The University of Kansas

Technical Report

Development of an Experimental Testbed for Evaluation of Covert Communications

Jonathan Owen, Victor Frost
Ghaith Shabsigh

ITTC-FY2015-TR-71328-03

July 2014

Project Sponsor:
National Science Foundation

Table of Contents

I. Abstract

The desire to hide the transmission of information has existed since antiquity. This has included the need to conceal the very existence of transmissions; exposing the presence of transmissions may reveal the location of the sender or an increase in the frequency of transmissions may indicate the impending occurrence of an event. The overarching goal of this project is to create new technologies to hide the transmission of information within the RF environment created by packet-based broadband wireless (infrastructure) networks, like LTE. Packet-based broadband wireless (infrastructure) systems use Orthogonal Frequency Division Multiplexing (OFDM) along with protocols that adapt to the environment, e.g., adaptive modulation and coding (AMC), Hybrid ARQ (HARQ), and opportunistic scheduling. The potential for covert communications arises from adaptive nature of AMC, HARQ, and opportunistic scheduling mechanisms used in infrastructure networks.

This paper describes techniques that have been implemented for experimental bit-error analysis of covert communications systems using the MXG Vector Signal Generator (N5182B) and the PXA Signal Analyzer (N9030A). This will be useful to a reader that has at least a Signals and Systems Analysis background. There will be a review of the communications theory needed to understand the work that is done to date. The reader can skip this section if the theory is already known. A description will then be given on how to model the complex dual system (Target and Covert) scenario using the Signal Generator (SG) and Signal Analyzer (SA). The end goal of this paper is to explain the steps used to create waveforms on the SG, read them back in on the SA, and analyze the data using Matlab. The locations of important files and figures will be noted clearly at the end of the paper. All of these files will be under the *Covert_Comm* network in the folder labeled *Equipment_Measurements*. These files can also be found within the index at the end of this report.

II. Review of Communications Theory

The binary logic bits are 0 and 1. Using bipolar NRZ line coding, the 0 bit translates to a voltage located at -1 V and the 1 bit is a voltage at +1V. An entire set of bits is called a single **sequence** of bits. Each bit is transmitted as one **symbol** of data at +1V or -1V. Each symbol is composed of a constant number of **samples**.

For a practice calculation, consider the sequence shown in Figure 1.
The bit sequence would be 01101101010010110010 (length of 20 bits). Therefore, there are $N = 20$ symbols to transmit. For this example, there are $k = 500$ samples that compose each symbol. By cancellation of units, it is determined that there are $N * k = 10000$ samples along the x-axis of Figure 1. Note that these values of N and k apply to this example only, although the same unit cancellation can be applied to digital sequences with different values of N and k.

The equation below shows the unit calculation:

$$N\ symbols * k\ \frac{samples}{symbol} = N * k\ \ samples\ along\ the\ x-axis\ in\ Figure\ 1.$$

