ECL332 Communication Lab

Department of Electronics & Communication Engg
Govt Engineering College, Barton Hill

Instructors: Birenjith Sasidharan, Sruthi B. R.

July 31, 2022

1. The lab has three parts: (1) simulation experiments (2) emulation of FM using software defined radio, and (3) design and setting up of prototype circuits realizing certain communication functionalities using ICs.
2. Logistics:
   - Timing: Tue, Fri 9am – 12 noon
   - Part 1 and 2 at DSP Lab; Part 3 at Communication Lab.
3. You must keep separate rough and fair record books. You must bring updated rough record when you come to lab.

Simulation Experiments

Instructions

1. We prefer to use MATLAB for two reasons: (1) students are exposed to python in ECL201 Scientific Computing Lab and ECL333 DSP Lab, and hence it is good to get exposed to another language; (2) many companies working in communication industry continue to use MATLAB. However, if any student is desperate to use python, she is permitted to do so.
2. Every student can login using their gecbh mail id at `https://matlab.mathworks.com/`, save and run their programs in the cloud. Use the naming convention of *e<n><abc>.m* where *<n>* is a placeholder for the experiment number and *<abc>* is a placeholder for description of the functionality (for eg., *e1pcm.m*). If you copy the file locally, please keep them inside a folder named with your roll number.
3. When you come to lab, you must have written in your rough record the aim, algorithm, expected output, and a rough sketch of the code (or at least a pseudocode) for the experiment of that day. The aim, algorithm and expected output are provided by the instructor, where as you have to write the code on your own. Unless you come prepared with a rough sketch of the code, it can be tough to finish the program in lab hours.
4. Please collaborate freely with your class mates and seek help from online resources while programming.

5. The class representative will share programs written by students (that are distinct to each other) with the instructor. (If two students collaborated and wrote the same program, you are not required to share two copies.) The instructor shall pick the best program, correct it if necessary, and pass it back. Every one shall write this program in the fair record.

Experiment 1: Encoding and Reconstruction of PCM

Aim To implement encoding and reconstruction algorithms for waveform coding using Pulse Code Modulation. To vary number of quantization levels L and plot signal-to-noise ratio (SNR) versus L.

Algorithm for Simulation

1. Generate samples of a raised sine wave of frequency $f = 2Hz$

$$x(t) = 255 \times \frac{1 + \sin(2\pi f t)}{2}$$

   with a sampling rate $f_s = 16$ samples per second. Note that the sampling rate is four times the Nyquist rate.

2. Quantize the samples using different uniform quantizers Q_L defined by the total numer of levels $L \in \{256, 128, 64, 32, 16\}$. Each of Q_L can be realized as given below:

   - $L = 256$: Round the sample to nearest integer.
   - $L = 128$: Approximate the sample to nearest even number.
   - $L = 64$: Approximate the sample to nearest multiple of 4.
   - $L = 32$: Approximate the sample to nearest multiple of 8.
   - $L = 16$: Approximate the sample to nearest multiple of 16.

   The resultant quantized singal is denoted by $XX(nT_s)$.

3. Encode each quantized sample using binary code with number of bits $R = \log_2(L)$. This results in the pulse-code modulated binary stream of the input.

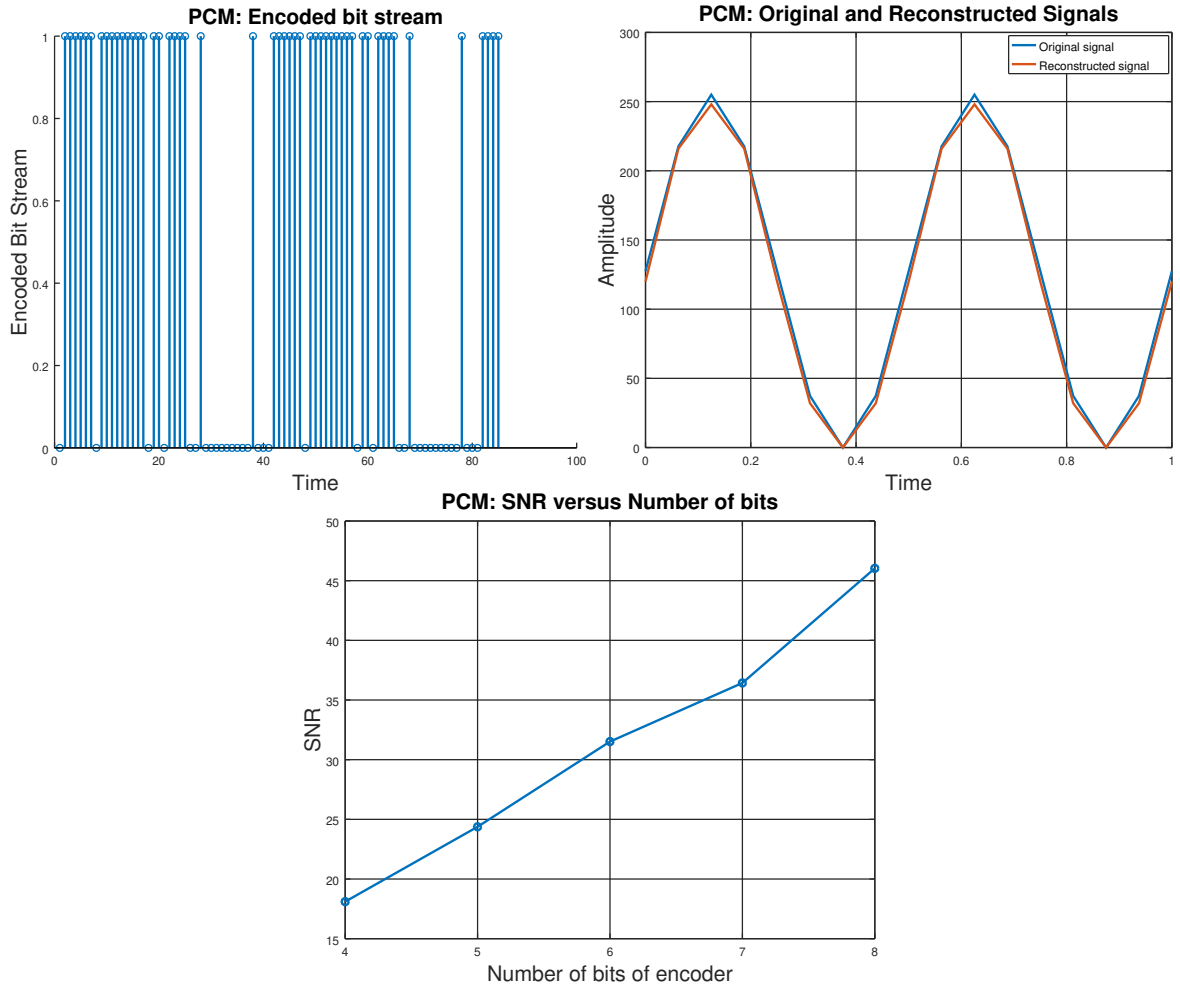
4. Given the binary string, it is possible to reconstruct back $XX(nT_s)$ by binary-to-decimal conversion follwed by appropriate scaling. Plot original signal $x(t)$ and reconstruted signal $XX(nT_s)$.

