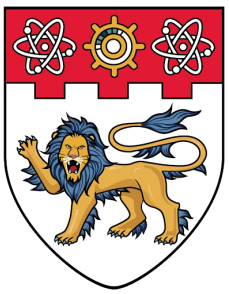


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering

CE3006 : Digital Communications

Project Report:
Simulation of Communication Systems using MATLAB

Submitted by:
Group 10

Name	Matric Number
Ang Qi Xuan	U1621610H
Arya Sreekumar	U1622224L
Hernandez, Nikki Arcilla	U1621592F
Hubert Angelo	U1721107E
Ong Yong Siang	U1722030B

Introduction

This project simulates different modulation and demodulation schemes and analyses their bit-error rate performance.

Phase 1: Baseband Modulation

This section involves the simulation of baseband modulation and demodulation. Firstly, randomly generated binary data will be modulated and transmitted. Then, additive white Gaussian noise will be added to the transmitted signal to simulate channel effects, this will be the received signal. Afterwards, the received signal will pass through a threshold logic to obtain the original binary data. Lastly, the bit-error rate performance of the baseband modulation will be analysed against different SNR values.

Transmitter

Step 1: Generate Random Binary Data

The baseband data to be transmitted is produced by randomly generating 1024 bits (n = 1024) of data with a uniform distribution ('0' and '1' have the same probability of occuring)

$$Data = randi([0\ 1], 1, n)$$

Matlab Code

Then we will convert binary generated to -1 and +1 for transmission

$$signal = 2 .* Data - 1$$

Matlab Code

Channel

Step 2: Generate White Gaussian Noise

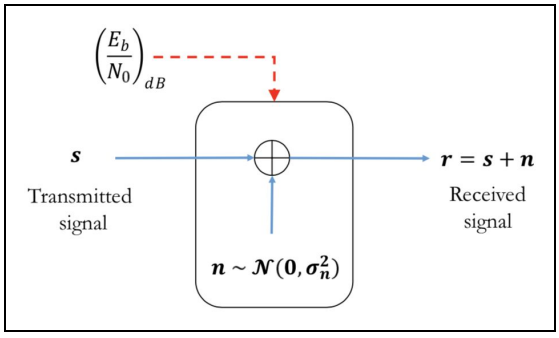
Generate normally distributed noise samples with zero mean and unit variance. Generate the same amount of noise samples as the signal

$$noise = randn(size(signal));$$

Matlab Code

Step 3: Add White Gaussian Noise

The logic of adding AWGN is shown in the figure below. The transmitted signal is added to the generated AWGN to obtain the received signal.



The proportion of the noise samples that will be added to the transmitted signal is determined by the Noise power (or variance), *N*, which is derived from the SNR equation:

$$SNR = 10\, log_{10}\,(S/N_0)$$

Assuming Signal power (or variance) has unit power (S = 1), we can derive N:

$$SNR = 10\, log_{10}\,(1/N)$$
$$SNR = -10\, log_{10}\,(N)$$
$$-SNR/10 = log_{10}\,(N)$$
$$N = 10^{(-SNR/10)} \text{ (Noise power in watts)}$$
$$\sigma = \sqrt{\frac{N}{2}}$$

The noise variance is multiplied to the generated noise samples to adjust its proportion to the signal then added to the transmitted signal to obtain the received signal. The figure below shows the MATLAB code for noise addition.

```
% ------ Noise Addition ------
function noise_signal = add_noise(s0,signal_power,snr_db)
    n = randn(size(s0)); % generate noise samples

    SNR = power(10,snr_db/10); % calculate noise variance
    Np = signal_power/SNR; %calculate noise power

    noise = sqrt(Np/2).*n;
    noise_signal = s0 + noise;

end
```

Receiver

Step 4: Apply Threshold Logic at the receiving end

The threshold logic will be used to convert the signal transferred over the channel back to its binary equivalent.

```
% ----- Apply Threshold -----
function y = signal_threshold(s0, threshold)
        for cols = 1:size(s0,2)
            if s0(cols) >= threshold
                s0(cols) = 1;
            else
                s0(cols) = 0;
            end
        end
    y = s0;
end
```

Step 3: Compute Bit Rate

We will then use the receiving signal to compare it against the original data . With the error will then divide by the number of bits (1024) to find our error rate.

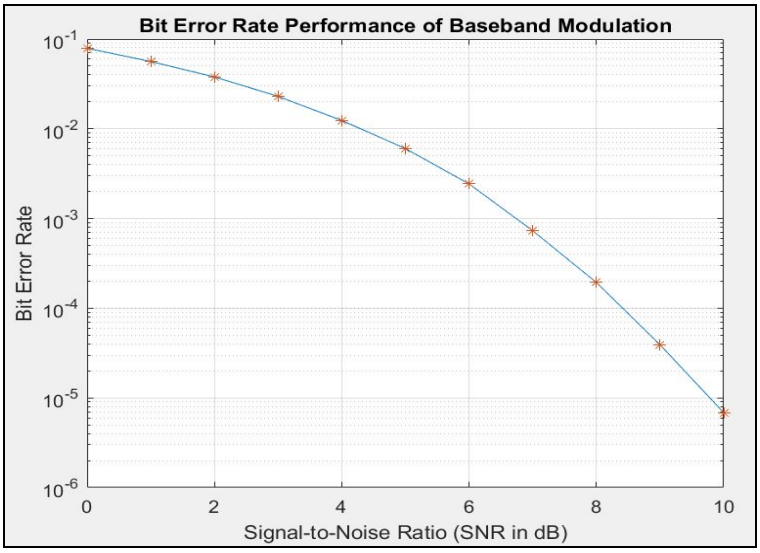
```
% ----- Compute Error -----
error_list = [];
steps = 10;
for k = 1: size(snr,2)    % traverse across col
    avg_error = 0;   %get average error
    for m = 1:steps
        noise_signal = add_noise(trans_data, signal_power, snr(k));
        received_signal = signal_threshold(noise_signal, 0);
        error_rate = compute_error(bin_data,received_signal);
        avg_error = avg_error + error_rate;
    end
    avg_error = avg_error / steps;
    error_list = [error_list,avg_error];
```

```
% -----Compute Bit error-----
function error_rate = compute_error(input,received)
    error = 0;
    for k = 1: size(input,2)
        if input(k) ~= received(k)
            error = error + 1;
        end
    end
    error_rate = error / size(input,2);       %calculate bit error rate
end
```

Bit-Error Rate

Compute and plot bit-error rate (BER) for different SNR values.

