the approximation is quantized into two levels ±Δ corresponding to positive or negative difference respectively. Here in the circuit the modulating signal m(t) and its quantized approximation m^(t) are applied to the comparator. Comparator provides a high-level output when m(t) > m^(t) and provides low level output when m(t) < m^(t). The output of the comparator is fed to a sample and hold circuit made by a D flip flop. Pulses at the output of D flip flop is made bipolar by an op-amp comparator. IC 741 is used as the comparator whose output becomes +5 V when the voltage at its pin 3 is greater than +0.5 V. The demodulator consists of an integrator and a low pass filter

CIRCUIT DIAGRAM

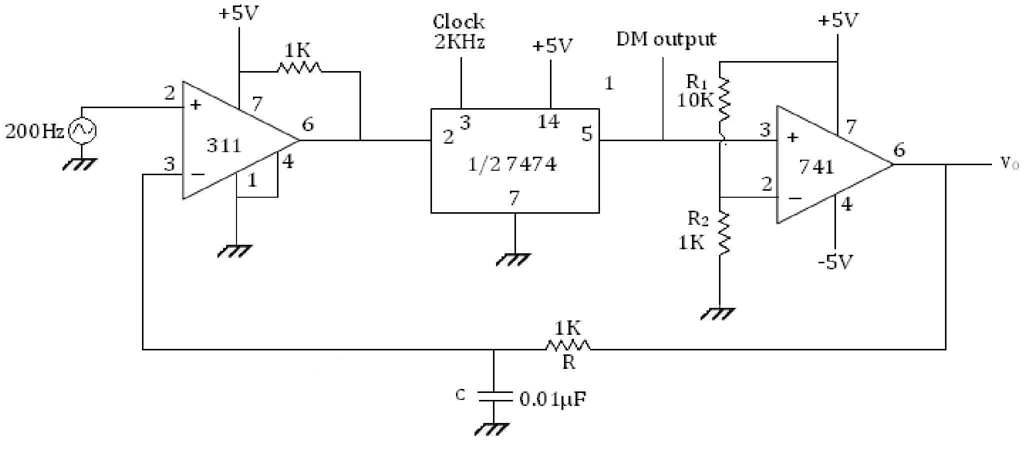


Figure1: Delta modulator

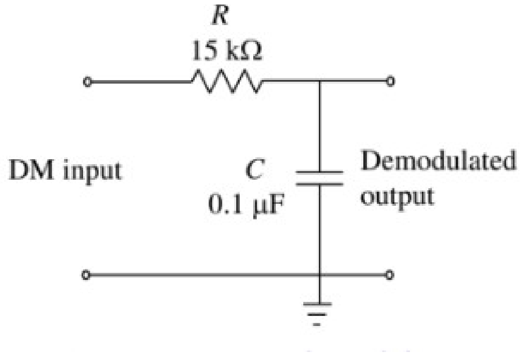


Figure 2: Delta demodulator

DESIGN

Let the input signal amplitude be 5V and frequency be 200 Hz.
i.e., m(t) = 5sin400πt
Maximum slope of m(t) = 2πfA = 2π2000 X 5 = 2000 π
To avoid slope overload error, slope of $\overline{m}(t)$ should be more than that of m(t).

$$Vcc/RC > E_m \omega_m$$

Selecting Vcc =15V and C = 0.01 μF, we get$R < 228\,K$. Take R = 100K.

Therefore $\frac{Vcc\,R_2}{R_1+R_2}$ = 1.35 V.

Take R_2 =1K. Then R_1 = 10K.
Let the clock frequency be 2KHz.

WAVEFORMS:

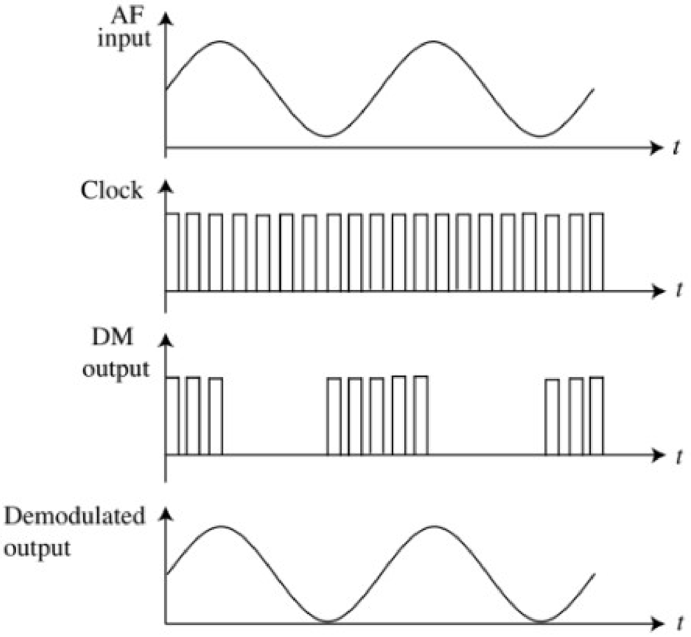


Figure3: Waveforms

PROCEDURE

1. Setup the circuit after verifying the components.
2. Feed a unipolar 5V, 200 Hz sine wave to the input using offset knob of function generator. Set the clock frequency at 2KHz.
3. Observe the DM output and Vo simultaneously on CRO.

RESULT

Part B

1: Error Performance of Binary Phase Shift Keying

AIM

1. Generate a string of message bits.
2. Encode using BPSK with energy per bit E_b and represent it using points in a signal-space.
3. Simulate transmission of the BPSK modulated signal via an AWGN channel with variance $N_0/2$.
4. Detect using an ML decoder and plot the probability of error as a function of SNR per bit E_b/N_0.

THEORY

Binary Phase Shift Keying

In a coherent binary PSK system, the pair of signals $S_1(t)$ and S_2 (t) used to represent binary symbols 1 & 0 are defined by

$S_1 (t) = \sqrt{}2E_b/\tau_b$ Cos $2\pi f_c t$

$S_2 (t) =\sqrt{}2E_b/T_b (2\pi f_c t+\pi) = - \sqrt{}\ 2E_b/T_b$ Cos $2\pi f_c t$ where $0 \leq t< T_b$ and

E_b = Transmitted signed energy for bit

The carrier frequency $f_c =n/T_b$ for some fixed integer n.

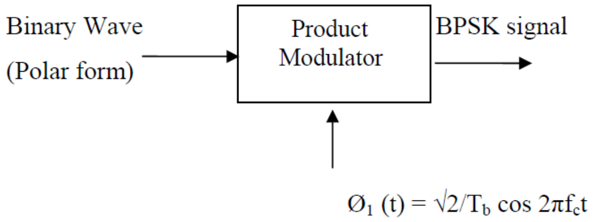
In BPSK, there is only one basis function of unit energy.

$Ø_b (t) = \sqrt{}2/T_b$ cos $2f\pi_c t$ $0 \leq t< T_b$

$S_1 (t) = \sqrt{}E_b\, Ø1 (t)\ 0 \leq\ t\ \leq T_b$
$S_2 (t) = \sqrt{}E_b\, Ø1 (t)\ 0 \leq\ t< T_b$
The signal space is 1dimensional (N=1) having two message points (M = 2)

Block Diagram of BPSK Transmitter

The input binary symbols are represented in polar form with symbols 1 & 0 represented by constant amplitude levels $\sqrt{}E_b$ & $-\sqrt{}E_b$. This binary wave is multiplied by a sinusoidal carrier in a product modulator. The result in a BSPK signal.



BSPK Receiver:

The received BPSK signal is applied to a correlator which is also supplied with a locally generated reference signal $Ø_1$ (t). The correlated output is compared with a threshold of zero volts. If $x_1> 0$, the receiver decides in favour of symbol 1. If $x_1< 0$, it decides in favour of symbol 0.

ALGORITHM

1. Generate a sequence of random bits of ones and zeros of certain length (N_{sym} typically set in the order of 10000)
2. Using the constellation points, map the bits to modulated symbols (For example, bit ‘0’ is mapped to amplitude value ***A***, and bit ‘1’ is mapped to amplitude value ***-A***)
3. Compute the total power in the sequence of modulated symbols and add noise for the given E_bN_0 (SNR) value. The noise added symbols are the received symbols at the receiver.
4. Use thresholding technique, to detect the bits in the receiver. Based on the constellation diagram above, the detector at the receiver has to decide whether the receiver bit is above or below the threshold 0.
5. Compare the detected bits against the transmitted bits and compute the bit error rate (BER).
6. Plot the simulated BER against the SNR values and compare it with the theoretical BER curve for BPSK over AWGN

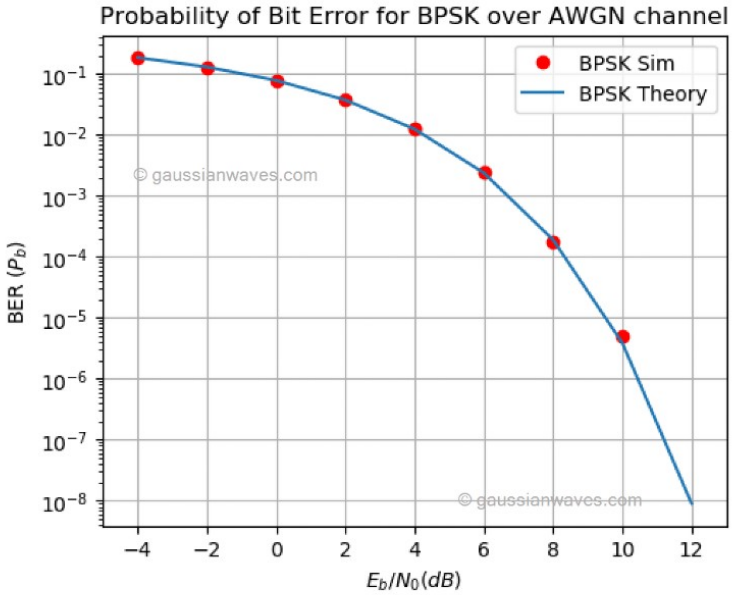
MATLAB CODE

```
% Demonstration of Eb/N0 Vs BER for BPSK modulation scheme

clear;

clc;

%---------Input Fields------------------------

N=10000000; %Number of input bits

EbN0dB = -6:2:10; % Eb/N0 range in dB for simulation
```

```
%--------------------------------------------
data=randn(1,N)>=0; %Generating a uniformly distributed random 1s and 0s
bpskModulated = 2*data-1; %Mapping 0->-1 and 1->1
M=2; %Number of Constellation points M=2^k for BPSK k=1
Rm=log2(M); %Rm=log2(M) for BPSK M=2
Rc=1; %Rc = code rate for a coded system. Since no coding is used Rc=1
BER = zeros(1,length(EbN0dB)); %Place holder for BER values for each Eb/N0
index=1;
for k=EbN0dB,
%------------------------------------------
%Channel Noise for various Eb/N0
%------------------------------------------
%Adding noise with variance according to the required Eb/N0
EbN0 = 10.^(k/10); %Converting Eb/N0 dB value to linear scale
noiseSigma = sqrt(1./(2*Rm*Rc*EbN0)); %Standard deviation for AWGN Noise
noise = noiseSigma*randn(1,length(bpskModulated));
received = bpskModulated + noise;
%------------------------------------------
%Threshold Detector
estimatedBits=(received>=0);
%------------------------------------------
%Bit Error rate Calculation
BER(index) = sum(xor(data,estimatedBits))/length(data);
index=index+1;
end
%Plot commands follows
plotHandle=plot(EbN0dB,log10(BER),'r--');
set(plotHandle,'LineWidth',1.5);
title('SNR per bit (Eb/N0) Vs BER Curve for BPSK Modulation Scheme');
xlabel('SNR per bit (Eb/N0) in dB');
ylabel('Bit Error Rate (BER) in dB');
grid on;
```

```
hold on;
theoreticalBER = 0.5*erfc(sqrt(10.^(EbN0dB/10)));
plotHandle=plot(EbN0dB,log10(theoreticalBER),'k*');
set(plotHandle,'LineWidth',1.5);
legend('Simulated','Theoretical');
grid on;
```

SAMPLE GRAPH



PYTHON CODE

```
#Eb/N0 Vs BER for BPSK over AWGN (complex baseband model)

import numpy as np #for numerical computing
import matplotlib.pyplot as plt #for plotting functions
from scipy.special import erfc #erfc/Q function

#---------Input Fields------------------------
nSym = 10**5 # Number of symbols to transmit
EbN0dBs = np.arange(start=-4,stop = 13, step = 2) # Eb/N0 range in dB for simulation
BER_sim = np.zeros(len(EbN0dBs)) # simulated Bit error rates

M=2 #Number of points in BPSK constellation
```

```
m = np.arange(0,M) #all possible input symbols
A = 1; #amplitude
constellation = A*np.cos(m/M*2*np.pi)  #reference constellation for BPSK

#------------ Transmitter---------------
inputSyms = np.random.randint(low=0, high = M, size=nSym) #Random 1's and 0's as input to
BPSK modulator
s = constellation[inputSyms] #modulated symbols

fig, ax1 = plt.subplots(nrows=1,ncols = 1)
ax1.plot(np.real(constellation),np.imag(constellation),'*')

#----------- Channel --------------
#Compute power in modulatedSyms and add AWGN noise for given SNRs
for j,EbN0dB in enumerate(EbN0dBs):
   gamma = 10**(EbN0dB/10) #SNRs to linear scale
   P=sum(abs(s)**2)/len(s) #Actual power in the vector
   N0=P/gamma # Find the noise spectral density
   n = np.sqrt(N0/2)*np.random.standard_normal(s.shape) # computed noise vector
   r = s + n # received signal

   #------------- Receiver ------------
   detectedSyms = (r <= 0).astype(int) #thresolding at value 0
   BER_sim[j] = np.sum(detectedSyms != inputSyms)/nSym #calculate BER

BER_theory = 0.5*erfc(np.sqrt(10**(EbN0dBs/10)))

fig, ax = plt.subplots(nrows=1,ncols = 1)
ax.semilogy(EbN0dBs,BER_sim,color='r',marker='o',linestyle='',label='BPSK Sim')
ax.semilogy(EbN0dBs,BER_theory,marker='',linestyle='-',label='BPSK Theory')
ax.set_xlabel('$E_b/N_0(dB)$');ax.set_ylabel('BER ($P_b$)')
ax.set_title('Probability of Bit Error for BPSK over AWGN channel')
ax.set_xlim(-5,13);ax.grid(True);
ax.legend();plt.show()
```