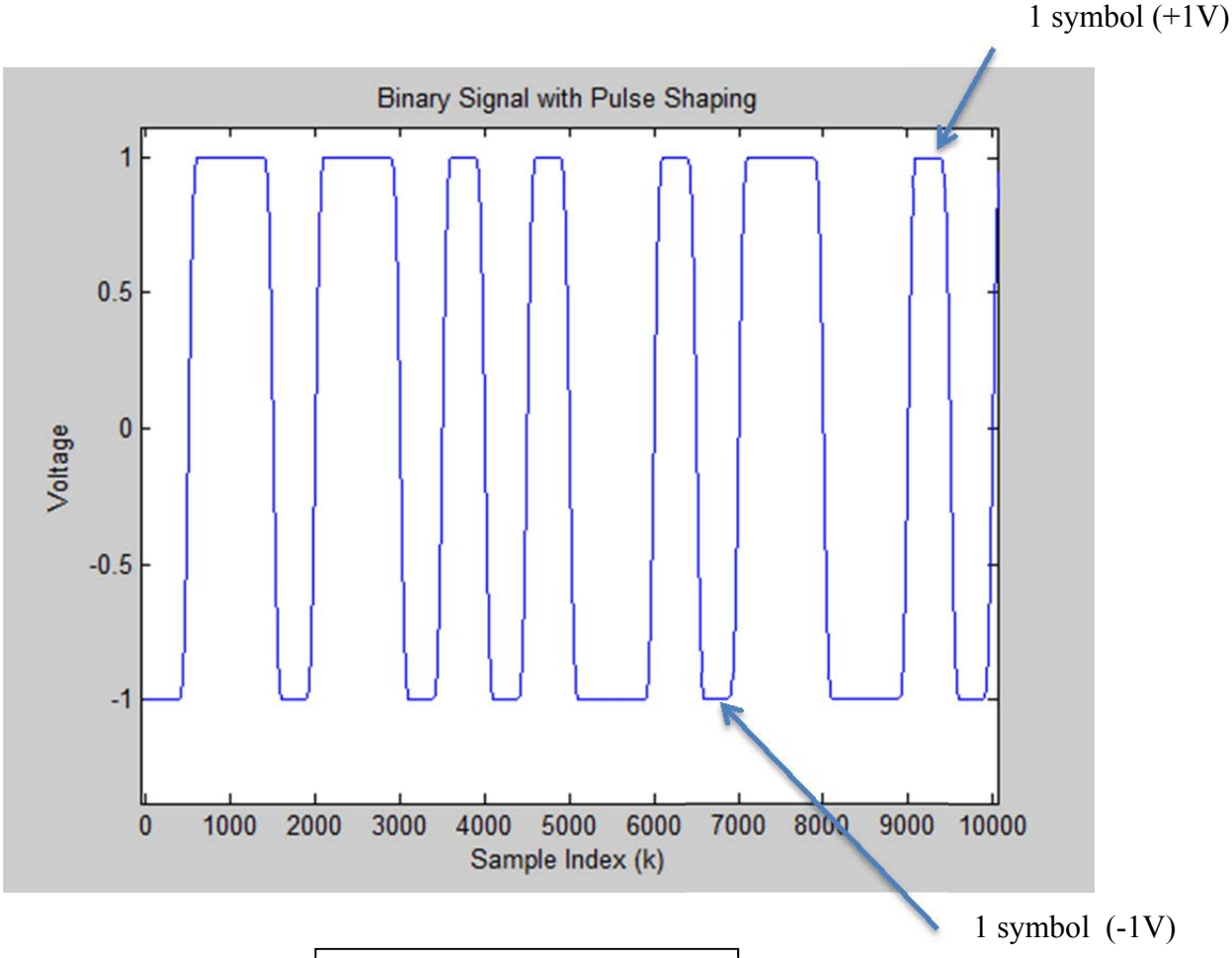


Figure 1. Sequence of Bits.

The power spectral density (PSD) of the signal in Figure 1 is found using FFTs and Bartlett's windowing method in Matlab; this would approximately represent a sinc^2 function in the frequency domain as shown in Figure 2.