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

$$\begin{aligned} P_x &= \frac{1}{2} \times (255/2)^2 \\ P_n &= \sum_n \frac{(XX(nT_s) - x(nT_s))^2}{N} \\ \text{SNR in dB} &= 10 \log_{10} \left(\frac{P_x}{P_n} \right) \end{aligned}$$

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

Expected Output The expected output of the simulation is given below:



Experiment 2: Pulse Shaping and Matched Filter

Aim We shall implement pulse shaping using square-root raised cosine spectrum pulse for a 4-PAM modulated input stream of symbols, pass it through a passband AWGN channel with various values of the signal-to-noise ratio (SNR), do matched filtering at receiver, and decode the input stream using minimum-distance decoding. We will also plot the symbol error rate against the SNR.

Algorithm for Simulation

1. Generate $N = 10^5$ samples of 4-PAM ($M = 4$) signal taking values in $\{-3, -1, 1, 3\}$. Let us call it d.

2. The samples are up-sampled by a factor of $L = 8$. The up-sampler inserts zeros so that two subsequent 4-PAM samples are separated by $(L - 1)$ zeros. Let us call the signal input stream as v.

3. Generate the square-root-raised-cosine pulse given by:

$$p(t) = \begin{cases} \frac{1}{T_s}\left(1+\beta\left(\frac{4}{\pi}-1\right)\right), & t=0 \\ \frac{\beta}{\sqrt{2}T_s}\left[\left(1+\frac{2}{\pi}\right)\sin\left(\frac{\pi}{4\beta}\right)+\left(1-\frac{2}{\pi}\right)\cos\left(\frac{\pi}{4\beta}\right)\right], & t=\pm\frac{T_s}{4\beta} \\ \frac{1}{T_s}\cdot\frac{\sin\left[\frac{\pi t}{T_s}(1-\beta)\right]+\frac{4\beta t}{T_s}\cos\left[\frac{\pi t}{T_s}(1+\beta)\right]}{\frac{\pi t}{T_s}\left[1-\left(\frac{4\beta t}{T_s}\right)^2\right]}. & \text{elsewhere.} \end{cases}$$

Use roll-factor $\beta = 0.3$, and symbol interval as $T_s = 1$. Sample the pulse in a time-base of $t = -4 : (1/8) : 4$, i.e., to allow pulse-shaping filter span of $N_{\text{sym}} = 8$ symbols, with an over-sampling rate of $L = 8$ samples in every symbol duration.

4. Convolve the up-sampled input stream v with the shaping pulse p to obtain the pulse-shaped input signal s. The convolution will introduce a delay of $D = LN_{\text{sym}}/2 = (\text{length}(p) - 1)/2$.

5. For a given SNR per bit EbN0dB in dB, add Gaussian noise to s to simulate transmission via a baseband AWGN channel. Compute signal power as

$$P \;=\; \sum_n \frac{|s(n)|^2}{NT_s} \;=\; L \cdot \sum_n \frac{|s(n)|^2}{\text{length}(s)}.$$

   The SNR per symbol is given by:

$$\text{EsN0dB} = 10 \log_{10}(\log_2(M)) + \text{EbN0dB}$$

   Then we can compute noise power spectral density $N_0/2$ as

$$(N_0/2) \;=\; \frac{1}{2} \cdot \frac{P}{10^{EsN0dB/10}}.$$

   Generate Gaussian noise samples from standard normal distribution $\mathcal{N}(0, 1)$ and scale it with $\sqrt{N_0/2}$ to make it match with the required SNR. If we denote the stream of noise samples as n, then $r = s + n$. The signal r simulates the received signal obtained at the end of a channel when SNR per bit is EbN0dB.

6. The matched filter at receiver must have impulse response $h[n] = p[-n]$. Since $p[n]$ is symmetric, $p[-n] = p[n]$. Therefore, convolve r with p to to obtain $\hat{s}$. This will further introduce a delay of D samples, making a total delay of $2D$.

7. Sample $\hat{s}$ at $2D + 1 : L : \text{len}(\hat{s}) - 2D$ where D is the delay caused by pulse-shaping filter. Observe that the sampling is done at every Lth sample, and this is done to account for the over-sampling. Due to the convolution with $p[n]$ two times, we shall scale the signal by a factor of $(1/E_p)$ where

$$E_p \;=\; \int |p(t)|^2 dt \;=\; \sum_n |p[n]|^2$$

   The resultant signal after down-sampling and scaling by $(1/E_p)$ is denoted by $\hat{v}$.

8. Demodulate $\hat{v}$ to the nearest 4-PAM point to obtain $\hat{d}$.

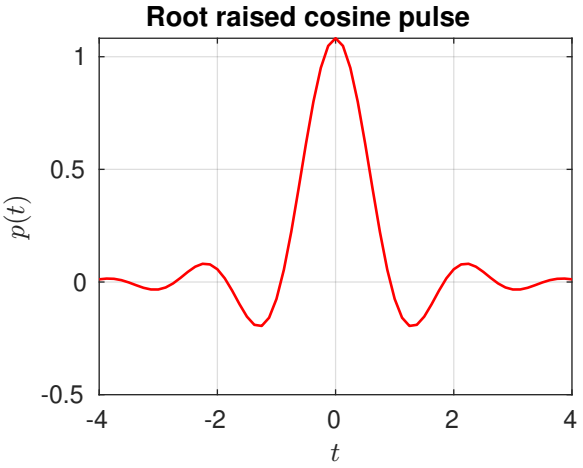
9. One may compute the probability of error as

$$P(E) \;=\; \frac{1}{N} \sum \mathbf{1}(d(i) \neq \hat{d}(i))$$

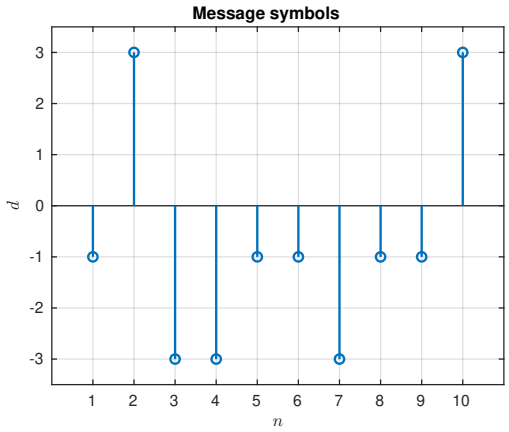
   We can plot $P(E)$ varying EbN0dB in the range -4dB to 12dB increasing by 2dB in every iteration.

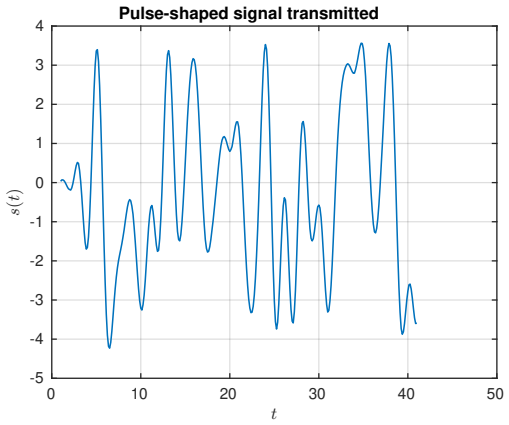
10. The parameters L, β, N_{sym} can be varied to see its effect.