2: Error Performance of QPSK

AIM

1. Generate a string of message bits.
2. Encode using QPSK with energy per symbol Es and represent it using points in a signal-space.
3. Simulate transmission of the QPSK modulated signal via an AWGN channel with variance N0/2 in both I-channel and Q-channel.
4. 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

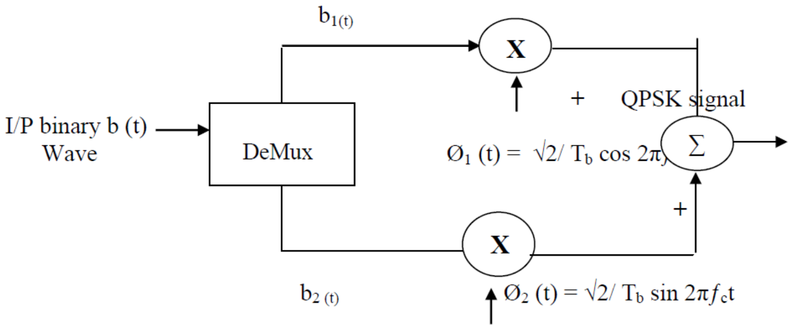
Phase of the carrier takes on one of four equally spaced values such as $\pi/4$, $3\pi/4$, $5\pi/4$,$7\pi/4$.

$S_i(t) = \sqrt{2E/T_b} \cos\{2\pi f_c t + (2i-1)\pi/4\}$, $0 \le t \le T_b$
0 , elsewhere
Where i = 1,2,3,4, & E= Tx signal energy per symbol
Tb = symbol duration
Each of the possible value of phase corresponds to a pair of bits called dibits.
Thus the gray encoded set of dibits: 10,00,01,11
$S_i(t) = \sqrt{2E/T_b} \cos[(2i-1)\pi/4] \cos(2\pi f_c t) - \sqrt{2E/T_b} \sin[(2i-1)\pi/4)] \sin(2\pi f_c t)$,$0 \le t \le T_b$ 0 , else where
There are two orthonormal basis functions
$Ø_1(t) = \sqrt{2/T_b} \cos 2\pi f_c t$, $0 \le t \le T_b$
$Ø_2(t) = \sqrt{2/T_b} \sin 2\pi f_c t$, $0 \le t \le T_b$

There are four message points

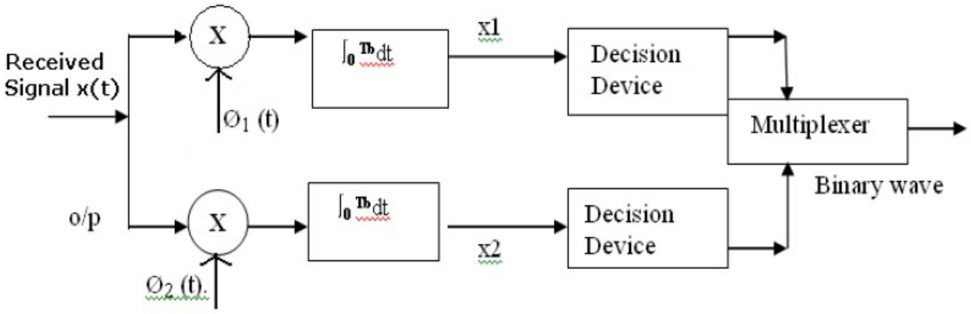
Input debits	Phase of QPSK signal	Co-ordinates of message signals	
		S1	S2
10	$\pi/4$	$\sqrt{E/2}$	$-\sqrt{E/2}$
00	$3\pi/4$	$-\sqrt{E/2}$	$-\sqrt{E/2}$
01	$5\pi/4$	$-\sqrt{E/2}$	$+\sqrt{E/2}$
11	$7\pi/4$	$+\sqrt{E/2}$	$+\sqrt{E/2}$

QPSK Transmitter



The i/p binary sequence b(t) is represented in polar from with symbols 1 & 0 represented as $+\sqrt{E/2}$ and $-\sqrt{E/2}$ .This binary wave is demultiplexed into two separate binary waves consisting of odd & even numbered I/P bits denoted by b1(t) & b2 (t). b1 (t) & b2(t) are used to modulate a pair of quadrature carrier or orthogonal Basis function Ø1 (t) & Ø2 (t). The result is two PSK waves' .These two binary PSK waves are added to produce the desired QPSK signal .

QPSK Receiver



QPSK receiver consists of a pair of correlators with common I/P & supplied with Locally generated Signal Ø1 (t) & Ø2 (t). The correlator O/P, x1, & x2 are each compared with a threshold of zero volts.If x1 > 0, decision is made in favour of symbol '1' for upper channel and if x1 > 0, decision is made in favour of symbol 0. Parallelly Y x2 >0, decision is made in favour of symbol 1 for lower channel & if x2 <0, decision is made in favour of symbol 0. These two channels are combined in a multiplexer to get the original binary output.

ALGORITHM

QPSK Modulation

1. Generate two carrier signals (Ø1 (t) = √2/Tb cos 2πfct and Ø2 (t) = √2/Tb cos2πfct)
2. Generate the base band data signal .
3. Binary wave is divided into odd(b1(t)) and even(b2(t)) numbered input bits.
4. Multiply the odd numbered data signal (b1(t)) and carrier signal 1 in one channel .
5. Multiply the even numbered data signal b2 (t)and carrier signal 2in another channel
6. Sum the output resultant signals of step 4 and 5.
7. The resultant signal is a QPSK signal
8. Plot the carrier, data and QPSK signal.

QPSK Demodulation

1. Multiply the received QPSK signal with the carrier signal Ø1 (t) = √2/Tb cos 2πfct in one channel and integrate the resultant signal(x1)
2. Multiply the received QPSK signal with the carrier signal Ø2 (t) =√2/Tb cos 2πf ct in another channel and integrate the resultant signal(x2)
3. If x1 is greater than zero then choose 1 and if it is less than 0 choose 0.
4. If x1 is greater than zero then choose 1 and if it is less than 0 choose 0
5. Multiply the resultant signal from step 3 and 4.
6. Plot the demodulated signal.

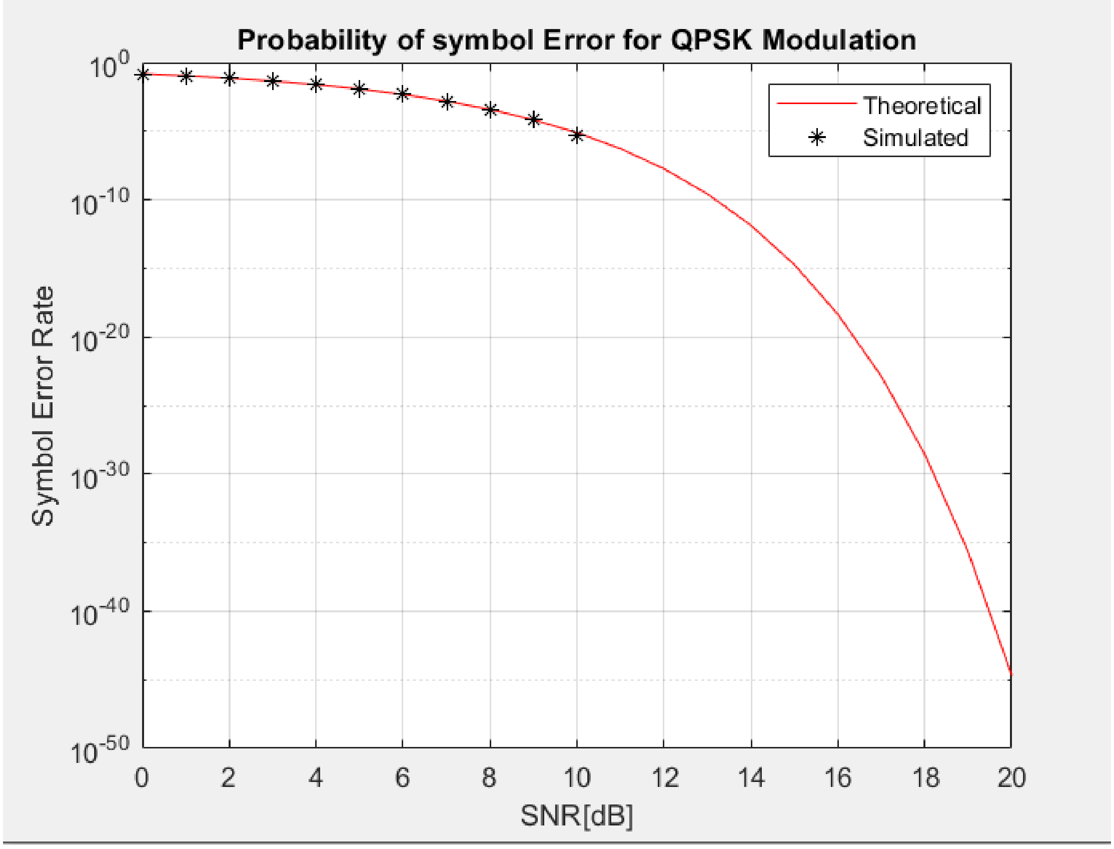
MATLAB CODE

```
% QPSK MODULATION AND BER ESTIMATION IN AWGN CHANNEL
```

```
clc; clear all; close all;
N=1e6; % Number of bits transmited
SNRdB= 0:1:20; % SNR for simulation
SNRlin=10.^(SNRdB/10);
BER = zeros(1,length(SNRlin));% simulated BER
SER = zeros(1,length(SNRlin));% simulated SER
b1 = rand(1,N) > 0.5;
b2 = rand(1,N) > 0.5;
% QPSK symbol mapping
I = (2*b1) - 1;
Q = (2*b2) - 1;
S = I + 1j*Q;
N0 = 1./SNRlin; % Variance
for k = 1:length(SNRdB)

   noise = sqrt(N0(k)/2)*(randn(1,N) + 1j*randn(1,N)); % AWGN noise

   sig_Rx = S + noise; % Recived signal

   % For BER calculation
   sig_I = real(sig_Rx); % I component
   sig_Q = imag(sig_Rx); % Q component

   bld_I = sig_I > 0; % I decision
   bld_Q = sig_Q > 0; % Q decision

   b1_error = (bld_I ~= b1); % Inphase bit error
   b2_error = (bld_Q ~= b2); % Quadrature bit error

   Error_bit = sum(b1_error) + sum(b2_error); % Total bit error
   BER(k) = sum(Error_bit)/(2*N); % Simulated BER

   % For SER calculation
   error_symbol = or(b1_error, b2_error); % if bit in I or bit in Q either wrong than error
   SER(k) = sum(error_symbol)/N;

end
BER_theo = 2*qfunc(sqrt(2*SNRlin)); % Theoretical BER
SER_theo = 2*qfunc(sqrt(2*SNRlin)) - (qfunc(sqrt(2*SNRlin))).^2; % Theoretical SER
figure(1);
semilogy(SNRdB, BER_theo,'r-')
hold on
semilogy(SNRdB, BER,'k*')
xlabel('SNR[dB]')
ylabel('Bit Error Rate');
legend('Theoretical', 'Simulated');
title(['Probability of Bit Error for QPSK Modulation']);
grid on;
hold off;
```

```
figure(2);
semilogy(SNRdB, SER_theo,'r-')
hold on
semilogy(SNRdB, SER,'k*')
xlabel('SNR[dB]')
ylabel('Symbol Error Rate');
legend('Theoretical', 'Simulated');
title(['Probability of symbol Error for QPSK Modulation']);
grid on;
hold off;
```

SAMPLE GRAPH



3 Performance of Waveform Coding Using PCM

AIM

1. Generate a sinusoidal waveform with a DC offset so that it takes only positive amplitude value.
2. Sample and quantize the signal using an 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

In Pulse code modulation (PCM) only certain discrete values are allowed for the modulating signals. The modulating signal is sampled, as in other forms of pulse modulation. But any sample falling within a specified range of values is assigned a discrete value. Each value is assigned a pattern of pulses and the signal transmitted by means of this code. The electronic circuit that produces the coded pulse train from the modulating waveform is termed a coder or encoder. A suitable decoder must be used at the receiver in order to extract the original information from the transmitted pulse train.