Figure 1 shows the relationship between SNR and BER. As SNR increases, BER decreases.



Phase 2: Modulation for communication

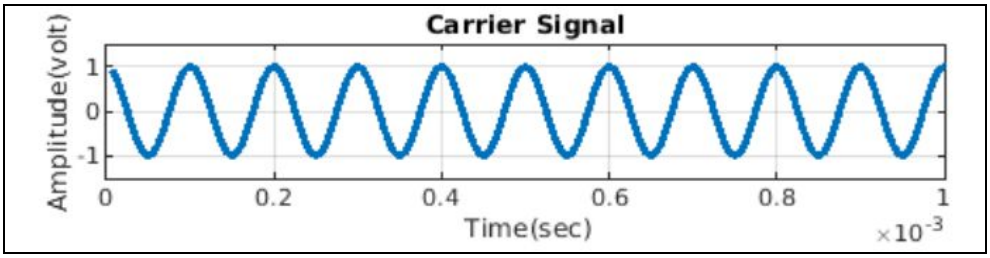
This section is related to band-pass modulation and demodulation. We have implemented On-Off Keying, Binary Phase Shift Keying and Binary Frequency Shift Keying.

Defining carrier frequency, over sampling rate and baseband frequency

```
N = 1024;
br = 1000;       % bit rate of 1kps
bp= 1/br;            % bit period
L=16; %oversampling factor,L=Tb/Ts(Tb=bit period,Ts=sampling period)
%if a carrier is used, use L = Fs/Fc, where Fs >> 2xFc

Fc=10000; %carrier frequency
Fs=L*Fc;%sampling frequency

signal_length = Fs/br;
```

The number of bits is 1024, bit rate is 1000bits/second. The bit period is 0.001s and the sampling factor is 16 as the carrier signal is 16 times oversampled (means one cycle contains 16 samples). The signal length is the number of samples in 1 bit period. The sampling frequency is 160KHz.

The figure below shows the carrier signal, $cos\,(2\pi ft)$ with a frequency of 10KHz.



Since carrier frequency = 10 KHz and baseband data rate = 1000 bits per second,

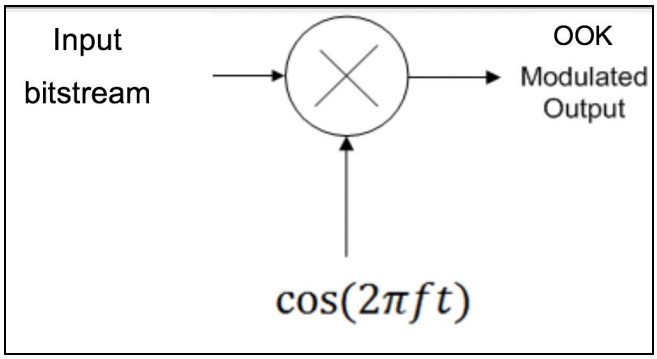
$$number\ of\ cycles\ /\ bit\ = \frac{10000Hz}{1000\ bps}\ =\ 10\ cycles\ /\ bit$$
$$sampling\ frequency\ (f_s)\ =\ 160KHz$$
$$number\ of\ sampled\ cycles\ per\ bit\ =\ 160$$

1. On-Off Keying

On-Off Keying is the simplest form of amplitude-shift keying (ASK) modulation that represents digital data at the presence or absence of a carrier wave. The presence of a carrier wave for a specific duration represents a binary 1, while its absence for the same duration represents a binary 0.

Modulation

OOK modulation is summarised in the figure below.



Step 1: Generate the modulated signal

OOK signals are realised by linear modulation. Using a mixer, the carrier signal is multiplied with the baseband symbol stream.

```
% Modulation
ook_mod_signal = ook_orig_signal.*carrier;
```

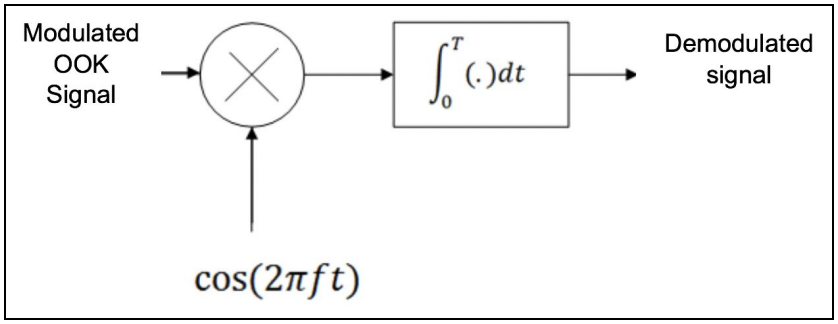
Step 2:

Additive white noise is added with required SNR value to the modulated signal. The received signal is modulated signal added with the noise using the 'add_noise' function explained above.

```
% Add Noise
ook_mod_noise_signal = add_noise(ook_mod_signal,1,snr);
```

Demodulation

A modulated signal must be demodulated in order to recover the original signal. The OOK demodulation logic is summarised in the figure below.



Step 1: Coherent demodulation of the OOK signal is performed at the receiver. Coherent demodulation requires the received signal to be multiplied with the carrier having the same frequency and phase as at the transmitter.