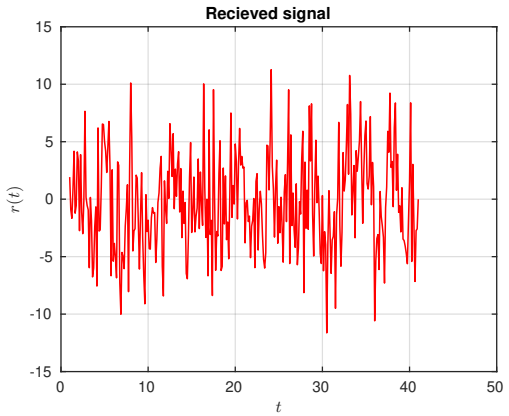
Expected Output We use $L = 8, \beta = 0.3$ and $N_{\text{sym}} = 8$ for all the plots given below. The output for EbN0dB $= -2$dB and EbN0dB $= 12$dB are given. The root-raised-cosine pulse is for 8 symbols taking symbol interval as $T_s = 1$ is given below:

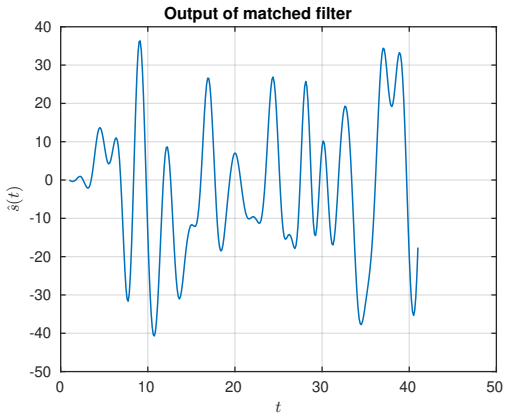


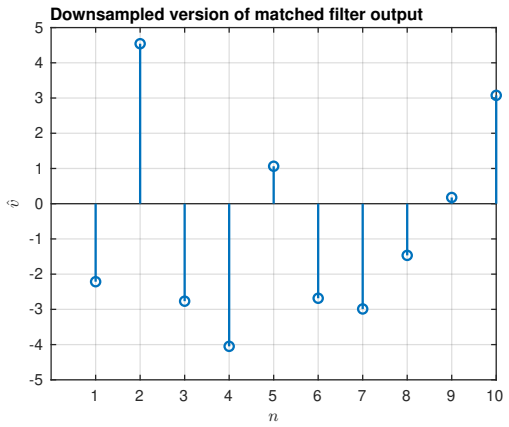
The plots of modulated input stream d, pulse-shaped signal s, received signal r, output of matched filter $\hat{s}$, downsampled decoder input $\hat{v}$, decoded stream of symbols $\hat{d}$ for EbN0dB $= -2$dB is given first.

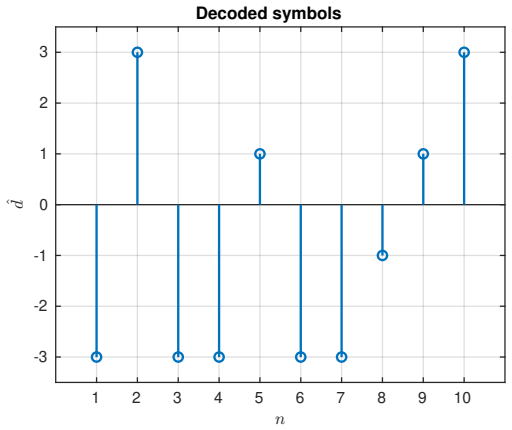




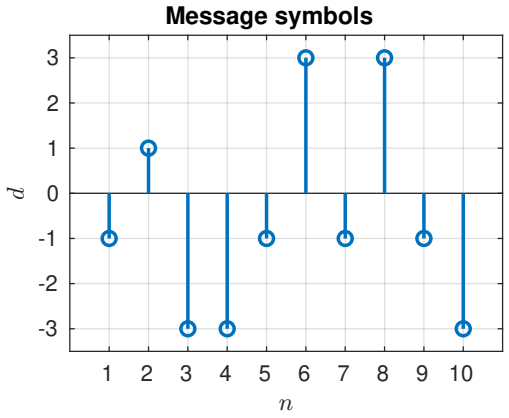


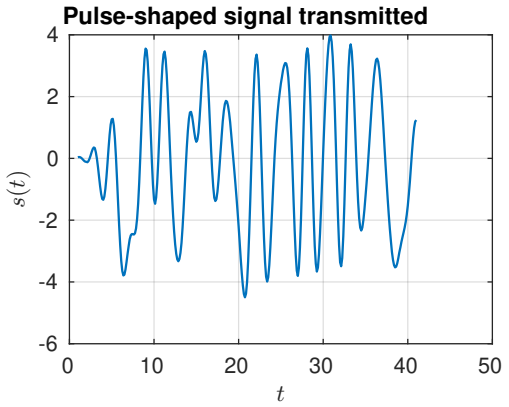


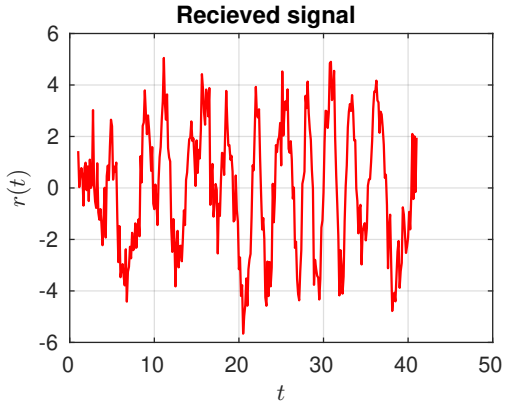


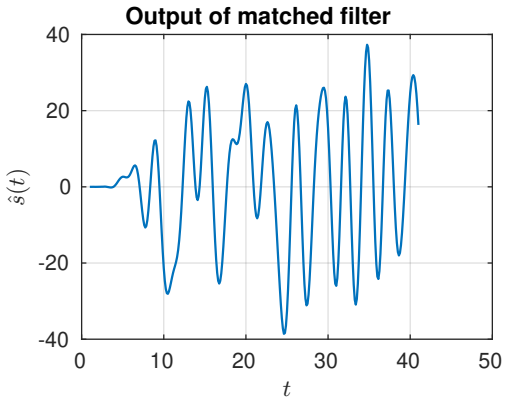


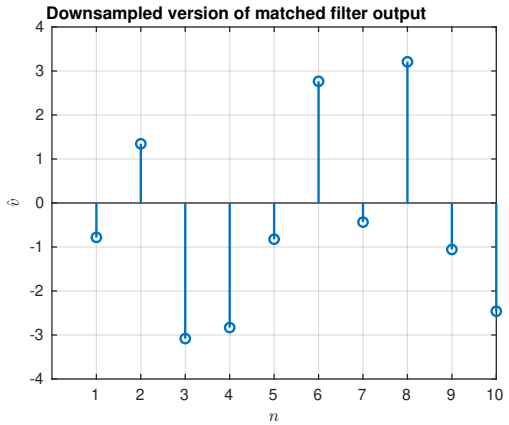
The same set of plots for EbN0dB = 12dB is given below. Observe that $\hat{d} = d$ for the first 10 samples when SNR is high.