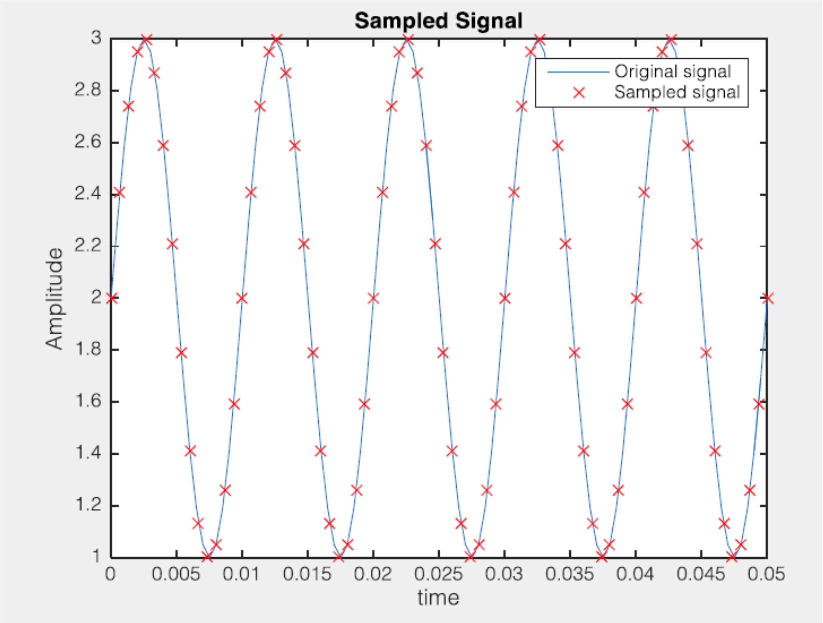
ALGORITHM:

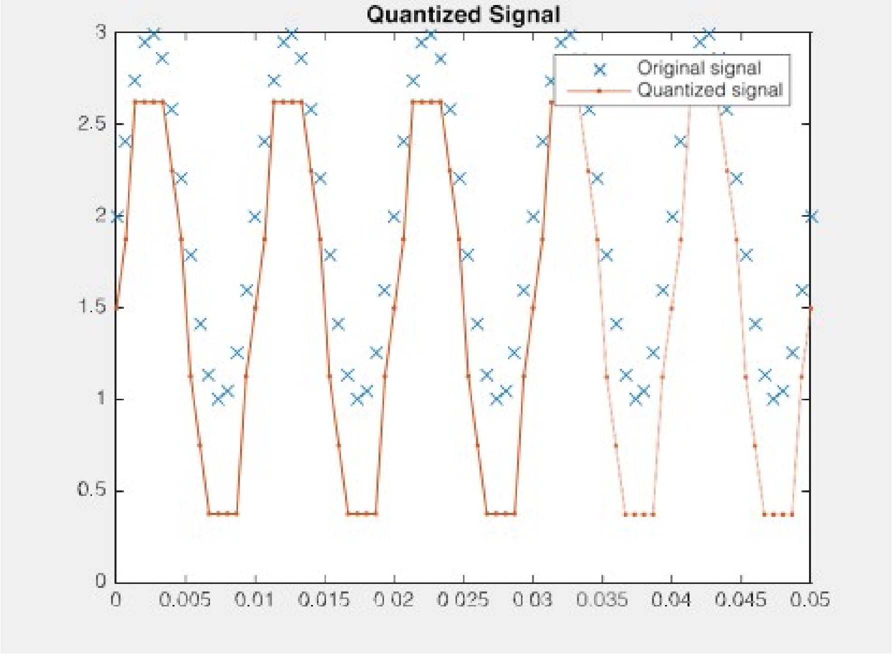
1. Generate a sinusoidal signal with a DC offset of 2.
2. Sample the generated signal.
3. Pass the sampled signal through a quantizer with quantization level=8
4. Encode the signal.
5. Calculate the quantization error and SNR.
6. Vary L and plot the SNR versus number of bits per symbol.

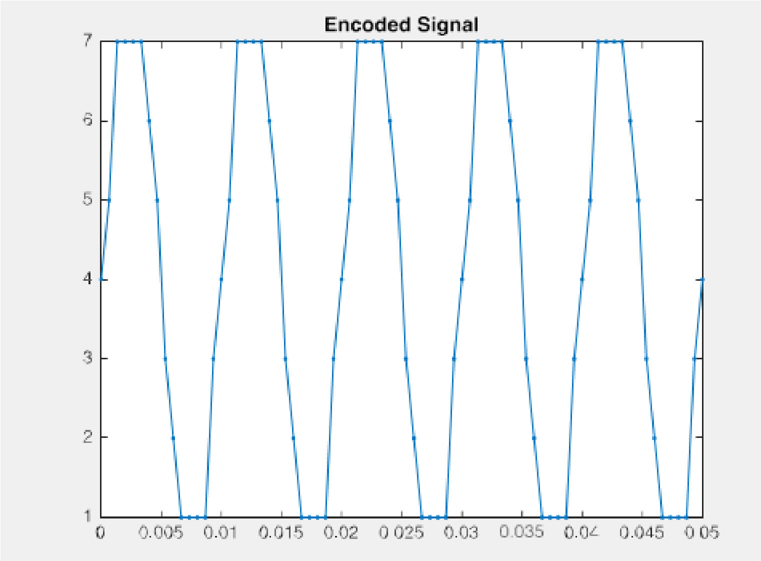
MATLAB CODE

```
% Generate a sinusoidal waveform with a DC offset so that it takes only positive amplitude value
clear;% clearing the variables
close all;% closing any opened figures
% Plotting the offset sinusoidal signal time = 0:.0005:.05;
freq_msg=100; %wave form frequency dc_ofst=2; % signal offset
signal=sin(2*pi*freq_msg*time)+dc_ofst; %Generating the signal
% plotting the signal figure;plot(time,signal)
xlabel('time') ylabel('Amplitude') title('Signal')
```

```
% Sampling the signal freq_sample=15*freq_msg; % sampling
frequency
samp_time=0:1/freq_sample:0.05; % sampling time
samp_signal=dc_ofst+sin(2*pi*freq_msg*samp_time);% generating the sampled signal
hold on
plot(samp_time,samp_signal,'rx') % plotting the sampled signal title('Sampled Signal')
legend('Original signal','Sampled signal');



% Uniform Quantizer
L=8; %No of Quantization levels
smin=round(min(signal));
smax=round(max(signal));
Quant_levl=linspace(smin,smax,L); % Length 8, to represent 9 intervals
codebook = linspace(0,smax,L+1); % Length 9, one entry for each interval
[index,quants] = quantiz(samp_signal,Quant_levl,codebook); % Quantize.

figure;plot(samp_time,samp_signal,'x',samp_time,quants,'.-')% plotting sampled signal and
quantization level title('Quantized Signal')
legend('Original signal','Quantized signal'); figure;plot(samp_time,index,'.-')%
plotting quantization levels of input signal
title('Encoded Signal')

% Binary coding
 for i=1:length(index) bincode_sig{i}=dec2bin(round(index(i)),7);
end
disp('binary encoded signal') disp(bincode_sig')
% SNR ratio calculation
noise=quants-samp_signal; % calculating noise
figure;plot(samp_time,noise,'.-') % plotting figure title('Noise')
r=snr(index,noise);% SNR
snr1=['SNR:',num2str(r)]; disp(snr1)
```

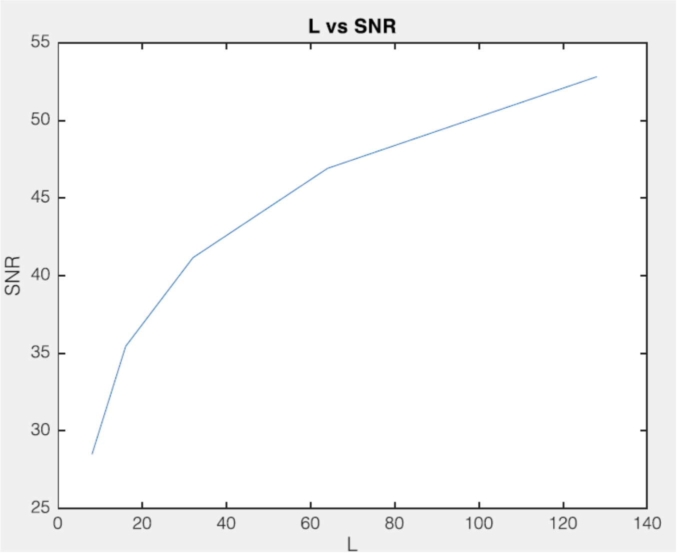

Sampled Signal
Original signal
Sampled signal
Amplitude
time
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05


Quantized Signal
Original signal
Quantized signal
3
2.5
2
1.5
1
0.5
0
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05



```
% function for ploting Quant_level vs SNR function [ r ] =
IMPL_Quant( l )
        % Plotting the offset sinusoidal signal time = 0:.0005:.05;
        freq_msg=100; %wave form frequency dc_ofst=2; %
        signal offset
      signal=sin(2*pi*freq_msg*time)+dc_ofst; %Generating the signal

        % Sampling the signal freq_sample=15*freq_msg; % sampling
        frequency
        samp_time=0:1/freq_sample:0.05; % sampling time
        samp_signal=dc_ofst+sin(2*pi*freq_msg*samp_time);%
generating the sampled signal

        % Uniform Quantizer
        L=l; %No of Quantization levels smin=round(min(signal));
        smax=round(max(signal));
        Quant_levl=linspace(smin,smax,L); % Length 8, to represent
9 intervals
      codebook = linspace(0.7,smax,L+1); % Length 9, one entry for each interval
        [index,quants] = quantiz(samp_signal,Quant_levl,codebook);
% Quantize.

        % Binary coding
        for i=1:length(quants) bincode_sig{i}=dec2bin(round(quants(i)),3);
        end

        % SNR ratio calculation
        noise=quants-samp_signal; % calculating noise r=snr(index,noise);% SNR

end


l=[8,16,32,64,128];% defining different levels for i=1:length(l)
        r(i) = IMPL_Quant(l(i));% calling the function
```

```
end
```



4.PULSE SHAPING AND MATCHED FILTERING

AIM:

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

THEORY:

Pulse Shaping

In communications, digital signals need to be mapped to an analog waveform in order to be transmitted over the channel. The mapping process is accomplished in two steps: (i) Mapping from source bits to complex symbols (also known as constellation points) (ii) Mapping from complex symbols to analog pulse trains(which is used here).

In our pulse shaping scheme, we introduce a set of complex-valued symbols as the input. These symbols are mapped from a bit stream via digital modulation. The bit stream may represent any data format (e.g., text, image, voice, video, etc). These complex-valued symbols are passed through the pulse shaping filter. A representation of this process is shown in Fig.1 below.

INPUT
PULSE SHAPING (FILTERING)
OUTPUT

Fig.1 A representation of the pulse shaping process.

Pulse shaping filter must be chosen carefully not to introduce inter-symbol interference. Here we will be using raised cosine filter for pulse shaping.

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). The frequency responses of raised-cosine pulses with different roll-off factors are shown in Fig. 2 below:

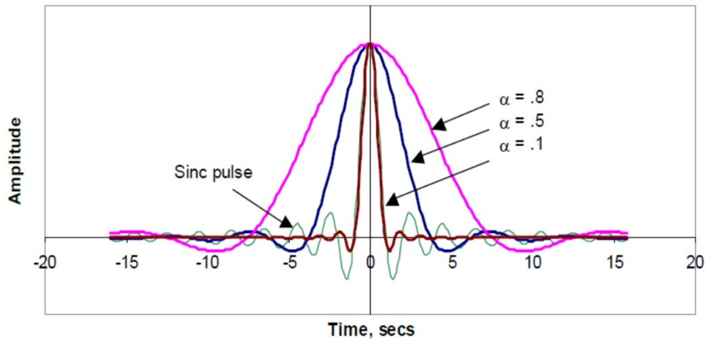


Fig. 2 Frequency response of a raised-cosine filter with different roll-off factor

Root raised-cosine pulse (RRC): The total effective filter of the transmission system is the combination of transmit and receive filter $g_{TX} * g_{RX}$, where $*$ is convolution. This effective filter (and not the individual filters) must fulfil the Nyquist criterion. We can achieve this goal if both filters have a transfer function that is equal to the square root of that of the raised cosine filter. Such a filter is therefore called a root raised cosine (RRC). The combination of both RRC filters then becomes a raised cosine and thus fulfils the Nyquist criterion. Furthermore, since the filters are real-valued and symmetric, the RRC is its own matched filter. The impulse response of the RRC filter is given as

$$g_{RRC}(t) = \frac{\frac{4\alpha}{\pi}\cos\left(\pi(1+\alpha)\frac{t}{T}\right)+(1-\alpha)\sin\left(\pi(1-\alpha)\frac{t}{T}\right)}{1-\left(4\alpha\frac{t}{T}\right)^2}$$

Matched Filtering

The receiver's RF front-end receives analog pulse trains. The information bits need to be recovered from these pulse trains. This is accomplished in two steps: (i) Mapping from analog pulse trains to constellation points. (ii) Mapping from complex symbols to bits, which is Detection.

The received analog signals are matched filtered to create the output complex waveform. Then the samples are detected. A representation of this process is shown in Fig.3 below.

INPUT COMPLEX WAVEFORM
MATCHED FILTERING
OUTPUT COMPLEX WAVEFORM

Fig. 3 – A representation of the matched filtering process

Matched filter design: A matched filter maximizes the signal to noise ratio (SNR) at its output. 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 (as explained earlier).