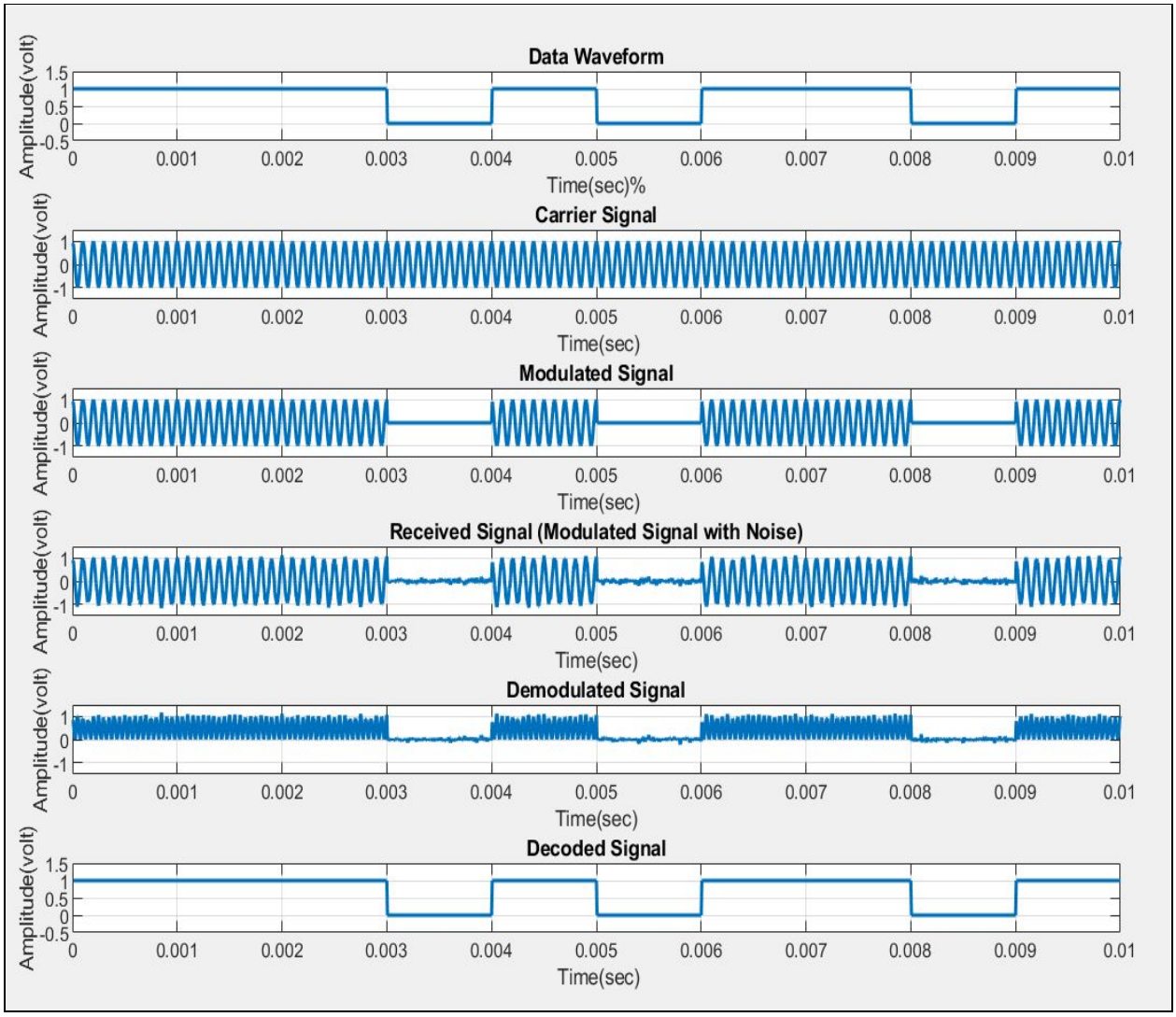
Step 2: After the multiplication with the carrier signal, the signal is integrated over the bit duration 'bp' and sampled. Then thresholding is applied to determine if a '1' was sent (+ve voltage) or a '0' was sent (-ve voltage). Here, the threshold applied is 0.5.

```
% Demodulation
ook_demod_signal = ook_mod_noise_signal(1,:).*carrier;
int_op = [];
for i = 0:signal_length:length(ook_demod_signal)-signal_length
    int_ = trapz(ook_demod_signal(i+(signal_length/2):i+(signal_length/2)+1));
    int_op = [int_op int_];
end

% Threshold
th = 0.5;
ook_rec_bits = (round(int_op,1)> th);
ook_rec_signal = replicate(ook_rec_bits, signal_length);
```

End to end simulation

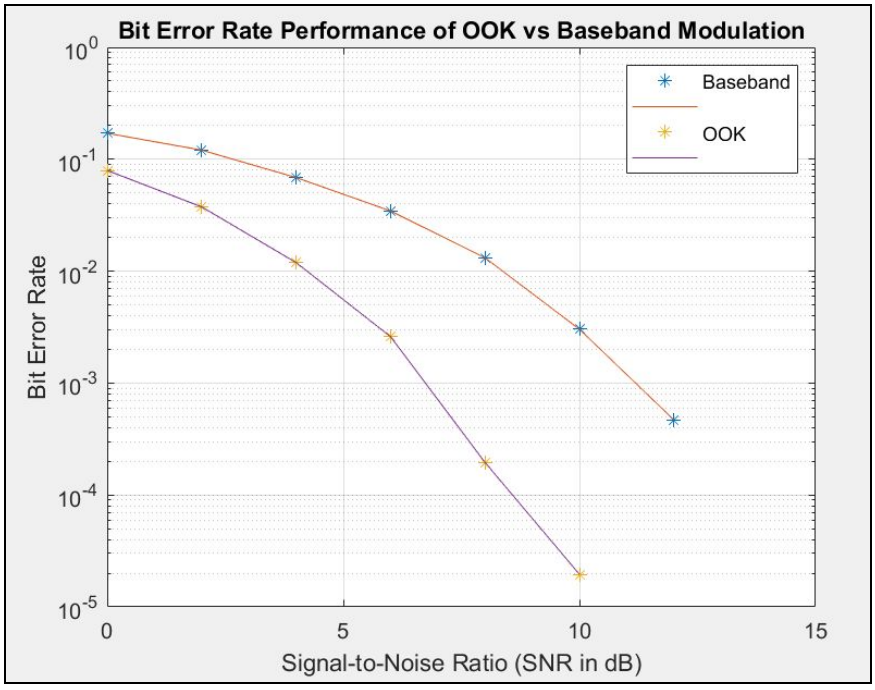
The complete waveform simulation for the end-to-end transmission of information using OOK modulation is shown in the figure below.