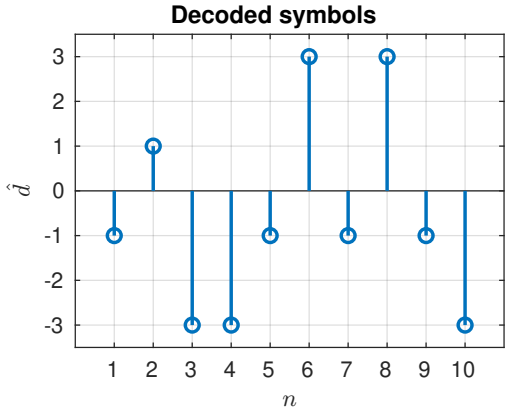




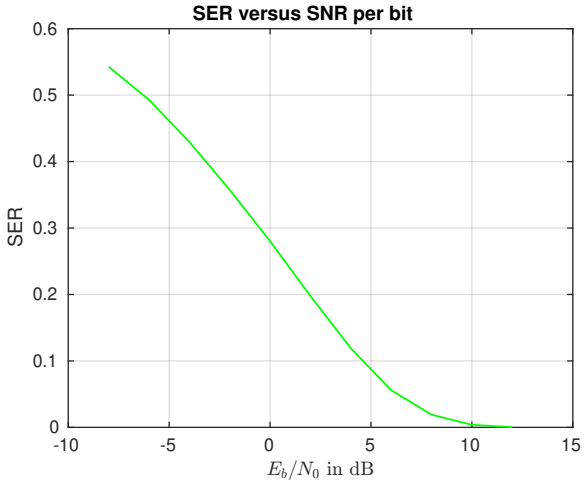








The plot of symbol error rate against EbN0dB = −8dB to 12dB in steps of 2dB is given below.



Experiment 3: Eye Diagram

Aim We shall plot eye diagrams for M-PAM signalling (with $M = 2$ and $M = 4$) using root-raised-cosine shaping pulse while the channel is modeled as a baseband AWGN channel. The eye diagrams with roll off factors $\beta = 0.3$, $\beta = 0.5$ and $\beta = 0.8$ will be plotted with noise and without noise. For every case, time interval over which received signal can be best sampled is identified.

Algorithm for Simulation

1. Generate $N = 10^3$ samples of M-PAM signal taking values in $\{-(M-1), -(M-3), \ldots, -1, 1, \ldots, (M-3), (M-1)\}$. Let us call it d.

2. The samples are up-sampled by a factor of $L = 8$. The up-sampler inserts zeros so that two subsequent M-PAM samples are separated by $(L-1)$ zeros. Let us call the signal input stream as v.

3. Generate the square-root-raised-cosine pulse given by:

$$p(t) = \begin{cases} \frac{1}{T_s}\left(1+\beta\left(\frac{4}{\pi}-1\right)\right), & t = 0 \\ \frac{\beta}{\sqrt{2}T_s}\left[\left(1+\frac{2}{\pi}\right)\sin\left(\frac{\pi}{4\beta}\right)+\left(1-\frac{2}{\pi}\right)\cos\left(\frac{\pi}{4\beta}\right)\right], & t = \pm\frac{T_s}{4\beta} \\ \frac{1}{T_s}\cdot\frac{\sin\left[\frac{\pi t}{T_s}(1-\beta)\right]+\frac{4\beta t}{T_s}\cos\left[\frac{\pi t}{T_s}(1+\beta)\right]}{\frac{\pi t}{T_s}\left[1-\left(\frac{4\beta t}{T_s}\right)^2\right]}. & \text{elsewhere.} \end{cases}$$

   While the roll-factor varies, fix symbol interval as $T_s = 1$. Sample the pulse in a time-base of $t = -4 : (1/8) : 4$, i.e., to allow pulse-shaping filter span of $N_{\text{sym}} = 8$ symbols, with an over-sampling rate of $L = 8$ samples in every symbol duration.

4. Convolve the up-sampled input stream v with the shaping pulse p to obtain the pulse-shaped input signal s. The convolution will introduce a delay of $D = LN_{\text{sym}}/2 = (\text{length}(p)-1)/2$.

5. Fix SNR per bit EbN0dB = 15dB. Add Gaussian noise to s to simulate transmission via a baseband AWGN channel. Compute signal power as

$$P = \sum_n \frac{|s(n)|^2}{NT_s} = L \cdot \sum_n \frac{|s(n)|^2}{\text{length}(s)}.$$

The SNR per symbol is given by:

$$\text{EsN0dB} = 10\log_{10}(\log_2(M)) + \text{EbN0dB}$$

Then we can compute noise power spectral density $N_0/2$ as

$$(N_0/2) \;=\; \frac{1}{2}\cdot\frac{P}{10^{EsN0dB/10}}.$$

Generate Gaussian noise samples from standard normal distribution $\mathcal{N}(0,1)$ and scale it with $\sqrt{N_0/2}$ to make it match with the required SNR. If we denote the stream of noise samples as n, then $r = s + n$. The signal r simulates the received signal obtained at the end of a channel when SNR per bit is EbN0dB.

6. The matched filter at receiver must have impulse response $h[n] = p[-n]$. Since $p[n]$ is symmetric, $p[-n] = p[n]$. Therefore, convolve r with p to do matched filtering. This will further introduce a delay of D samples, making a total delay of $2D$. Due to the convolution with $p[n]$ two times, we shall scale the signal by a factor of $(1/E_p)$ where

$$E_p \;=\; \int |p(t)|^2 dt \;=\; \sum_n |p[n]|^2$$

The resultant signal after scaling by $(1/E_p)$ is denoted by $\hat{s}$. The signal $\hat{s}$ will be used to draw the eye diagram.

7. Implement a subroutine *plotEyeDiagram* to plot eye diagram for the signal $\hat{s}$. The function of the subroutine is described as follows:

   (a) In addition to the input signal, the subroutine will have input arguments L (oversampling factor), m (number of samples per trace), offset (initial offset after which $\hat{s}$ is sampled), t (number of traces).

   (b) A total of $N_1 = mt$ samples are collected from $\hat{s}$ starting from offset + 1.