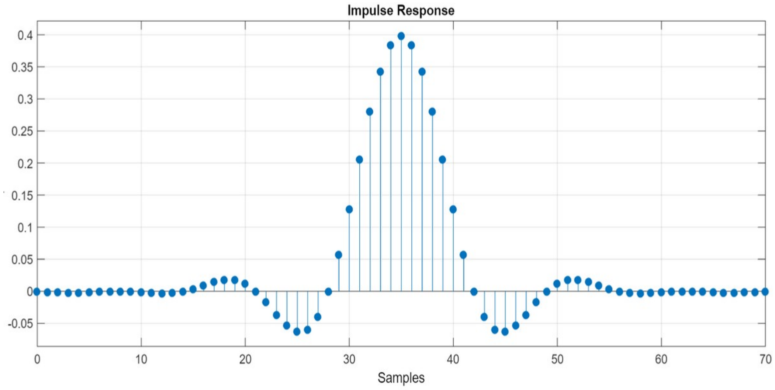
ALGORITHM:

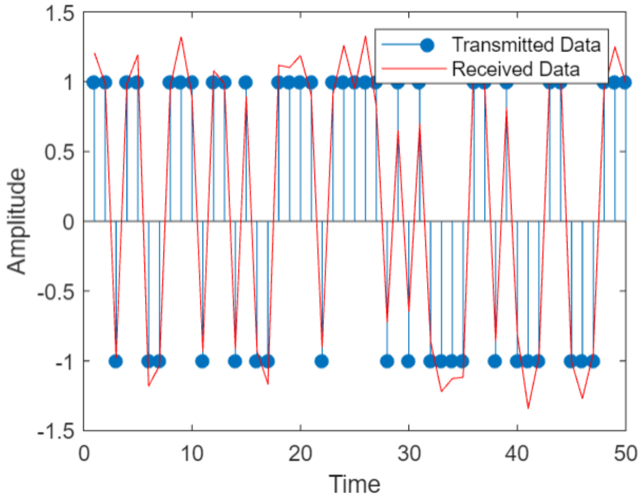
1. Generate a string of message bits.
2. Design square root raised cosine filter with roll off factor 0.4
3. Upsample and filter the data for pulse shaping.
4. Convert the ratio of energy per bit to noise power spectral density (EbNo) to an SNR value for use by the awgn function.
5. Filter the signal through an AWGN channel.
6. Downsample and pass the received signal through matched filter.
7. Plot impulse response of square root raised cosine filter, transmitted data, received data and filtered signal through AWGN channel.

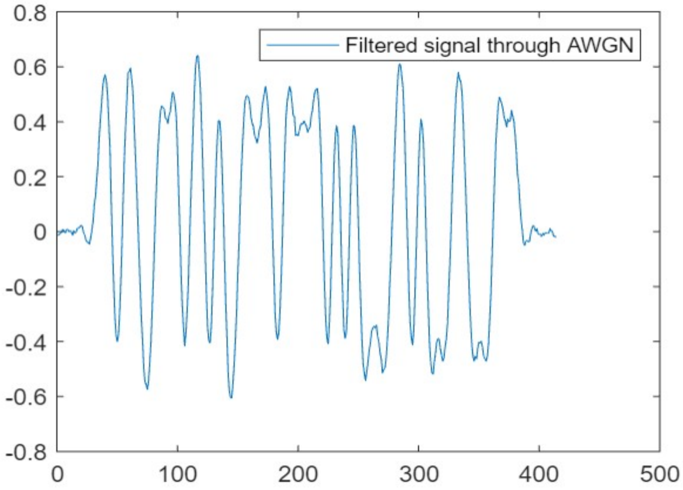
MATLAB CODE

```
% Program for pass a signal through a square-root, raised cosine filter.
% Pulse Shaping and Matched Filtering
close all;
clear;
rolloff = 0.4;     % Rolloff factor
```

```
span = 10;        % Filter span in symbols
sps = 7;          % Samples per symbol
M = 16;           % Modulation order
k = log2(M);      % Bits per symbol

% Generate the square-root, raised cosine filter coefficients.
rctFilt = rcosdesign(rolloff, span, sps,'normal');
fvtool(rctFilt,'Analysis','impulse')

% Create a vector of bipolar data.
BP_Data = 2*randi([0 1], 50, 1) - 1;

% Upsample and filter the data for pulse shaping.
UP_s = upfirdn(BP_Data, rctFilt, sps,1);

% Using the number of bits per symbol (k)
% and the number of samples per symbol (sps),
% convert the ratio of energy per bit to noise power spectral density (EbNo)
% to an SNR value for use by the awgn function.
EbNo = 100;
snr = EbNo + 10*log10(k) - 10*log10(sps);
filtlen = 10;     % Filter length in symbols

rxSignal = awgn(UP_s,snr,'measured');% filtering the signal through an AWGN channel.

% Add noise.
rxSignal = rxSignal + randn(size(rxSignal))*.01;

rxFiltSignal = upfirdn(rxSignal,rctFilt,1,sps);      % Downsample and filter
rxFiltSignal = rxFiltSignal(filtlen + 1:end - filtlen); % Account for delay

% Plotting the function
figure;
stem(BP_Data,'filled')
hold on
plot(rxFiltSignal,'r')
xlabel('Time'); ylabel('Amplitude');
legend('Transmitted Data','Received Data')
figure;
plot(rxSignal)
legend('Filtered signal through AWGN')
```


Impulse Response
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
0
10
20
30
40
50
60
70
Samples


Transmitted Data
Received Data
1.5
1
0.5
0
-0.5
-1
-1.5
Amplitude
0
10
20
30
40
50
Time


Filtered signal through AWGN
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
100
200
300
400
500

5.EYE DIAGRAM

AIM:

1. Generate a string of message bits.
2. Use rasied cosine pulse p(t) as the shapig pulse, and generate the corresponding baseband signal with a fixed bit duration T_b. You may use roll-off factor as α = 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:

In telecommunication, an **eye pattern**, also known as an **eye diagram**, is an oscilloscope display in which a digital signal from a receiver is repetitively sampled and applied to the vertical input, while the data rate is used to trigger the horizontal sweep. It is so called because, for several types of coding, the pattern looks like a series of eyes between a pair of rails. It is a tool for the evaluation of the combined effects of channel noise, dispersion and inter symbol interference on the performance of a baseband pulse-transmission system.

ALGORITHM:

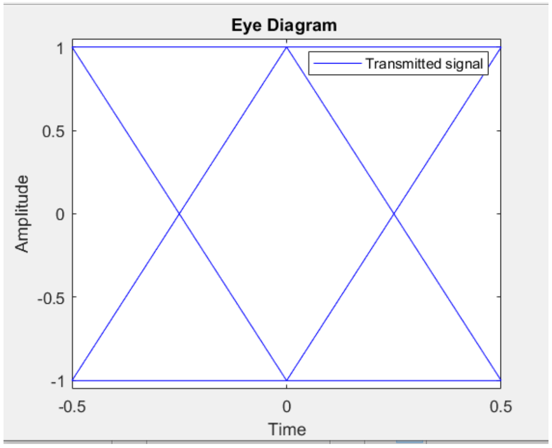
1. Generate a string of message bits.
2. Design square root raised cosine filter with roll off factor 0.4
3. Upsample and filter the data for pulse shaping.
4. Convert the ratio of energy per bit to noise power spectral density (EbNo) to an SNR value for use by the awgn function.
5. Filter the signal through an AWGN channel.
6. Downsample and pass the received signal through matched filter.

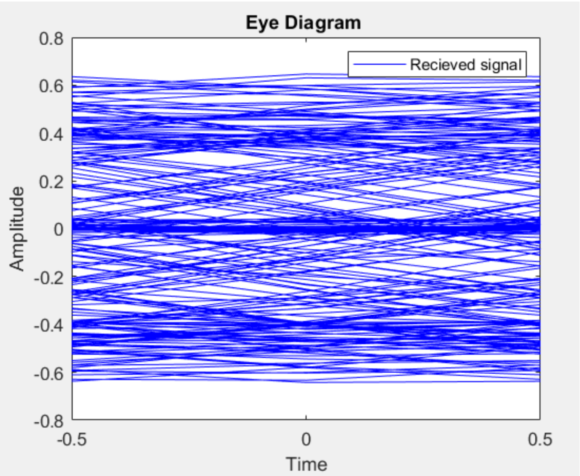
7. Plot eye diagram for transmitted signal, received signal and filtered signal.

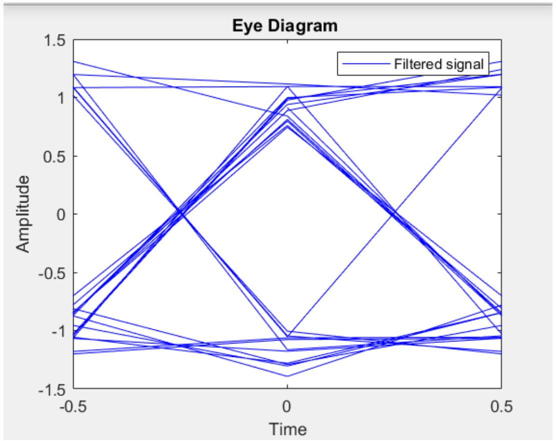
MATLAB CODE

```
% Program for pass a signal through a square-root, raised cosine filter.
% Pulse Shaping and Matched Filtering
close all;
clear;
rolloff = 0.4;     % Rolloff factor
span = 10;         % Filter span in symbols
sps = 7;          % Samples per symbol
M = 16;           % Modulation order
k = log2(M);       % Bits per symbol

% Generate the square-root, raised cosine filter coefficients.
rctFilt = rcosdesign(rolloff, span, sps,'normal');
fvtool(rctFilt,'Analysis','impulse')

% Create a vector of bipolar data.
BP_Data = 2*randi([0 1], 50, 1) - 1;

% Upsample and filter the data for pulse shaping.
UP_s = upfirdn(BP_Data, rctFilt, sps,1)

% Using the number of bits per symbol (k)
% and the number of samples per symbol (sps),
% convert the ratio of energy per bit to noise power spectral density (EbNo)
% to an SNR value for use by the awgn function.
EbNo = 100;
snr = EbNo + 10*log10(k) - 10*log10(sps);
filtlen = 10;      % Filter length in symbols

rxSignal = awgn(UP_s,snr,'measured');% filtering the signal through an AWGN channel.

% Add noise.
rxSignal = rxSignal + randn(size(rxSignal))*.01;

rxFiltSignal = upfirdn(rxSignal,rctFilt,1,sps);       % Downsample and filter
rxFiltSignal = rxFiltSignal(filtlen + 1:end - filtlen); % Account for delay

% Plotting the function
figure;
stem(BP_Data,'filled')
hold on
plot(rxFiltSignal,'r')
xlabel('Time'); ylabel('Amplitude');
```

```
legend('Transmitted Data','Received Data')
figure;
plot(rxSignal)
legend('Filtered signal through AWGN')

eyediagram(BP_Data,2);legend('Transmitted signal')
eyediagram(rxSignal,2);legend('Recieved signal')
eyediagram(rxFiltSignal,2);legend('Filtered signal')
```







PART C

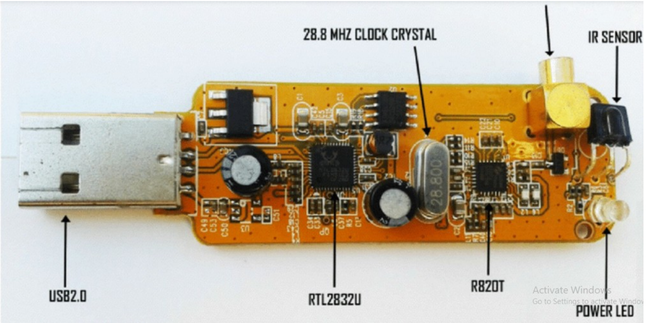
FAMILIARIZATION WITH SOFTWARE DEFINED RADIO (HARDWARE AND CONTROL SOFTWARE)

AIM:

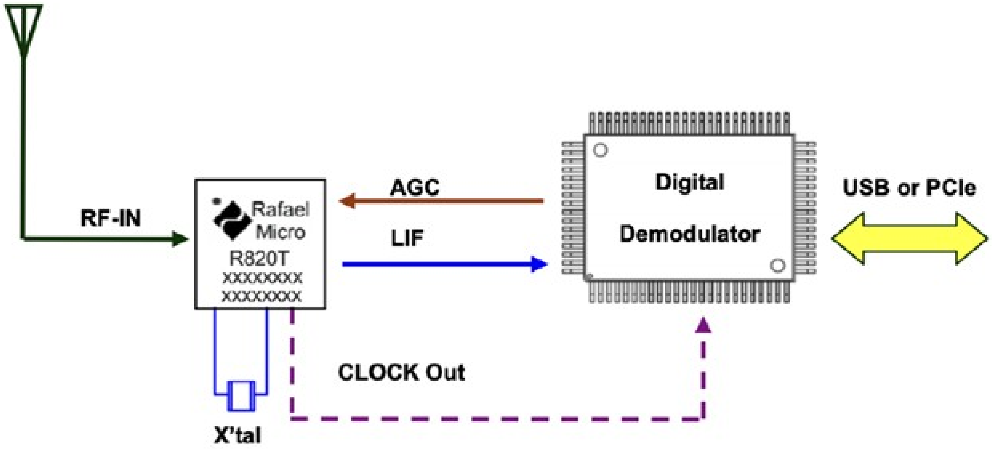
1.To Familiarize with an SDR hardware for reception and transmission of RFsignal.
2.To Familiarize how it can be interfaced with computer.
3.To Familiarize with GNU Radio (or similar software's like Simulink/ Lab-View) that can be used to process the signals received through the SDRhardware.
4.To Familiarize available blocks in GNU Radio. Study how signals can begenerated and spectrum (or power spectral density) of signals can beanalyzed. Study how filtering can be performed.