1. The 1st plot shows the generation of random message bits (data waveform).
2. The 2nd plot shows the carrier signal $cos\,(2\pi ft)$
3. The 3rd plot shows the modulated OOK signal
4. The 4th plot shows the addition of AWGN noise according to the chosen signal-to-noise ratio.
5. The 5th plot shows the OOK demodulation of the noisy signal using a coherent receiver.
6. The 6th plot shows the decoded signal (original message) for OOK.



Bit Error Rate Performance

The graph below shows the bit-error rate performance of OOK modulation against Baseband modulation done in Phase 1. As shown in graph, OOk has better performance than baseband modulation. This shows the advantage of using bandpass modulation in bit-error rate performance by being more robust to noise, thus, requiring less power (SNR) for the same level of performance.



2. Binary Phase Shift Keying

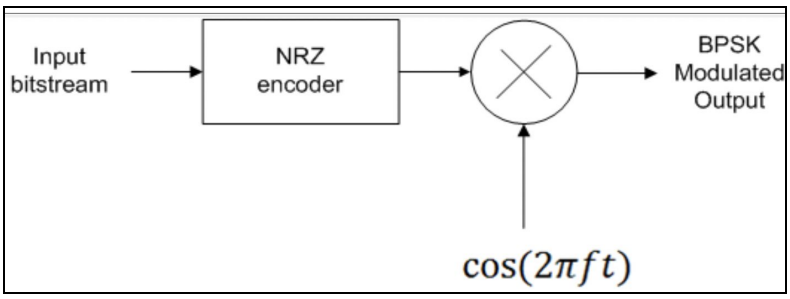
Binary Phase Shift Keying (BPSK) is a two phase modulation scheme, where the 0's and 1's in a binary message are represented by two different phase states in the carrier signal: θ=0 degree for binary 1 and θ=180 degrees for binary 0 for 1 bit symbols. Hence,

$$S_1 = \sqrt{\frac{2E}{T}}\cos(2\pi ft) \rightarrow represents\ '1'$$

$$S_2 = -\sqrt{\frac{2E}{T}}\cos(2\pi ft) \rightarrow represents\ '0'$$

Modulation

BPSK modulation is summarised in the figure below.



Step 1: Signals need to be in +1 or -1 format. Hence a 'for' loop is used to convert every generated data bit that is 0 to -1 as shown below. This is also the polar NRZ encoding.

```
for k =1:length(bpsk_orig)
    if bpsk_orig(k) == 1
        bpsk_orig(k) = 1;
    else
    bpsk_orig(k)=-1;
    end
end
```

Step 2: The data samples are modulated with carrier signal (cos(2pi.f.t)) where f is the carrier frequency as shown below. The data signal is multiplied to the carrier signal to obtain the modulated signal. The BPSK modulated output is the binary input converted to NRZ line codes and multiplied with carrier signal.

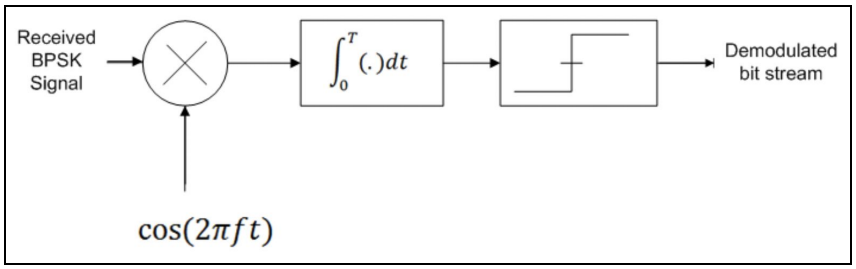
```
% Modulation
bpsk_mod_signal = bpsk_orig_signal.*carrier;
```

Step 3: Additive white noise is added with required SNR value to both the modulated signals. The received signal is modulated signal added with the noise using the same function used in OOK above.

```
% Add Noise
bpsk_mod_noise_signal = add_noise(bpsk_mod_signal,1,5);
```

Demodulation

BPSK demodulation is summarised in the figure below.



Step 1: Coherent demodulation of the BPSK signal is performed at the receiver. Coherent demodulation requires the received signal to be multiplied with the carrier having the same frequency and phase as at the transmitter.