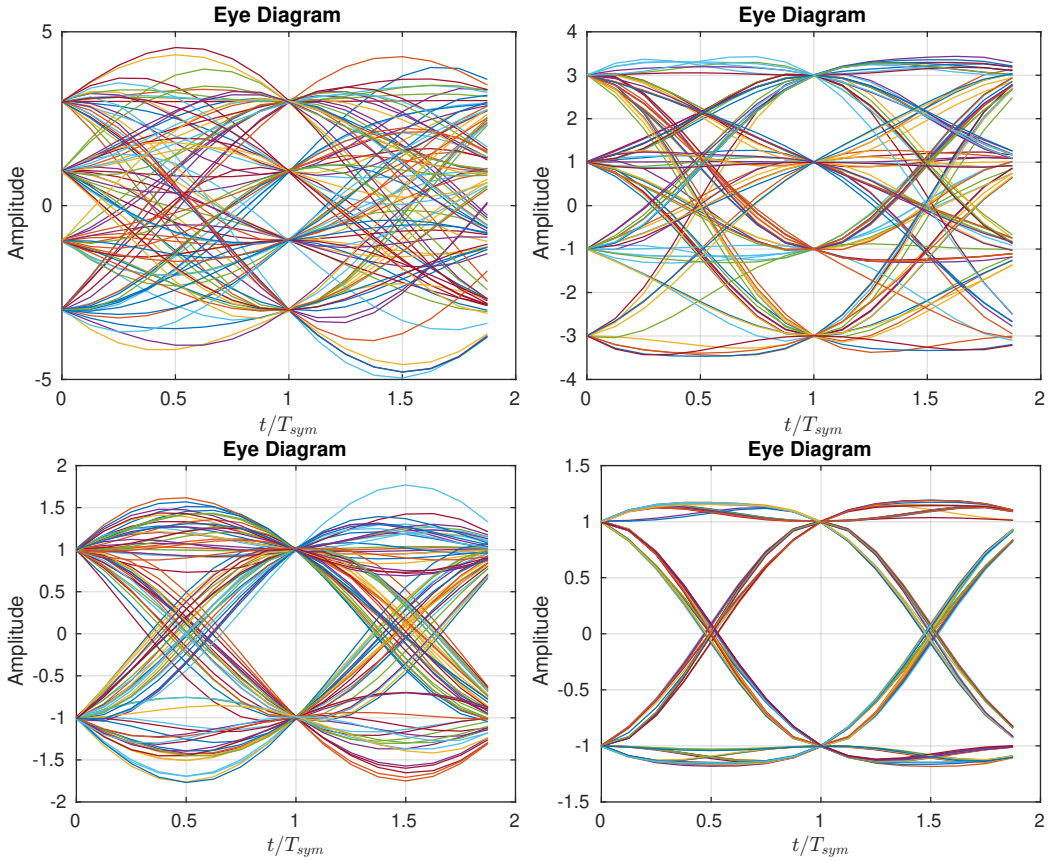
   (c) These samples are reshaped to matrix of size $(m \times t)$.

   (d) Each column of the matrix is plotted on the same plot against a time-base of $(0 : 1 : m-1)/L$. Here, we are normalizing the time by T_s and in any case we have assumed that $T_\text{s} = 1$. The diagram obtained is referred to as the eye diagram.

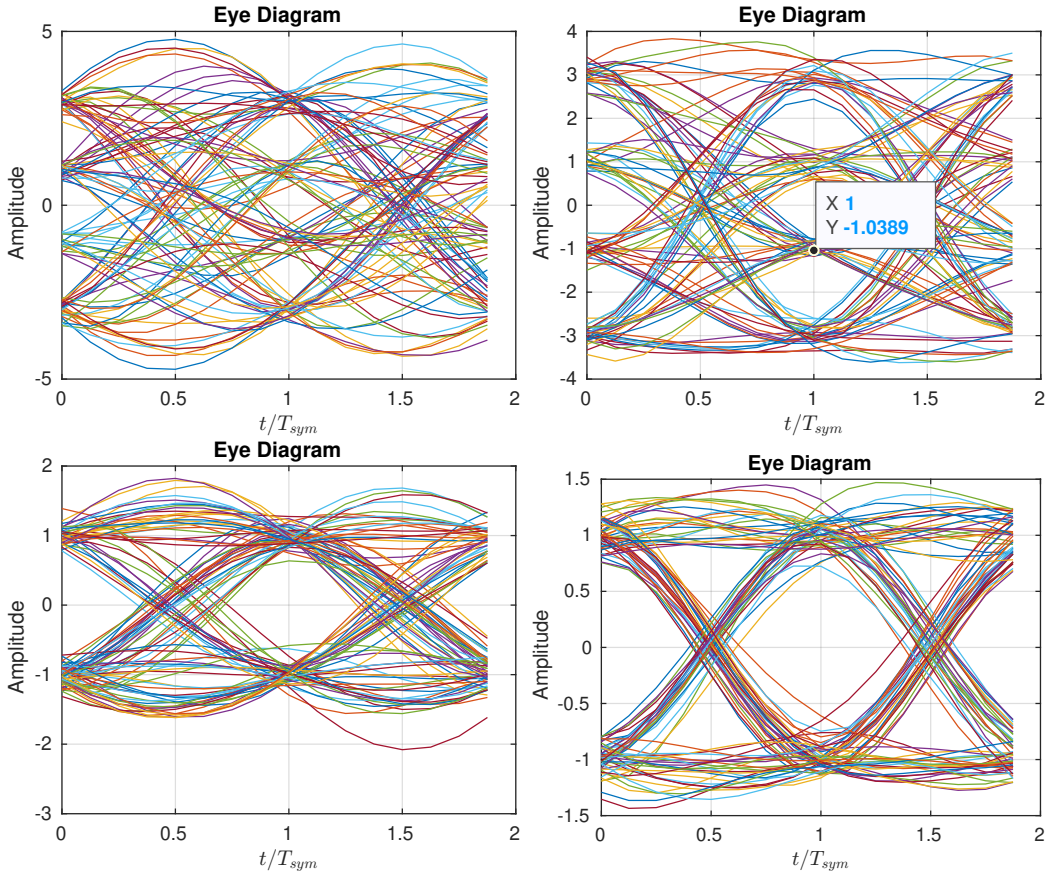
8. The position of eye opening will give the ideal sampling time at the receiver. For $M = 2$, there is one eye with two dominant amplitude levels at sampling time. For $M = 4$ there are three eyes with four dominant amplitude levels at sampling time. The time width associated to the open part of eye gives an interval of best sampling points. This width gives the maximum error tolerance during timing recovery.

9. Plot the eye diagram for $M \in \{2, 4\}$, $\beta \in \{0.3, 0.5, 0.8\}$. The case $M = 2$ is binary signalling and $M = 4$ is quaternary signalling. It shall also be plotted with noise and without noise.

Expected Output The eye diagrams for $M = 4, 2$ for $\beta = 0.3, 0.8$ in that order when there is no noise are given below. The ideal sampling time at the receiver is at multiples of T_s. There is window of tolerance for sampling times.



The eye diagrams for $M = 4, 2$ for $\beta = 0.3, 0.8$ in that order when there channel introduces additive noise are given below. The ideal sampling time at the receiver still remains to be multiples of T_s. There is window of tolerance for sampling times, but it can be a little less than what was the case of no noise.



In every case, a higher β provides a wider eye.

Experiment 4: Binary Phase Shift Keying

Aim We shall simulate the binary phase-shift keying (BPSK) modulation of a bit stream, its transmission over an AWGN channel with a given signal-to-noise ratio (SNR), and decoding by an ML decoder. We will also plot the bit error rate (BER) against SNR per bit E_b/N_0.