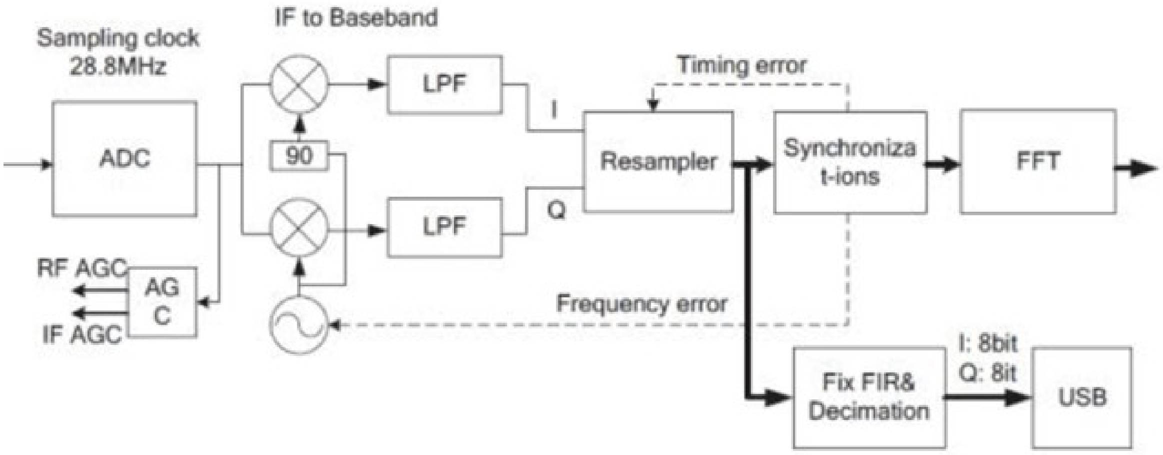
RTL SDR DONGLE CONNECTION DIAGRAM

Inside RTL SDR Dongle



Block Diagram of SDR Dongle





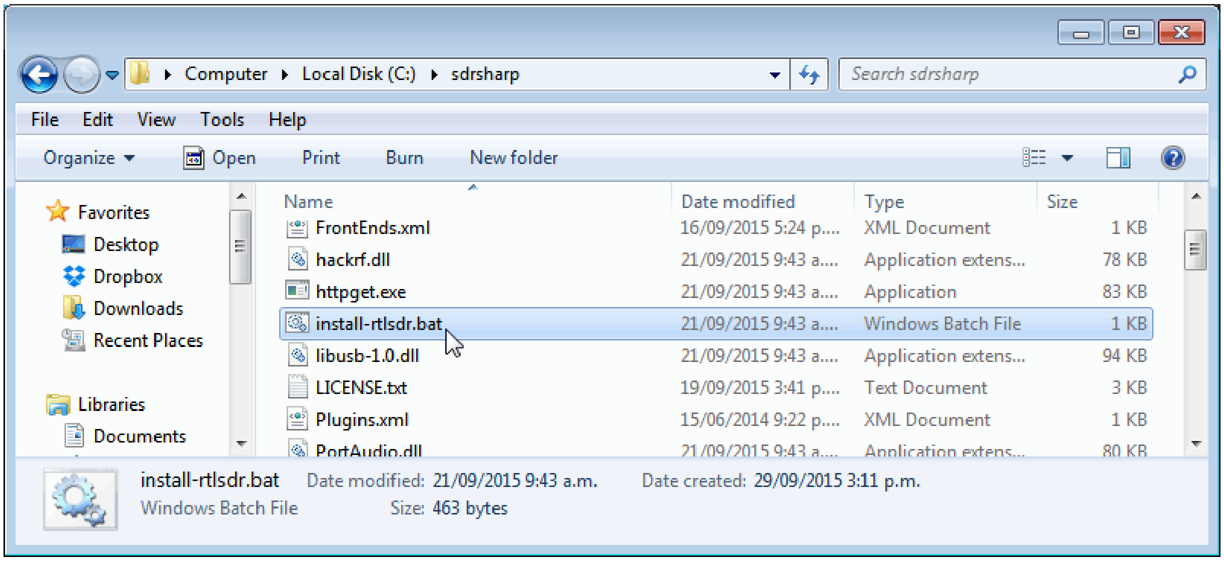
FM RECEPTION

Aim:

To receive digitized FM signal (for the clearest channel in the lab) using the SDR board.

SDR# (SDRSharp) Set Up

1. Go to www.airspy.com and click on the download button to download sdrsharp-x86.zip
2. Extract (unzip) sdrsharp-x86.zip to a folder
3. Double click on install-rtlsdr.bat from within the extracted folder. This will start a command prompt that will download all the drivers required to make SDRSharp work with RTL-SDR. The command prompt will automatically close after a few seconds when it is done. It the bat file ran successfully the files rtlsdr.dll and zadig.exe will be downloaded into the SDR# directory.

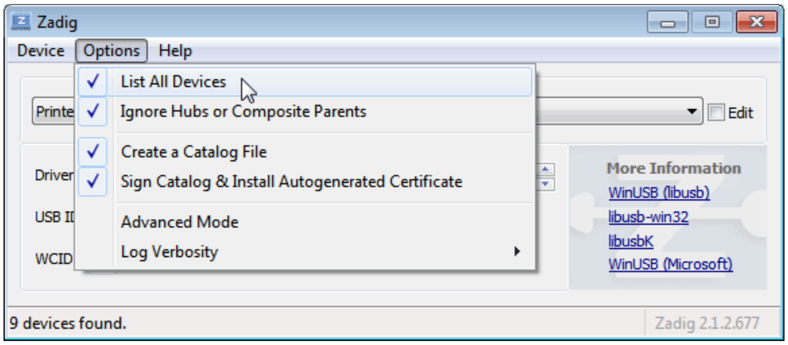

Computer ▸ Local Disk (C:) ▸ sdrsharp
Search sdrsharp
File Edit View Tools Help
Organize ▾ Open Print Burn New folder
Favorites
Desktop
Dropbox
Downloads
Recent Places
Libraries
Documents

Name	Date modified	Type	Size
FrontEnds.xml	16/09/2015 5:24 p....	XML Document	1 KB
hackrf.dll	21/09/2015 9:43 a....	Application extens...	78 KB
httpget.exe	21/09/2015 9:43 a....	Application	83 KB
install-rtlsdr.bat	21/09/2015 9:43 a....	Windows Batch File	1 KB
libusb-1.0.dll	21/09/2015 9:43 a....	Application extens...	94 KB
LICENSE.txt	19/09/2015 3:41 p....	Text Document	3 KB
Plugins.xml	15/06/2014 9:22 p....	XML Document	1 KB
PortAudio.dll	21/09/2015 9:43 a....	Application extens...	80 KB

install-rtlsdr.bat Date modified: 21/09/2015 9:43 a.m. Date created: 29/09/2015 3:11 p.m.
Windows Batch File Size: 463 bytes

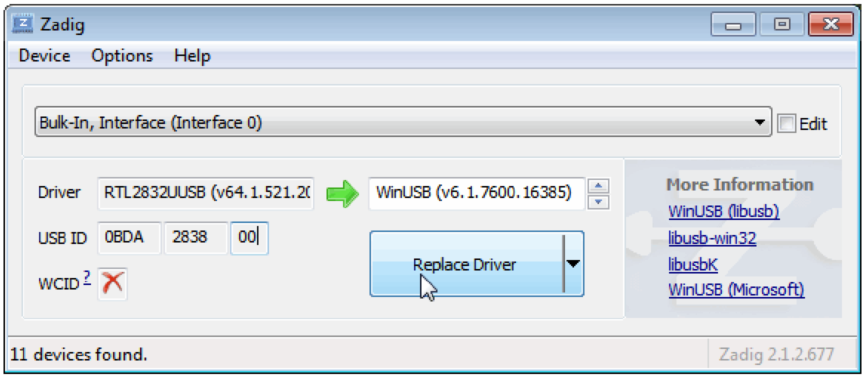
4. **Plug in SDR dongle**. In the folder where you extracted the sdrsharp files find the file called **zadig.exe**. Right click this file and select "Run as administrator".

5. In Zadig, go to "**Options->List All Devices**" and make sure this option is checked.



6. Select "**Bulk-In, Interface (Interface 0)**" from the drop down list.

7. We need to install the WinUSB driver, so also **ensure that WinUSB is selected** in the box after the arrow next to where it says Driver (this is the default selection). The box to the left of the green arrow is not important, and it may show (NONE) or (RTL...). This left hand box indicates the currently installed driver, and the box to the right the driver that will be installed after clicking Replace/Install Driver.



Bulk-In, Interface (Interface 0) ☐ Edit

Driver RTL2832UUSB (v64.1.521.2(→ WinUSB (v6.1.7600.16385)

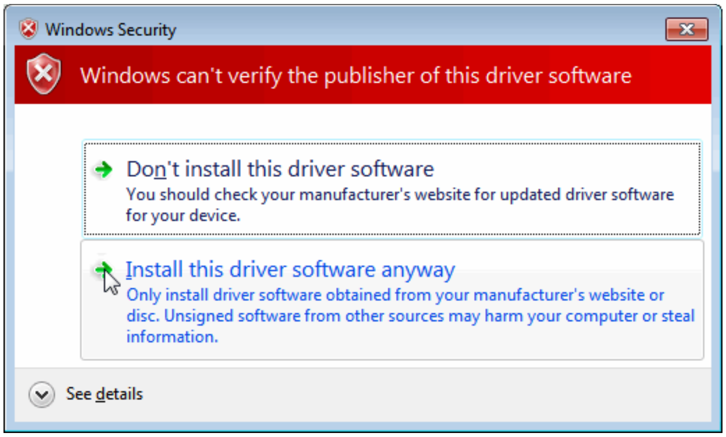
USB ID 0BDA 2838 00

WCID ? ✗

11 devices found.

Zadig 2.1.2.677

8. Click Replace Driver. On some PC's you might get a warning that the publisher cannot be verified, but just accept it by clicking on "Install this driver software anyway". This will install the drivers necessary to run the dongle as a software defined radio.

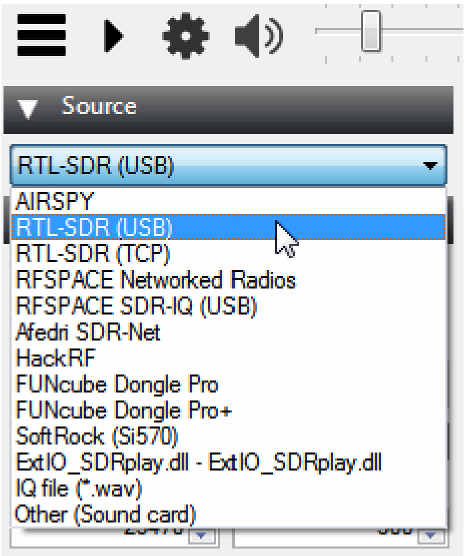


Windows can't verify the publisher of this driver software

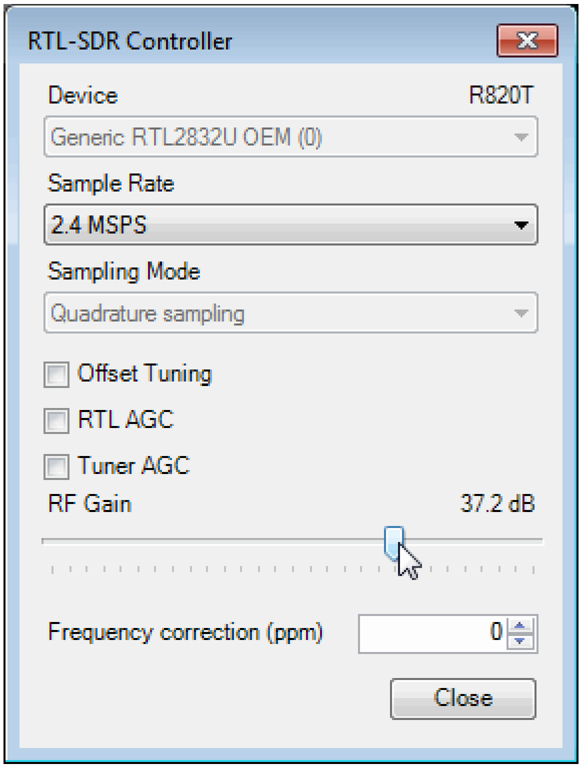
→ Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.

→ Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

9. Open SDRSharp.exe and set the "Source" drop down box to 'RTL-SDR USB'. This "Source" tab is on the lower left menu bar by default.
10. Press the Play button (the right facing triangle in the top left of the program).RTL-SDR software radio should now be set up and ready to use.



11. Also **adjust the RF gain settings** by pressing the Configure button up the top next to the Play button. By default the RF gain is set at zero. A gain of zero will probably receive nothing but very strong broadcast FM - increase the gain until you start seeing other signals.