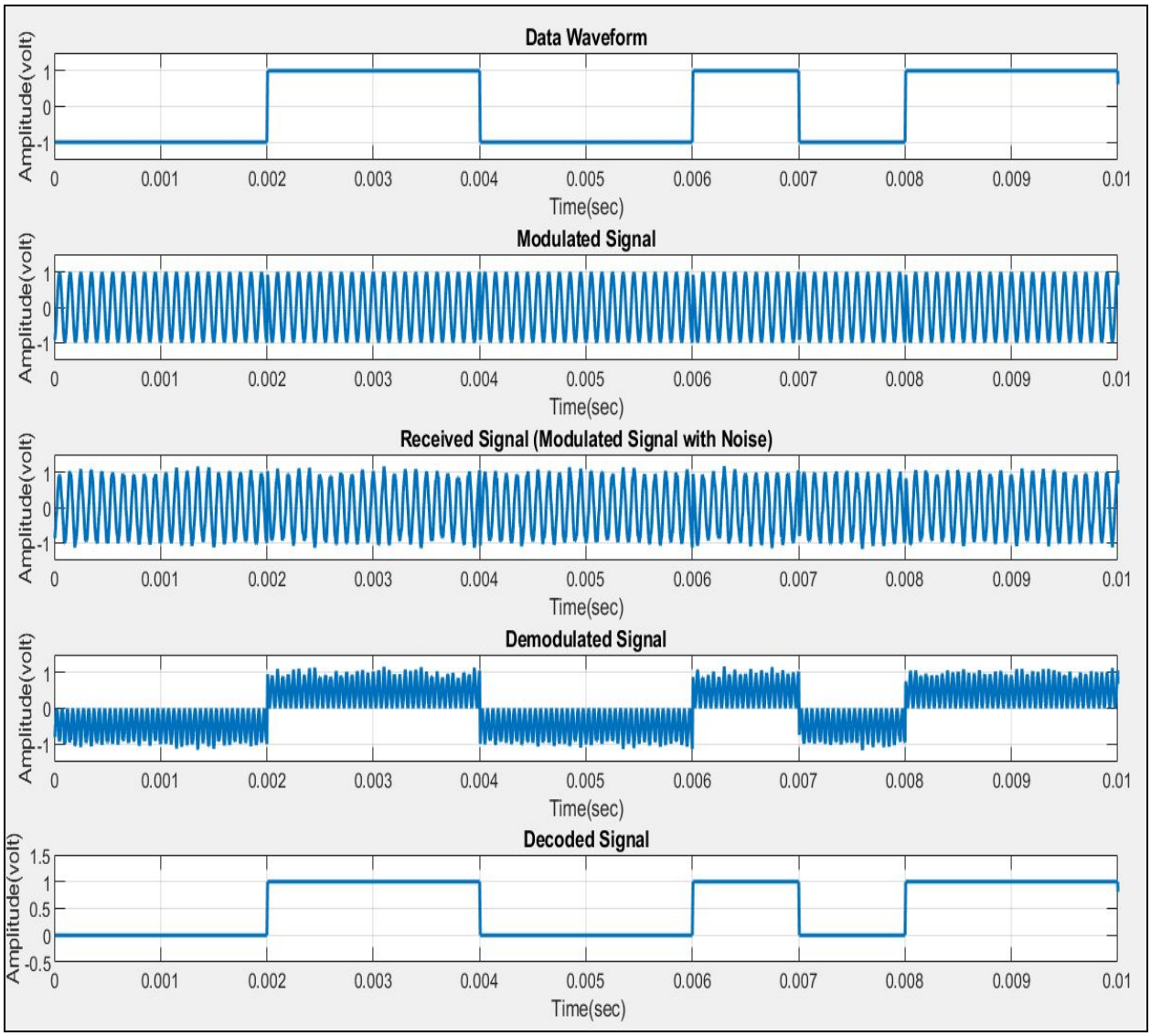
Step 2: After the multiplication with 2 times the carrier signal, the signal is filtered with a 6th order low pass filter. Then thresholding is applied to determine if a '1' was sent (+ve voltage) or a '0' was sent (-ve voltage).

```
bpsk_int_signal = bpsk_mod_noise_signal.*(2*carrier);
[b,a] = butter(6,0.2);
bpsk_demod_signal = filtfilt(b,a,bpsk_int_signal);
```

End to End Simulation

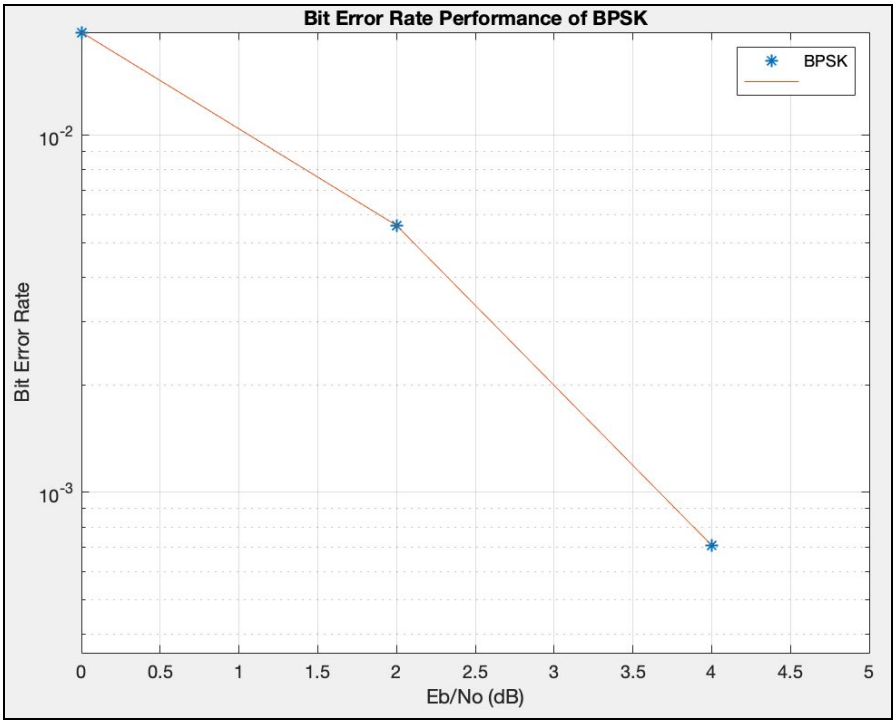
The complete waveform simulation for the end-to-end transmission of information using BPSK modulation is shown in the figure below.