Algorithm for Simulation

1. Generate a random binary sequence $\{b_i, i = 1, 2, \cdots N\}$ of length $N = 100000$. Each bit b_i is either 0 or 1 with equal probability half.

2. The AWGN channel is modeled as a vector channel

$$x_i \quad = \quad s_i + w_i, i = 1, 2, \ldots N.$$

with $s_i \in \{\sqrt{E_b}, -\sqrt{E_b}\}$. Input bit 1 is mapped to $\sqrt{E_b}$, and bit 0 to $-\sqrt{E_b}$. The Guassian noise random variable w_i with mean 0 and variance $\frac{N_0}{2}$ can be generated using built-in function.

3. We are interested in the variation of BER against $\frac{E_b}{N_0}$ in dB scale.

$$(E_b/N_0)_{\text{dB}} \quad = \quad 10 \log_{10} \left(\frac{E_b}{N_0} \right)$$

If we fix $E_b = 1$ (i.e., $s_i = +1/-1$), then

$$\frac{N_0}{2} \quad = \quad 0.5 \times 10^{-0.1(E_b/N_0)_{\text{dB}}}$$

Thus we shall vary $(E_b/N_0)_{\text{dB}} \in \{-10, -9.5, -9, \cdots, 9, 9.5, 10\}$, and compute the corresponding $(N_0/2)$ with the above forumla, keeping $E_b = 1$ always. After $(N_0/2)$ is determined for a fixed value of $(E_b/N_0)_{\text{dB}}$, we can generate the AWGN channel and simulate transmission of N bits via the channel. This will generate $\{x_i, i = 1, 2, \cdots N\}$.

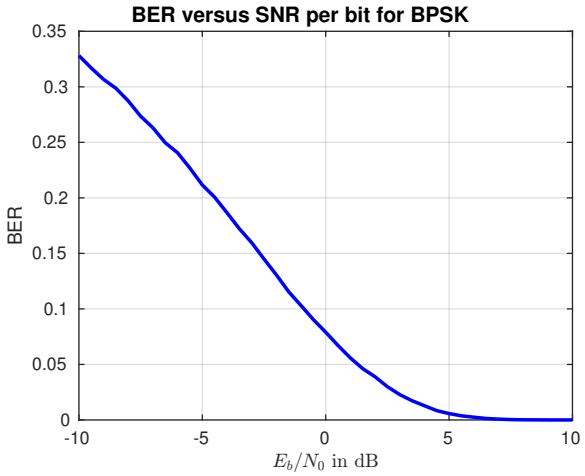
4. Upon receiving $x_i, i = 1, 2, \ldots, N$, receiver makes a decision using:

$$\hat{b}_i \quad = \quad \begin{cases} 1, & x_i \geq 0 \\ 0, & x_i < 0 \end{cases}$$

5. The probability of error

$$\text{BER}((E_b/N_0)_{\text{dB}}) \quad = \quad \frac{\text{Number of times } \hat{b}_i \neq b_i}{N}$$

Expected Output The BER varies with $(E_b/N_0)_{\text{dB}}$ as a waterfall curve, and is plotted below. The SNR per bit varies from -10dB to 10dB.



Experiment 5: Quadrature Phase Shift Keying

Aim We shall simulate the quadrature phase-shift keying (QPSK) modulation of a bit stream, its transmission over an AWGN channel with a given signal-to-noise ratio (SNR), and decoding by an ML decoder. We will also plot both the symbol error rate (SER) and the bit error rate (BER) against SNR per bit E_b/N_0.

Algorithm for Simulation

1. Generate a random binary sequence $\{b_i, i = 1, 2, \cdots N\}$ of length $N = 2 * 50000$. Each bit b_i is either 0 or 1 with equal probability half. The binary sequence b is split into two binary subsequences b_I and b_Q as illustred by an example below:

$$\begin{aligned} b &= 101110101000\ldots \\ b_I &= 11110\ldots \\ b_Q &= 01000\ldots \end{aligned}$$

2. The AWGN channel is modeled as a vector channel

$$\begin{aligned} x_i &= s_i + w_i, \quad i = 1, 2, \ldots (N/2) \\ \text{with} \quad s_i = \begin{bmatrix} s_{i1} \\ s_{i2} \end{bmatrix} &\in \left\{ \begin{bmatrix} \sqrt{E/2} \\ \sqrt{E/2} \end{bmatrix}, \begin{bmatrix} -\sqrt{E/2} \\ \sqrt{E/2} \end{bmatrix}, \begin{bmatrix} -\sqrt{E/2} \\ -\sqrt{E/2} \end{bmatrix}, \begin{bmatrix} \sqrt{E/2} \\ -\sqrt{E/2} \end{bmatrix} \right\} \\ &:= \{s_1, s_2, s_3, s_4\}. \end{aligned}$$

   Each dibit (pair of bits) $(b_{Ii}, b_{Qi}) \in \{11, 01, 00, 10\}$ is mapped to s_1, s_2, s_3 and s_4 in order. The Guassian noise random vector w_i has two components w_{i1} and w_{i2}. Each of w_{ij} has mean 0 and variance $\frac{N_0}{2}$, and can be generated using built-in function. Though simulation is oblivious to it, it must be known that the actual signal transmitted is

$$s_i(t) = s_{i1}\sqrt{\frac{2}{T}}\cos(2\pi f_c t) - s_{i2}\sqrt{\frac{2}{T}}\sin(2\pi f_c t)$$

3. We are interested in the variation of BER against $\frac{E_b}{N_0}$ in dB scale. Since $E = 2E_b$

$$(E_b/N_0)_{\text{dB}} = 10\log_{10}\left(\frac{E}{2N_0}\right)$$

If we fix $E = 2$ i.e., $s_{ij} \in \{+1, -1\}$, then

$$\frac{N_0}{2} = 0.5 \times 10^{-0.1(E_b/N_0)_{\text{dB}}}$$

Thus we shall vary $(E_b/N_0)_{\text{dB}} \in \{-10, -9.5, -9, \cdots, 9, 9.5, 10\}$, and compute the corresponding $(N_0/2)$ with the above forumla, keeping $E = 2$ always. After $(N_0/2)$ is determined for a fixed value of $(E_b/N_0)_{\text{dB}}$, we can generate the AWGN channel and simulate transmission of N bits via the channel. This will generate $\{x_i, i = 1, 2, \cdots N/2\}$. Note that the number of received symbols are half of the number of bits transmitted.