RTL-SDR Controller

Device R820T

Generic RTL2832U OEM (0)

Sample Rate

2.4 MSPS

Sampling Mode

Quadrature sampling

□ Offset Tuning

□ RTL AGC

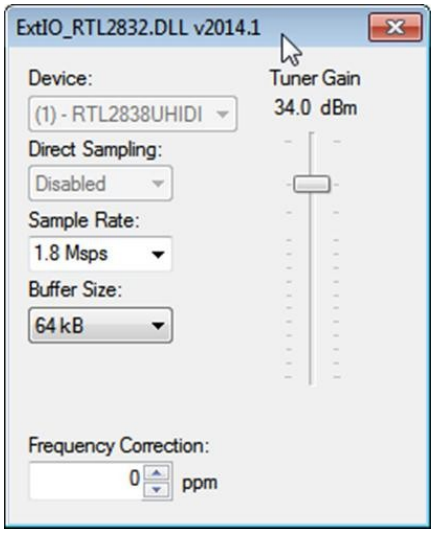
□ Tuner AGC

RF Gain 37.2 dB

Frequency correction (ppm) 0

12. Press Start or alternatively press F2. This will start the SDR.

13. To set the RTL-SDR sample rate, gain and frequency correction click on the ExtIO button .


ExtIO_RTL2832.DLL v2014.1

Device: (1) - RTL2838UHIDI

Direct Sampling: Disabled

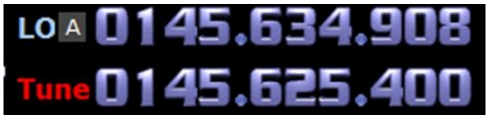
Sample Rate: 1.8 Msps

Buffer Size: 64 kB

Tuner Gain 34.0 dBm

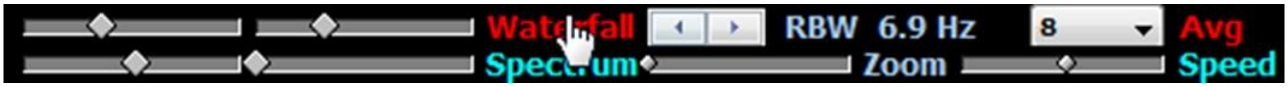
Frequency Correction: 0 ppm

14. To tune to a station, change the Local Oscillator frequency to a frequency near the frequency you are interested in. Then tune to the desired frequency either by clicking in the RF spectrum, or using the Tune numbers.

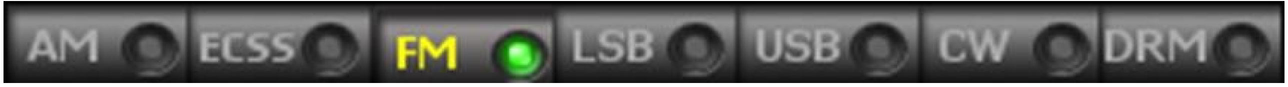

LO A 0145.634.908

Tune 0145.625.400

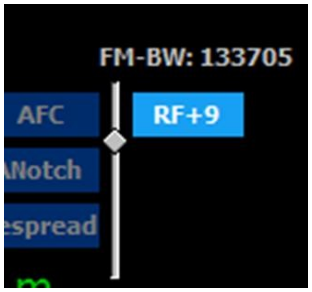
15. You can zoom in and out of the spectrum by using the Zoom slider which is to the left of the word zoom.


RBW 6.9 Hz

16. The mode can be altered by clicking on the mode buttons.

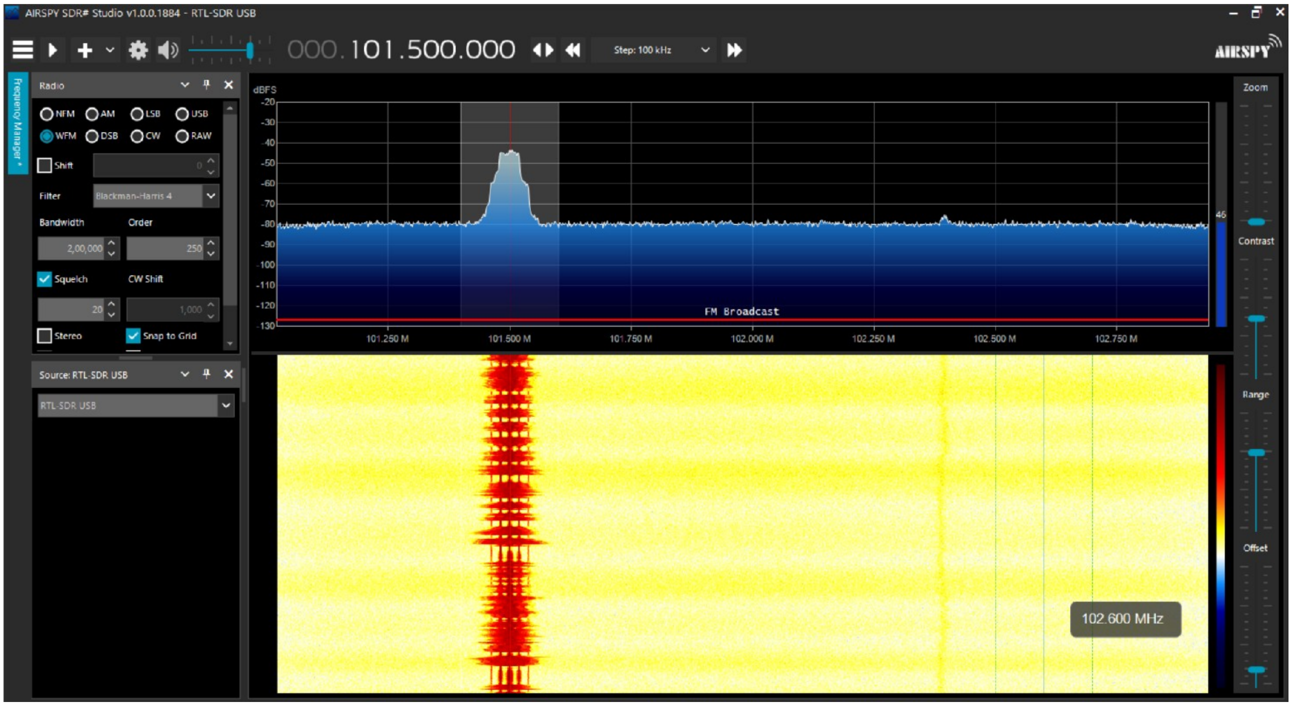


17. After clicking on the FM mode button, the FM bandwidth can be manually modified with the FM-BW slider.


FM-BW: 133705

18. To listen to a typical wideband broadcast FM station, you will need to change the audio sampling rate to 192000 Hz. Do this by clicking on the Bandwidth button or alternatively by pressing F6 and then selecting the output sampling rate as 192000 Hz.

Receiving All India Radio AIR Kannur 101.5 FM


AIRSPY SDR# Studio v1.0.0.1884 - RTL-SDR USB

000.101.500.000

Step: 100 kHz

Radio

○ NFM ○ AM ○ LSB ○ USB

● WFM ○ DSB ○ CW ○ RAW

□ Shift 0

Filter: Blackman-Harris 4

Bandwidth: 2,00,000 Order: 250

☑ Squelch: 20 CW Shift: 1,000

□ Stereo ☑ Snap to Grid

Source: RTL-SDR USB

RTL-SDR USB

Familiarize with GNU Radio

Aim:

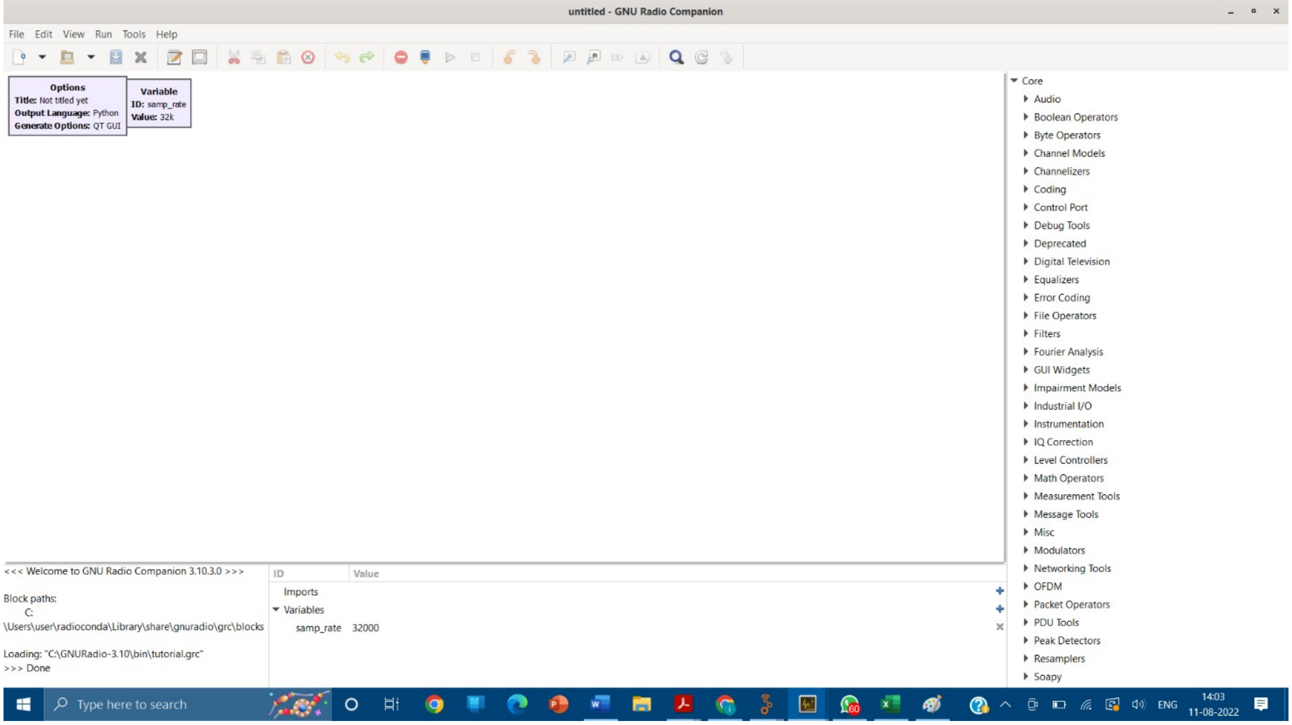
To familiarize available blocks in GNU Radio. Study how signals can be generated and spectrum (or power spectral density) of signals can be analysed. Study how filtering can be performed.

GNU Radio

GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It 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.

1. Open a terminal window using: Applications > Accessories > Terminal. At the prompt type: gnuradio-companion.

An untitled GRC window similar to the one below should open.



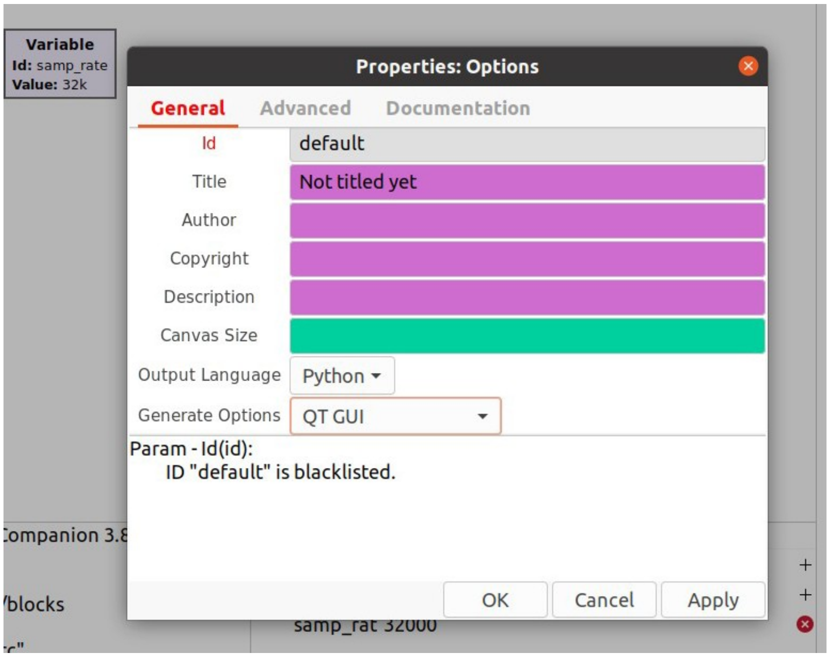
Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

```
<<< Welcome to GNU Radio Companion 3.10.3.0 >>>

Block paths:
	C:
\Users\user\radioconda\Library\share\gnuradio\grc\blocks

Loading: "C:\GNURadio-3.10\bin\tutorial.grc"
>>> Done
```

ID	Value
Imports	
Variables	
samp_rate	32000

Double click on the Options block. This block sets some general parameters for the flow graph. Type in a project title (such as tutorial) and author. Set Generate Options to QT GUI, Output language to Python. Then close the properties window. The other block that is present is the Variable block. It is used to set the sample rate.