1. The 1st plot shows the generation of random message bits.
2. The 2nd plot shows the carrier signal $cos\ (2\pi ft)$
3. The 3rd plot shows the modulated BPSK signal
4. The 4th plot shows the addition of AWGN noise according to the chosen signal-to-noise ratio
5. The 5th plot shows the BPSK demodulation of the noisy signal using a coherent receiver.



Bit Error Rate Performance

The graph below shows the bit-error rate performance of BPSK across different SNR values.



3. Binary Frequency Shift Keying

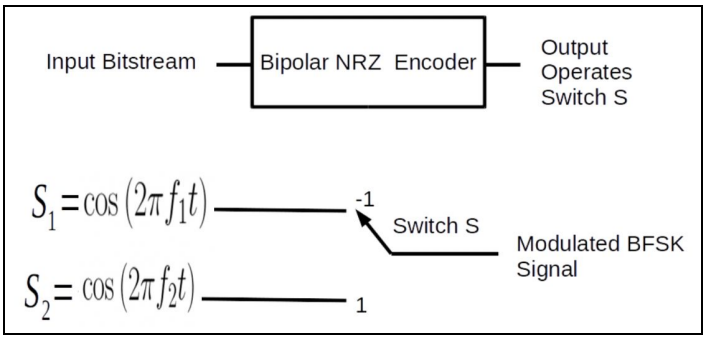
Binary Frequency Shift Keying (BFSK) is a digital modulation technique in which we transmit one bit per symbol '0' or a '1'. Hence, the bit rate and symbol rate are the same. In BFSK, the information is encoded in the variation of the frequency of the carrier. '0' is represented by carrier frequency 'f_1' and '1' is represented by carrier frequency 'f_2'. Hence,

$$S_1 = \sqrt{\frac{2E}{T}} \cos(2\pi f_1 t) \rightarrow represents\ '0'$$

$$S_2 = \sqrt{\frac{2E}{T}} \cos(2\pi f_2 t) \rightarrow represents\ '1'$$

Modulation

For BFSK modulation, the data stream needs to be encoded into bipolar NRZ signals. The NRZ waveform is generated by over sampling these impulses. The NRZ encoder output then operates a switch. If encoder's output is +ve, switch sends S1 and if -ve, the switch S2. The logic is demonstrated in the figure below.



Step 1: We first defined the 2 carrier frequencies and the 2 carrier signals as shown below. The second carrier frequency to be half the first.

```
% Carrier Wave
f1 = Fc;
f2 = Fc/2;
t2 = bp/signal_length:bp/signal_length:bp;
carrier1 = cos(2*pi*f1*t2);
carrier2 = cos(2*pi*f2*t2);
```

Step 2: In this code, we did not perform the NRZ encoding as we used an 'if-else' function to match the input to the correct carrier signal. The data samples are modulated with the 2 carrier signals as shown below. If input bit = 1, the modulated signal is the 1st carrier signal. If input bit = 0, the modulates signal is the 2nd carrier signal.

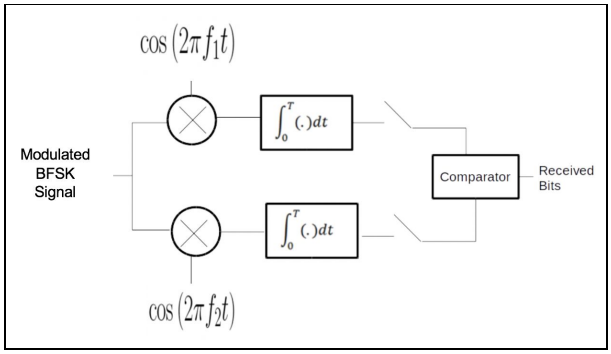
```
% Modulation
bfsk_mod_signal=[];
for (i=1:1:length(bfsk_orig))
    if (bfsk_orig(i)==1)
        y = carrier1;
    else
        y = carrier2;
    end
    bfsk_mod_signal=[bfsk_mod_signal y];
end
```

Step 3: Additive white noise is added with required SNR value to both the modulated signals. The received signal is modulated signal added with the noise using the same function used in OOK above.

```
% Add Noise
bfsk_noise_signal = add_noise(bfsk_mod_signal,1,snr);
```

Demodulation

The demodulation logic for BFSK is shown below.



Step 1: Coherent demodulation is performed here by multiplying the modulated signal with the carrier signal of the same frequency and phase as at the transmitter in 2 branches.

Step 2: At every branch, the result is integrated over the bit period 'bp' and sampled.

If S_1 is transmitted, the output of the branch where the first carrier signal is being multiplied is higher since input signal frequency equals carrier frequency 1 (10KHz). At the same time, the second branch the frequency of S_1 will be orthogonal to the frequency of carrier signal 2. The output of these two branches are compared to decide if S_1 or S_2 is the input.