4. Upon receiving $x_i, i = 1, 2, \ldots, N/2$, receiver makes a decision using the rule:

$$(\hat{b}_{Ii}, \hat{b}_{Qi}) = \begin{cases} (11), & x_{i1} \geq 0, x_{i2} \geq 0 \\ (01), & x_{i1} < 0, x_{i2} \geq 0 \\ (00), & x_{i1} < 0, x_{i2} < 0 \\ (10), & x_{i1} \geq 0, x_{i2} < 0 \end{cases}$$

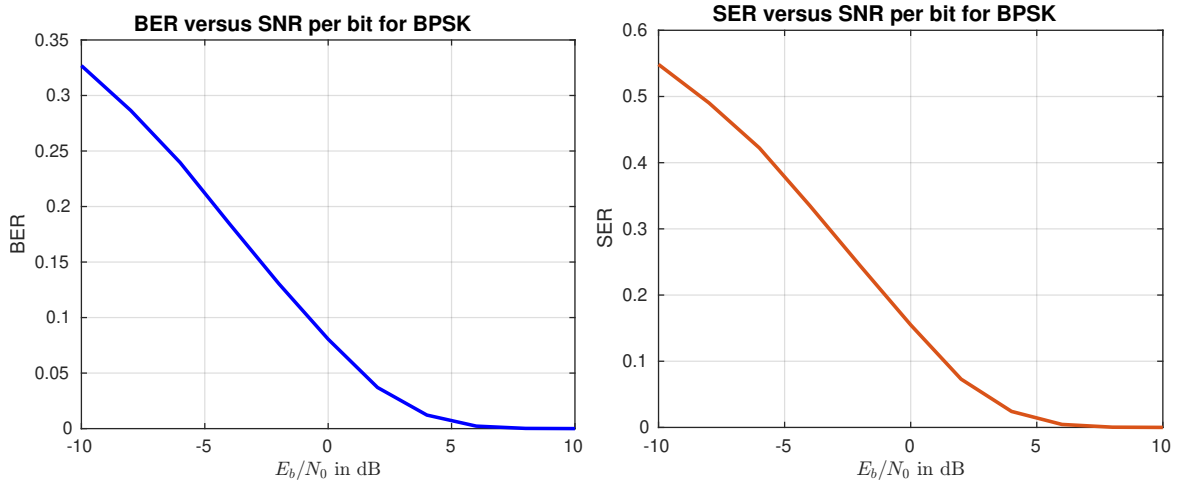
5. The probability of error

$$\text{SER}((E_b/N_0)_{\text{dB}}) = \frac{\text{Number of times } (\hat{b}_{Ii}, \hat{b}_{Qi}) \neq (b_{Ii}, b_{Qi})}{(N/2)}$$

$$\text{BER}((E_b/N_0)_{\text{dB}}) = \frac{\text{Number of times } \hat{b}_i \neq b_i}{N}$$

Compare the curves with that of BPSK.

Expected Output Both the BER and SER varies with $(E_b/N_0)_{\text{dB}}$ as waterfall curves, and are plotted below. The SNR per bit varies from -10dB to 10dB.



Observe that SER is larger than BER as an error in a bit implies an error in the symbol it is part of, where as an error in a symbol (comprising of two bits) need not imply errors in both bits.

Emulation Experiments

Instructions

1. We prefer to use Ubuntu in DSP Lab to conduct emulation experiments using software-defined-radio (SDR). If anyone decides use Windows, you are permitted to do so, but limited help will be offered to debug any issue that you may face.

2. Please ensure that the following command line programs are already installed in your machine: rtl_sdr, rtl_test, rtl_tcp, rtl_fm. You can write rtl in terminal, press tab, and verify if they are in place.

3. Please use the commands below to install two additional softwares Gqrx and GNU Radio.

   $ sudo apt install gqrx-sdr

   $ sudo apt install gnuradio

4. You can use the dongle-only RTL-SDR available in DSP Lab. If you wish to play with it at your home, you can think of purchasing either of the following products:

   (a) R820T2 RTL2832U 1ppm tcxo SMA software designed radio with dipole antenna

   (b) E4000 RTL2832U 0.5–1ppm tcxo SMA software designed radio with dipole antenna

   The components mentioned as part of products are described below:

   - R820T2: a tuner chip designed by Raphael Micro.
   - E4000: a tuner chip designed by Elonics.
   - RTL2832U: a DVB-T demodulator chipset designed by Realtek.
   - ppm: parts per million error in frequency calibration.
   - tcxo: temperature compensated oscillator.
   - SMA: SubMiniature Version A is a type of RF coaxial connector used to connect with an antenna.

Experiment 1: Familiarization with SDR

Aim We shall familiarize with an RTL-SDR dongle, tune to a commercial FM channel using gqrx software, collect I/Q sample, and plot them.

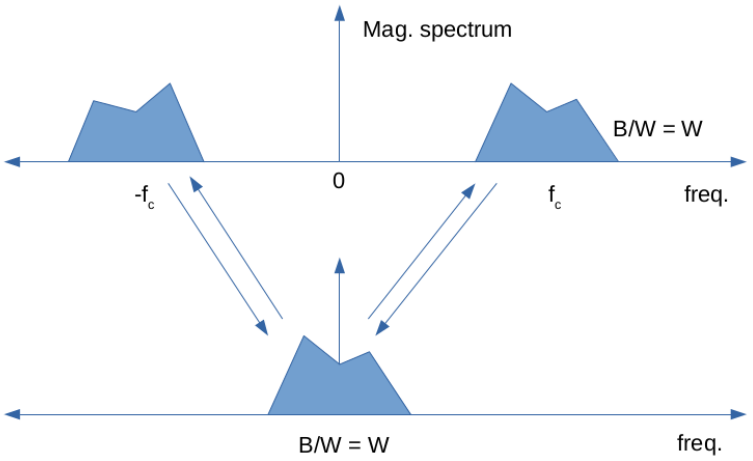
An Introduction to Software Defined Radios (SDR) and RTL-SDRs

1. **What are SDRs?:** [Reference: `https://www.rtl-sdr.com/about-rtl-sdr/`]

   Radio components such as modulators, demodulators and tuners are traditionally implemented in analog hardware components. The advent of modern computing and analog to digital converters (ADC) allows most of these traditionally hardware based components to be implemented in software instead. This led to software defined radios (SDR) that not only allows signal processing using computers, but also reconfigurability of radio systems via software.

2. **Basic Function of SDRs:**

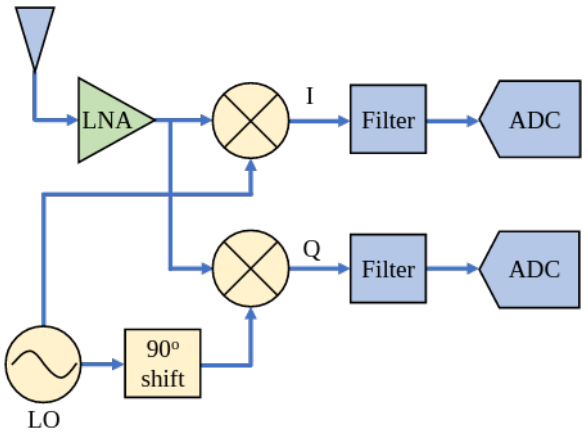
   A real bandpass signal $g(t)$ can be represented by an equivalent complex baseband (low-pass) signal. The magnitude spectrum of the equivalent baseband signal is shown in figure below.