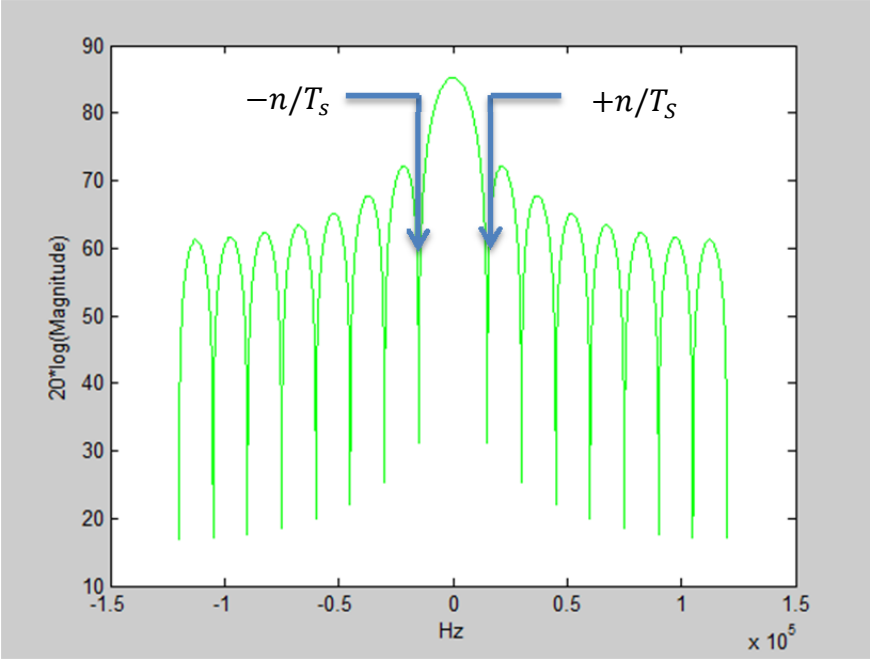


Figure 2. FFT of Sequence of Bits

There are a few properties to take note of with respect to the PSD. This sinc^2 function has magnitudes of zero in the frequency domain at frequencies $-n\ /\ T_S$ and $+n\ /\ T_S\ Hz$, where n is any integer and T_S is the time required to transmit one symbol. In Figure 2, notice that the first magnitude with value zero to the right of the center frequency is located at $f = 15000$ Hz. Therefore, $1/T_S = 15000$ Hz. Then, T_S is calculated to be 1/15000 seconds. One symbol from Figure 1 takes T_S = 1/15000 seconds to transmit in the time domain. If one symbol takes 1/15000 seconds to transmit, then the amount of time required to transmit N symbols in the time domain is represented by N symbols $* T_S \ \frac{\text{seconds}}{\text{symbol}}$.

Orthogonal Frequency Division Multiplexing (OFDM) is a method of encoding digital data on multiple carrier frequencies. All sub-carriers within an OFDM system are orthogonal to each other and are separated in frequency by $1/T_S$ Hz, as shown in Figure 3 where $T_S = 1 \,/\, 15000$ seconds.

A **sub-carrier** is created when a baseband signal is frequency shifted to create a new signal of higher frequency. Note that a sub-carrier is created prior to being modulated by the main carrier frequency. Each of the signals shown in Figure 3 are operating on sub-channels. The process of frequency shifting is explained on the next page.

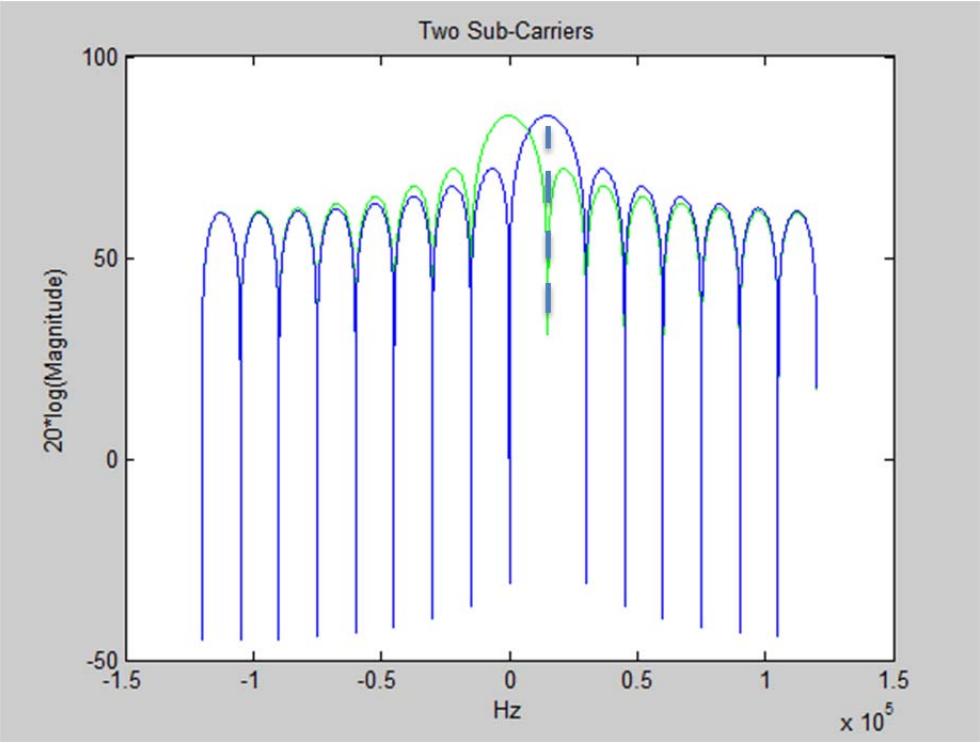


Figure 3. Two Signals with OFDM Characteristics

Consider a complex baseband signal $x(t)$. If it is desired that $x(t)$ be centered at a frequency of 15 kHz, then the scalar components of the signal are multipled by $e^{j2\pi ft}$. Therefore, the blue sub-carrier in Figure 3 can be represented as:

$$x(t) * e^{j2\pi ft}$$

When the center frequency of the original signal $x(t)$ is changed, the real and imaginary components of $x(t)$ are affected. Figure 4 demonstrates (a) how the real part of the signal appears at baseband, and (b) how the real component of the signal appears when frequency shifted by $e^{j2\pi ft}$.