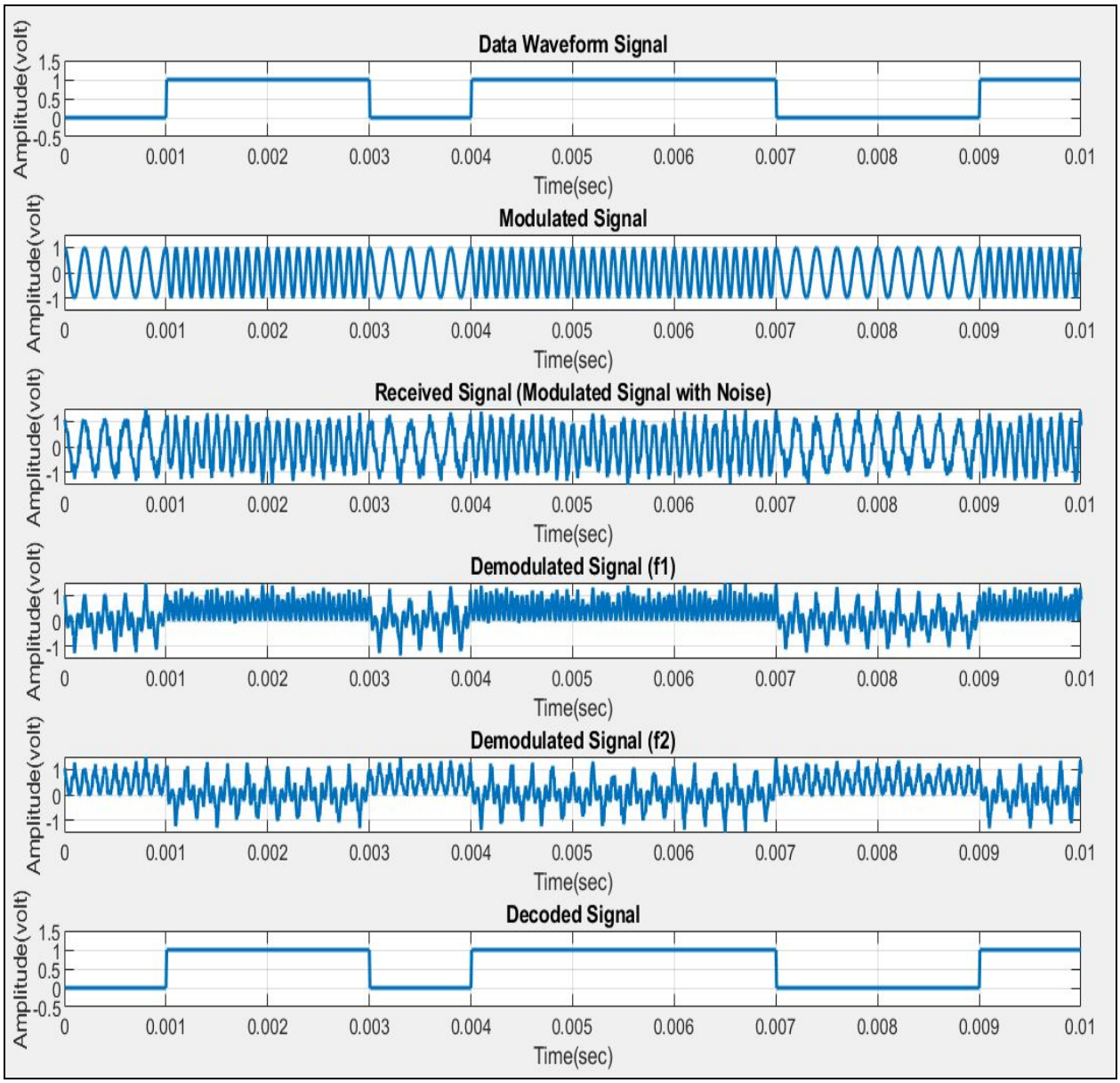
The figure below shows the implemented code.

```
% Demodulation
y1 = bfsk_noise_signal.*cos(2*pi*f1*t1);
y2 = bfsk_noise_signal.*cos(2*pi*f2*t1);
int1_bfsk = [];
for i = 0:signal_length:length(y1)-signal_length
    int_ = trapz(y1(i+(signal_length/2):i+(signal_length/2)+1));
    int1_bfsk = [int1_bfsk int_];
end
int2_bfsk = [];
for i = 0:signal_length:length(y2)-signal_length
    int_ = trapz(y2(i+(signal_length/2):i+(signal_length/2)+1));
    int2_bfsk = [int2_bfsk int_];
end
int = int1_bfsk - int2_bfsk;

bfsk_demod_signal = signal_threshold(int, 0);

bfsk_error_rate = compute_error(bfsk_orig, bfsk_demod_signal);

bfsk_demod_signal = replicate(bfsk_demod_signal, signal_length);
```

End to End Simulation

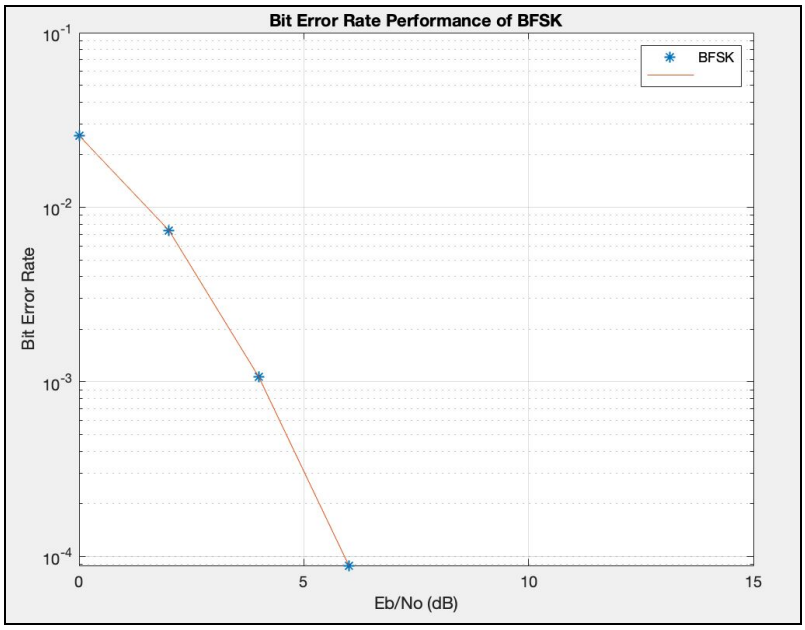
The complete waveform simulation for the end-to-end transmission of information using BPSK modulation is shown in the figure below.