It is straightforward to see that the baseband signal $g_\ell(t)$ preserves information contained in the spectrum of bandpass signal, but as it sacrifices the property of evenness, the resultant baseband signal can be complex. Thus we write $g_\ell(t) = i(t) + jq(t)$ where $i(t)$, $q(t)$ are respectively referred to as the in-phase and quadrature components of $g(t)$. The mathematical expression relating $g(t)$ to $g_\ell(t)$ is given by:

$$g(t) \quad = \quad i(t)\cos(2\pi f_c t) - q(t)\sin(2\pi f_c t),$$

where f_c is the center frequency of $g(t)$. Conversely, the in-phase and quadrature components can be obtained from $g(t)$ and an SDR precisely does this job. The samples of $i(t)$ and $q(t)$ are generally referred to as I/Q samples. The basic function of an SDR is to supply I/Q samples of a real bandpass signal to a computer. This is depicted in figure below.



(Image courtesy: `https://pysdr.org/content/sampling.html`)

There are two receiver architectures, the first directly converts RF signal to baseband signal, and the second down-converting incoming RF signal first to an intermediate frequency (IF) signal before taking it to the baseband. The first is referred to as zero-IF receiver and second as IF receiver.

3. **The genesis of RTL-SDRs:** [Reference: `https://www.rtl-sdr.com/about-rtl-sdr/`]

   RTL-SDR is a very cheap USB dongle that can be used as an SDR for receiving live radio signals. Depending on the particular model it could receive frequencies from 500 kHz up to 1.75 GHz. Most software for the RTL-SDR is also community developed, and provided free of charge. Note that RTL-SDRs cannot transmit.

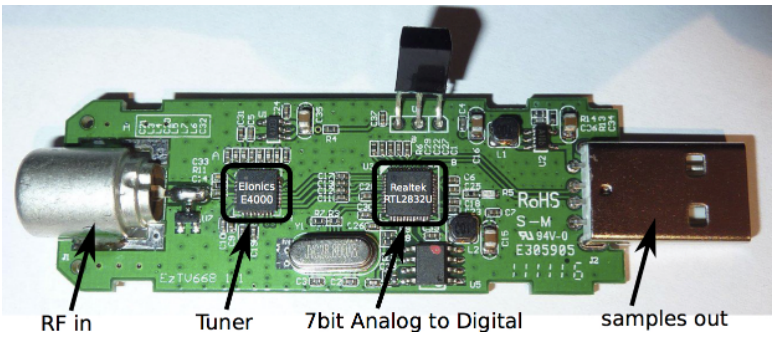
The origins of RTL-SDR stem from mass produced DVB-T TV tuner dongles manufactured by Realtex. With the combined efforts of Antti Palosaari (Feb 2012), Eric Fry (March 2010) and Osmocom (in particular Steve Markgraf) it was found that the raw I/Q data on the RTL2832U chipset inside these DVB-T TV tuner dongles could be accessed directly, which allowed the DVB-T TV tuner to be converted into a wideband software defined radio via a custom software driver developed by Steve Markgraf. Thus the invention of RTL-SDRs is the result of a hacking attempt made on a commercially available TV tuner dongles. Thanks to the RTL2832U chipset inside these dongles, the SDRs thus developed are known as RTL-SDRs.

4. **The backbone chip (RTL2832U) of an RTL-SDR:** [Reference: `http://superkuh.com/rtlsdr.html`]

   The RTL2832U is originally designed as a demodulator for the DVB-T TV tuner. However in RTL-SDRs, they are repurposed to produce I/Q samples of an RF signal. The RTL2832U chips use a phased locked loop based synthesizer to function as the local oscillator required to produce center frequency f_c. The quadrature mixer produces the complex baseband signal, and they are sampled by the ADC to produce I/Q samples. The Sigma-Delta ADC samples at a high rate but low precision, producing a 28.8 mega samples per second stream, each sample encoded to 8 bits. That can be resampled inside the RTL2832U to present whatever sample rate is desired to the host PC. The resampled output can be up to 3.2 MS/s but 2.56 MS/s is the max recommended to avoid losing samples. But some people have are able to work well with 2.8MS/s and 3.2 MS/s on some USB 3.0 ports. The minimum resampled output supported is 0.5 MS/s.

5. **The tuning range of RTL-SDR:** [Reference: `http://superkuh.com/rtlsdr.html`]

   The RTL-SDRs are available with different types of tuner chipsets, and it determines the tuning range. Among them, two types of tuner chipsets are well-known. The first one is the Elonics E4000 chipset that works with a zero-IF receiver architecture and support $52 - 2200$ MHz with a gap from 1100 MHz to 1250 MHz. The second one is the Raphael Micro R820T (or R820T2) that uses an IF frequency of 3.57 MHz or 4.57 MHz. It supports a tuning range of $24 - 1766$ MHz. The internal diagram of an RTL-SDR with Elonics E4000 tuner chipset inside (that is used in our lab) is shown below.



Plotting I/Q Samples

1. Plug the RTL-SDR into USB port. You can check lsusb to see if the device is identified by the OS.

2. We can collect I/Q samples into a binary file by using the command

   $ rtl_sdr iqdata.bin -s 1.8e6 -f 101.9e6

3. Use a MATLAB program to read the binary file and store it into an array s of integers.

4. Then $s[1:2:end]$ and $s[2:2:end]$ will respectively provide I and Q samples. Subtract the signal values by 128 to convert from the default data type of unsigned int to the actual signed int.

5. Plot the I/Q samples separately and the envelope $\sqrt{I^2+Q^2}$.

Using Gqrx to tune to an FM channel We can use Gqrx software to tune to an FM channel after plugging the RTL-SDR into an USB port. This is a way to test if the RTL-SDR is working fine.

Expected Output You can provide (a) the plot of I/Q samples and the envelope (b) the gqrx GUI interface showing the tuning to an FM channel.

Experiment 2: FM Reception Using SDR

Aim We shall tune to a commercial FM channel, receive I/Q samples using RTL-SDR, emulate an FM receiver using GNU Radio, and play it using computer speakers. We will try to emulate an FM mono receiver as well as a stereo one.

An Introduction to GNU Radio (Use `https://wiki.gnuradio.org/index.php/Main_Page` to write some introductory information on GNU Radio software.)

GNU Radio Flow Graph of FM Receiver (Use the grc file shared in Google Class room to obtain the flow graph.)

Playing an FM Channel

1. Plug in the RTL-SDR into the USB port. Create a new GNU Radio project for FM receiver.

2. Use RTL-SDR Source as a block in GNU Radio project to acquire I/Q samples via the SDR. Use sampling frequency of 2×10^6 samples per second. Choose the channel frequency as 101.9MHz (Akashavani broadcast frequency).

3. Use a rational resampler to down-convert the sampling rate 500×10^3 samples per second.

4. Use a low-pass filter block with cut-off frequency of 200kHz and transition width of 20kHz to remove unwanted frequencies beyond the FM spectrum.

5. Use a block of WBFM Receive (for mono) or WBFM PLL Receive (for stereo) to demodulate the signal.

6. Adjust the samplng rate using a rational resampler block to match with rates supported by computer speakers (for eg: 24, 48 kHz).

7. Generate and execute the flow graph to emulate an FM receiver.

Expected Output The FM channel is played on computer speakers.