Id: default

Title: Not titled yet

Author:

Copyright:

Description:

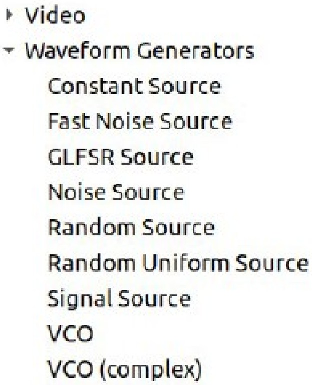
Canvas Size:

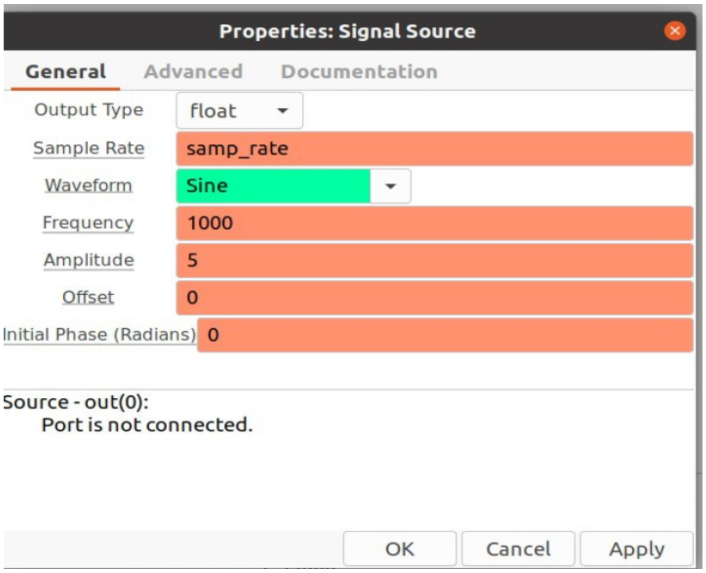
Output Language: Python

Generate Options: QT GUI

Param - Id(id):
ID "default" is blacklisted.

2. 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.
3. Open the Waveform generators category and double click on the Signal Source. Note that 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 to match those as shown in the figure below and close the window. This Signal Source is now set to output a real valued 1KHzsinusoid with an amplitude of .5. Change the output type to float.

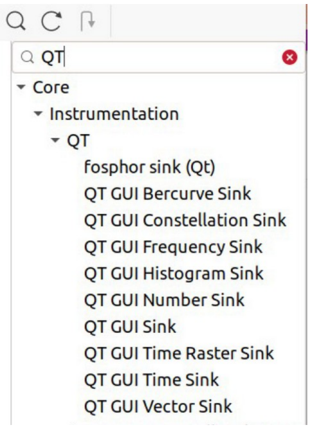




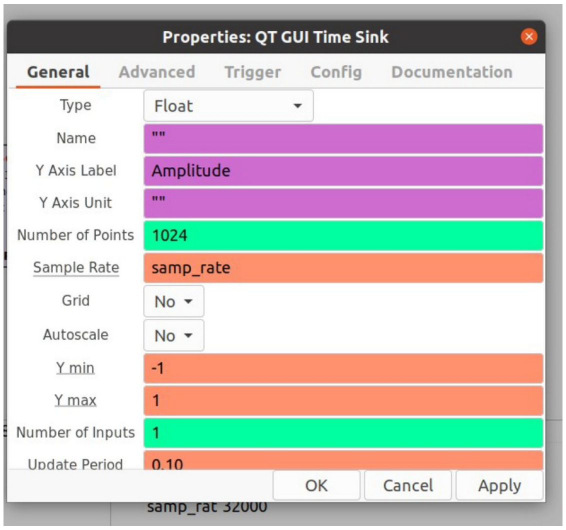
Output Type: float
Sample Rate: samp_rate
Waveform: Sine
Frequency: 1000
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

Source - out(0):
Port is not connected.

4. In order to view this wave we need one of the graphical sinks. Choose Core→Instrumentation→ QT →QT GUI Time sink and then double click on it

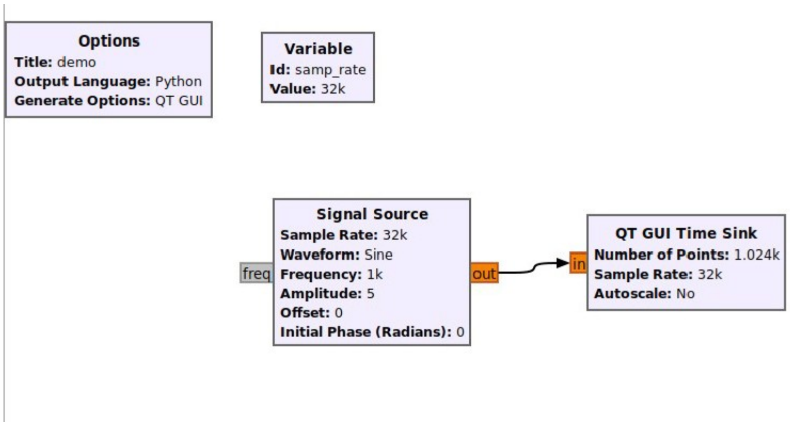


5. Change the type to float. This sink will act as an oscilloscope. If you want multiple channel, increase the number of inputs to 2.



Type: Float
Name: ""
Y Axis Label: Amplitude
Y Axis Unit: ""
Number of Points: 1024
Sample Rate: samp_rate
Grid: No
Autoscale: No
Y min: -1
Y max: 1
Number of Inputs: 1
Update Period: 0.10

6. In order to connect these two blocks, click once on the "out" port of the Signal Source, and then once on the "in" port of the Scope Sink. The following flow graph should be displayed.



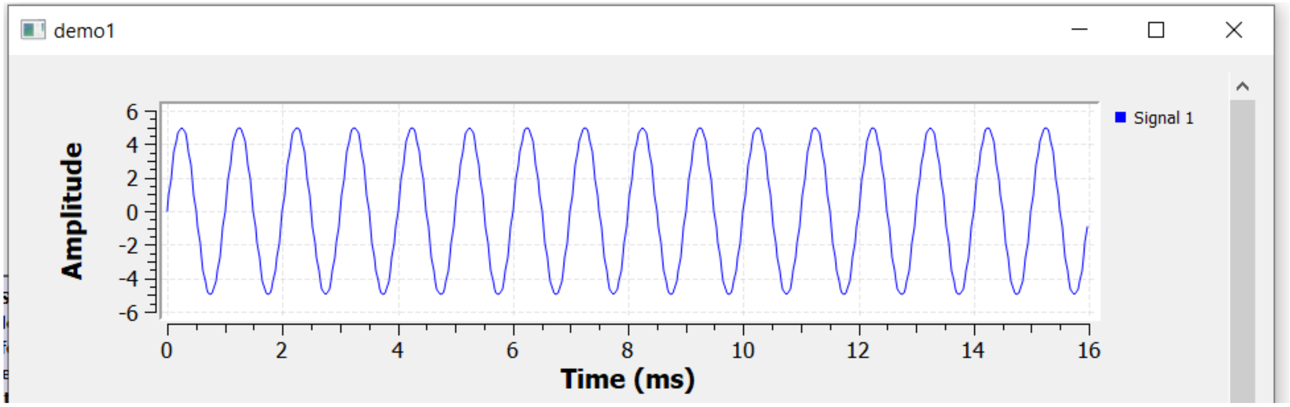
Options
Title: demo
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 32k

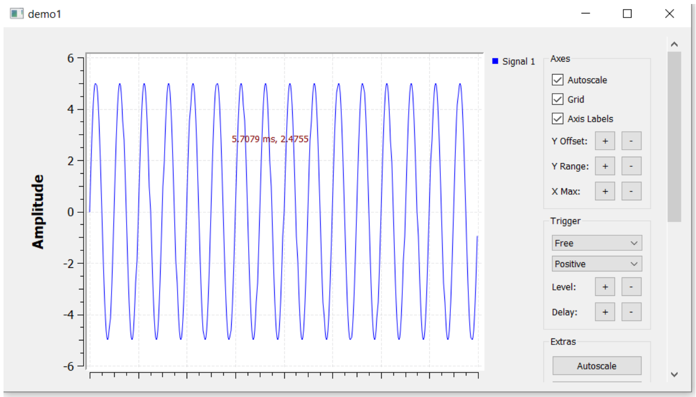
Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 1k
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

7. 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: tutorial1.grc and save. Click the "Execute the flow graph" icon. A scope plot should open displaying several cycles of the sinusoid.

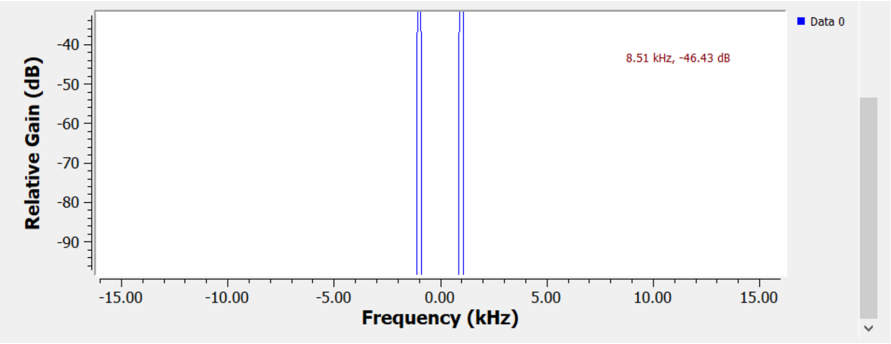


8. Go back to QT GUI sink properties and switch on control panel under config Re run the flow graph

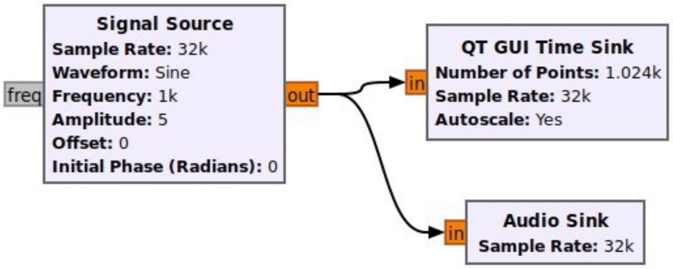


9. Add QT GUI frequency sink . Change the Type to Float and show control panel and leave the remaining parameters at their default values.

10. Connect this to the output of the Signal Source by clicking on the out port of the Signal Source and then the in port of the QT GUI Sink. Generate and execute the flow graph. You should observe the scope as before along with an FFT plot correctly showing the frequency of the input at 1KHz. Close the output windows.



11. Now bring in an Audio sink. Create the flow graph shown below. The Audio Sink is found in Core->Audio -> Audio Sinks category. Generate and execute the flow graph. Along with graphical display of the signal you can hear the 1KHz tone.

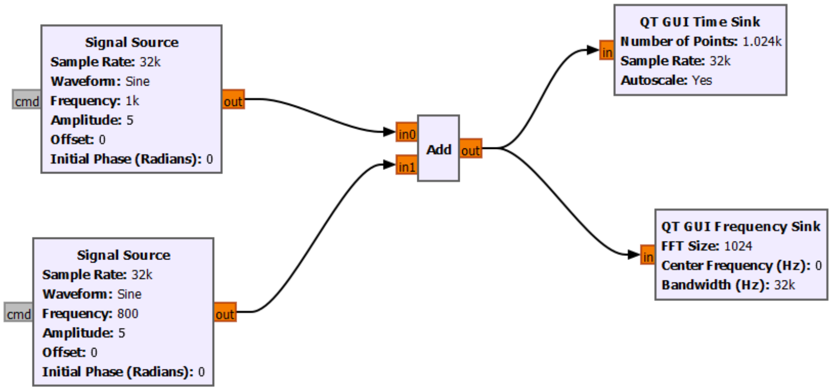


Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 1k
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: Yes

Audio Sink
Sample Rate: 32k

12. Construct the flow graph shown below. Set the sample rate to 32000. The two Signal Sources should have frequencies of 1000 and 800, respectively. The Add block is found in the Math Operators category.



Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 1k
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

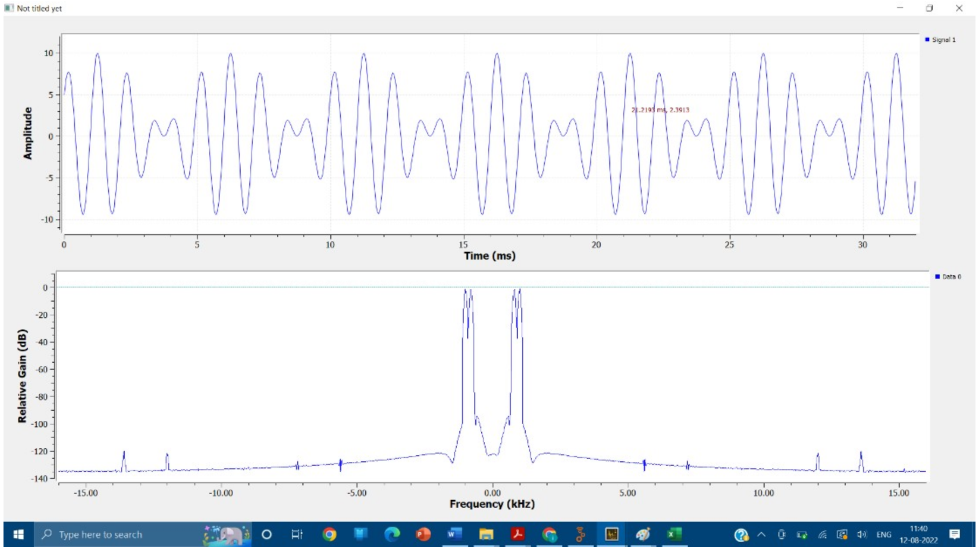
Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 800
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

Add

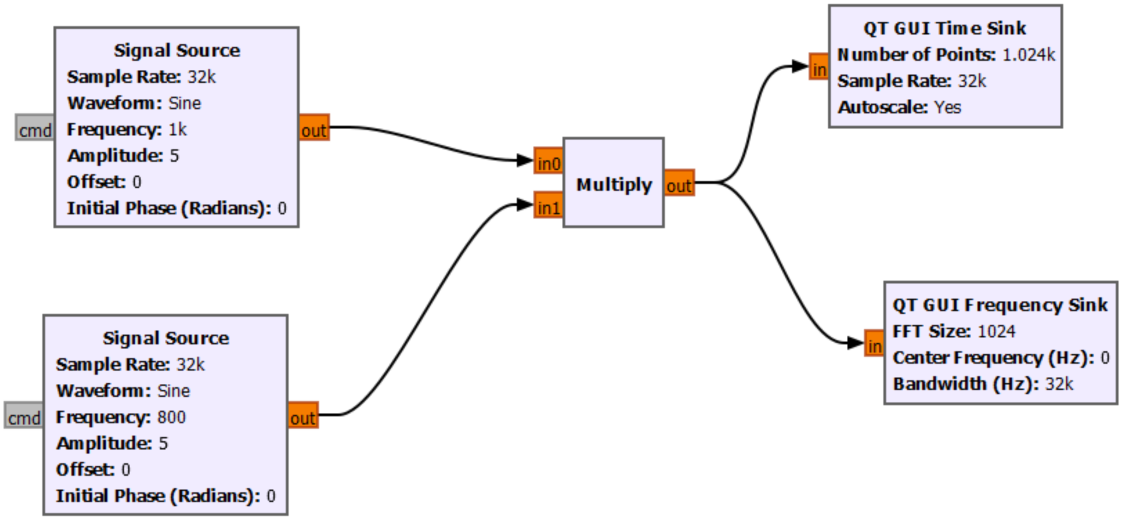
QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: Yes

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 32k

13. Generate and execute the flow graph. On the Scope plot you should observe a waveform corresponding to the sum of two sinusoids. On the FFT plot you should see components at both 800 and 1000 Hz.



14. Replace the Add block with a Multiply block.



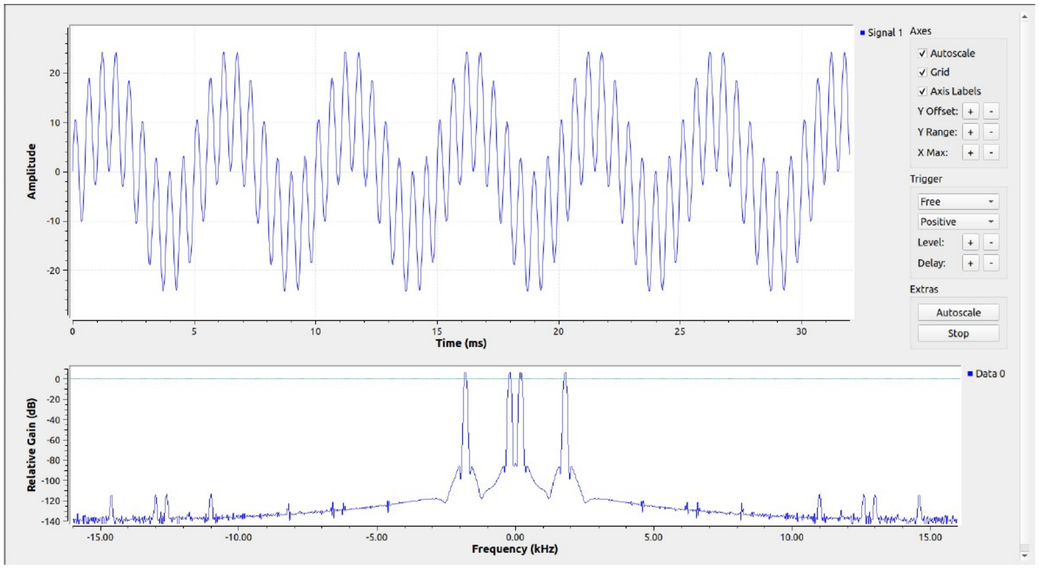
Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 1k
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 800
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

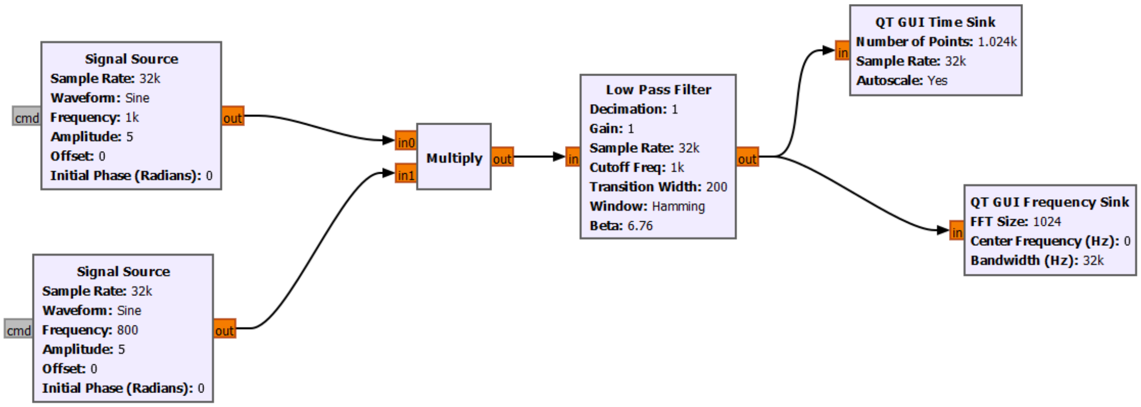
Multiply

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: Yes

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 32k



15. Modify the flow graph to include a Low Pass Filter block as shown in the figure below. This block is found in the Core→ Filters category. Multiply block outputs a 200Hz and a 1.8KHz sinusoid. We want to create a filter that will pass the 200Hz and block the 1.8KHz component. Set the low pass filter to have a cutoff frequency of 1KHz and a transition width of 200 Hz. Use a Rectangular Window. Generate and execute the flow graph. Only the 200Hz component passes through the filter.



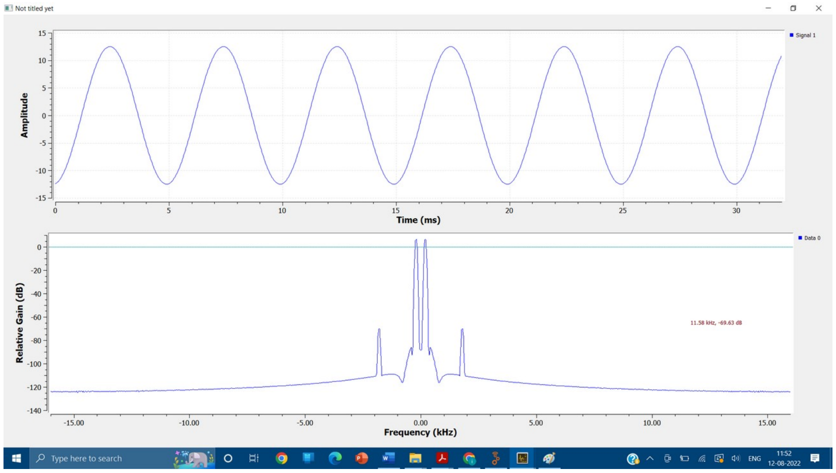
Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 1k
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

Signal Source
Sample Rate: 32k
Waveform: Sine
Frequency: 800
Amplitude: 5
Offset: 0
Initial Phase (Radians): 0

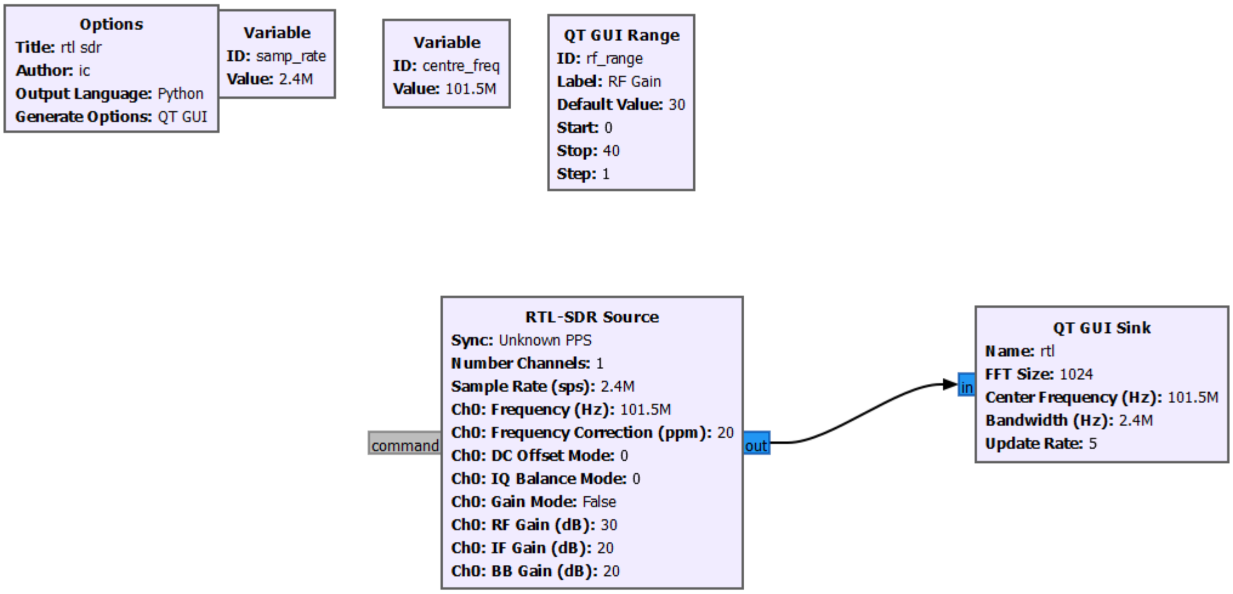
Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 32k
Cutoff Freq: 1k
Transition Width: 200
Window: Hamming
Beta: 6.76

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: Yes

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 32k



RTL-SDR for RF Signal Capture on GNU Radio



Options
Title: rtl sdr
Author: ic
Output Language: Python
Generate Options: QT GUI

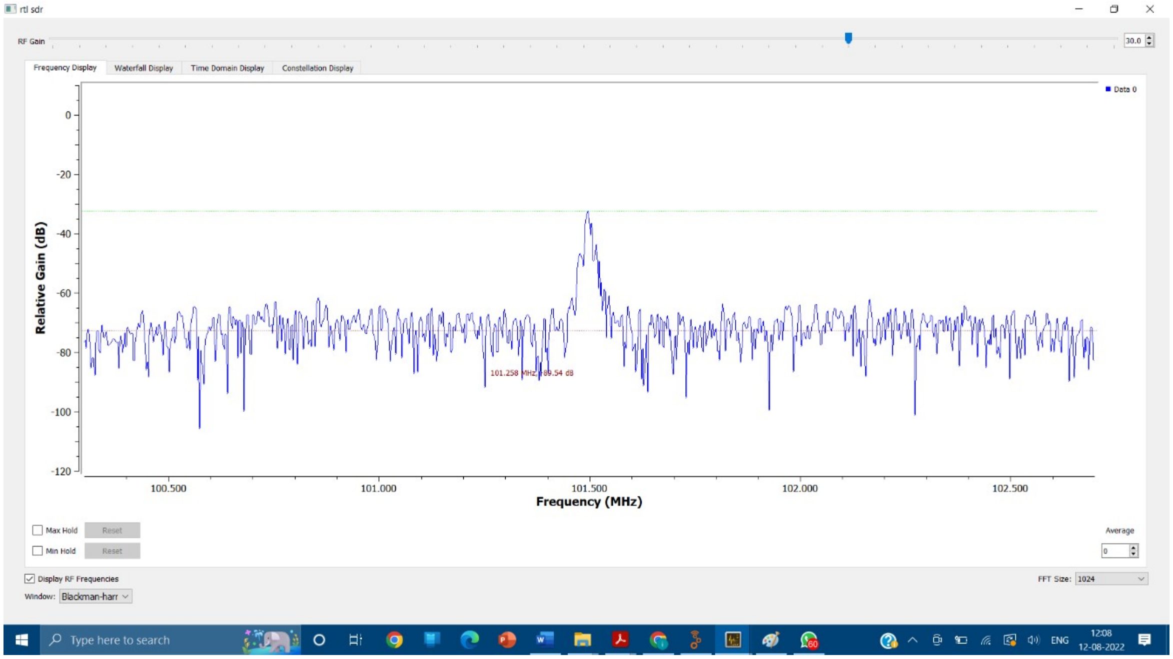
Variable
ID: samp_rate
Value: 2.4M

Variable
ID: centre_freq
Value: 101.5M

QT GUI Range
ID: rf_range
Label: RF Gain
Default Value: 30
Start: 0
Stop: 40
Step: 1

RTL-SDR Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 2.4M
Ch0: Frequency (Hz): 101.5M
Ch0: Frequency Correction (ppm): 20
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 30
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Sink
Name: rtl
FFT Size: 1024
Center Frequency (Hz): 101.5M
Bandwidth (Hz): 2.4M
Update Rate: 5



rtl sdr

RF Gain 30.0

Frequency Display | Waterfall Display | Time Domain Display | Constellation Display

□ Max Hold Reset

□ Min Hold Reset

Average 0

☑ Display RF Frequencies

Window: Blackman-harr

FFT Size: 1024