1. The 1st plot shows the generation of random message bits.
2. The 2nd plot shows the modulated BFSK signal
3. The 3rd plot shows the addition of AWGN noise
4. The 4th and 5th plots BFSK demodulation of the noisy signal using a coherent receiver for the 2 different carrier frequencies. The 4th plot is for f1 = 10KHz and the 5th plot is for f2 = 5KHz
5. The 6th plot shows the decoded signal from BFSK

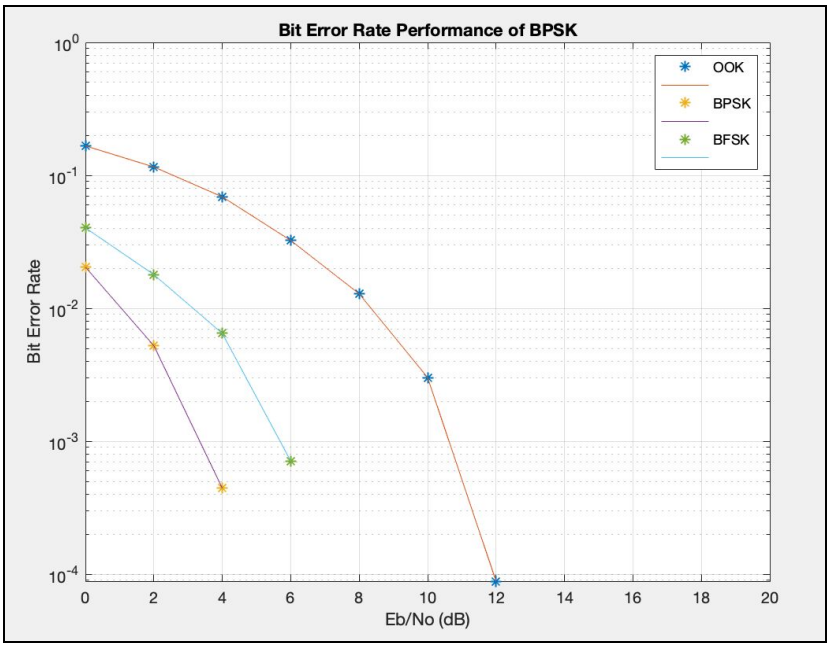


Bit Error Rate Performance

The graph below shows the bit-error rate performance of BFSK modulation.



The graph below compares the bit-error rate performance of BFSK, BPSK, and OOK. As shown in the graph, BPSK is the most power efficient among the three modulation schemes. On the other hand, OOK, which is a form of ASK, has the worst power efficiency among the three schemes but is the most bandwidth efficient as it modulates by changing amplitudes.



Phase 3 : Error Control Coding

In this section, basic error control coding will be used to improve the performance of the modulation schemes. The bit-error rate performance of OOK modulation will be analysed before and after applying Hamming codes as its error control coding scheme.

Hamming codes

The Hamming code has the following properties:

$$
\begin{gathered}
Hamming(7,4) \\
Message\ bits\ (k) = 4 \\
Codeword\ Length\ (n) = 7 \\
Parity\ Bits\ (n-k) = 3 \\
Code\ Rate\ (k/n) = 4/7
\end{gathered}
$$

The goal of the Hamming codes is to create a set of parity bits that overlap so that a single-bit error in a data bit or a parity bit can be detected and corrected. Hamming codes can detect up to 2 bit errors and correct 1 bit error with minimum distance of 3 bits between codewords.

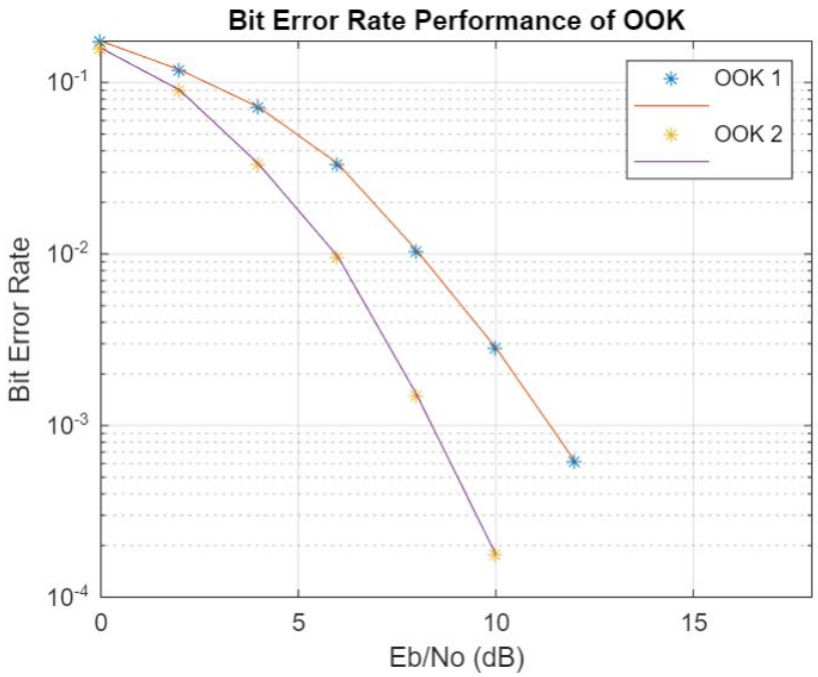
The generated random binary baseband data will be encoded before modulation and decoded after modulation with the following functions:

$$
\begin{gathered}
encode(Data, 7, 4, hamming/fmt); \\
decode(Data, 7, 4, hamming/fmt);
\end{gathered}
$$

Internally the encoding and decoding function in matlab generate a corresponding G (code generator matrix) and H (parity-check matrix) to identify and correct errors in the received signal.

Power efficiency

The power efficiency of the OOK modulation improved from *OOK1* to *OOK2* after applying Hamming codes as shown in the graph below. *OOK2* would require lesser power (SNR) to produce the same performance as *OOK1*



Bandwidth efficiency

The improved power efficiency of *OOK2* is at the cost of lower bandwidth efficiency. Hamming codes used in *OOK2* encodes 4 bits of data into 7 bits by adding 3 parity bits. This increases the total message length from 1024 bits to 1792 bits. This results in a decrease in data rate. If *OOK1* has a data rate of 1000bps then *OOK2* only has a data rate of 571bps (data rate here refers to the number of message bits per second).