At the receiver, the signal is demodulated to recover the symbols originally transmitted. This is accomplished by multiplying the received signal by $e^{-j2\pi ft}$. The full frequency shifting process can be seen mathematically as:

$$x(t) * [\mathrm{e}^{2\pi ft} * \mathrm{e}^{-2\pi ft}] = x(t) * [\,1\,] = x(t)$$

(a)

(b)

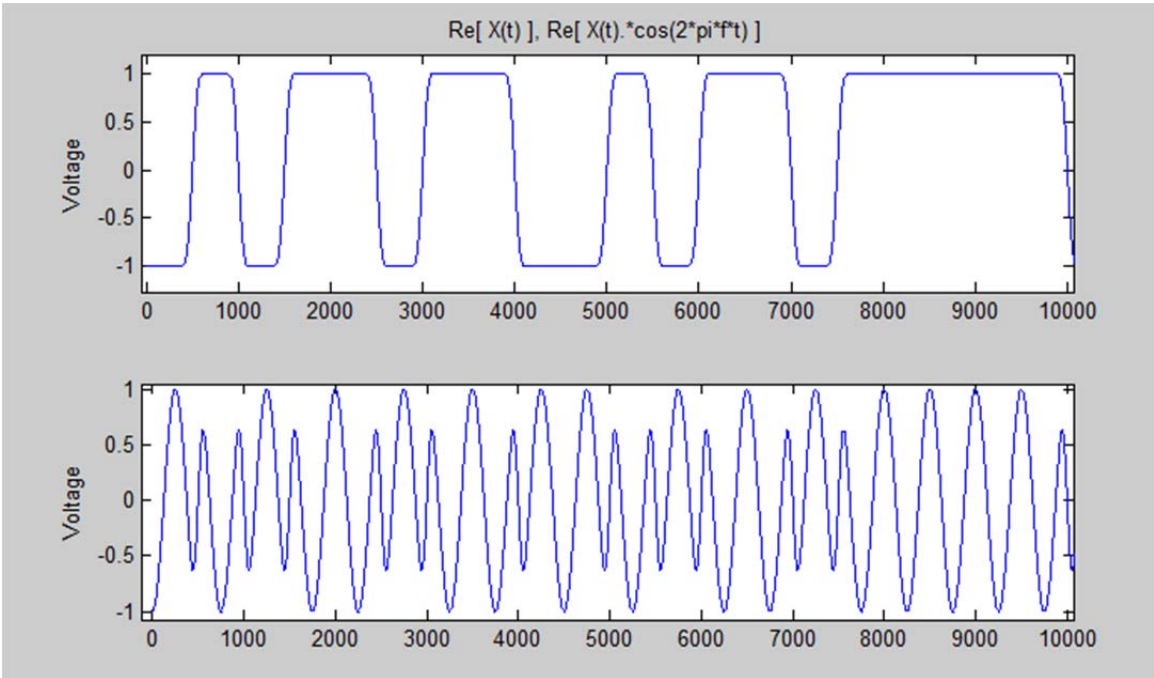


Figure 4. Frequency Shifted Signal $x(t)$

As a side note, notice that the waveforms in Figure 1 and 4 have smooth edge transitions between every symbol as a result of pulse shaping. **Pulse shaping** is the process of passing a waveform through a filter to create a new waveform. Recall that signal filtering in the time domain is represented as the convolution between an input signal and the filtering signal.

$$y(t) = x(t) \otimes h(t)$$

Pulse shaping of the emulated transmission signal was accomplished using a raised-cosine filter, which attenuates the high frequencies from a waveform. Raised-cosine filtering is typically used when transmission is done over a band-limited channel.

To recover the originally transmitted bits from the received signal, the signal is put through a matched filter. The original bits are then recovered by taking samples at the peaks of the match-filtered signal every symbol time T_S . A matched filter can be modeled as an integrate-and-dump function in Matlab.

Figure 5 demonstrates a 16QAM signal transmitted over a band-limited channel without pulse shaping. Notice that there is a spike of distortion between each symbol due to band-limiting of the channel. This overshoot can cause an undesired bit error in some cases, but will not always significantly affect BER analysis results.

The coding process used to add pulse shaping to a signal is under the folder *Matlab_System_Model -> PulseShapingExample*.

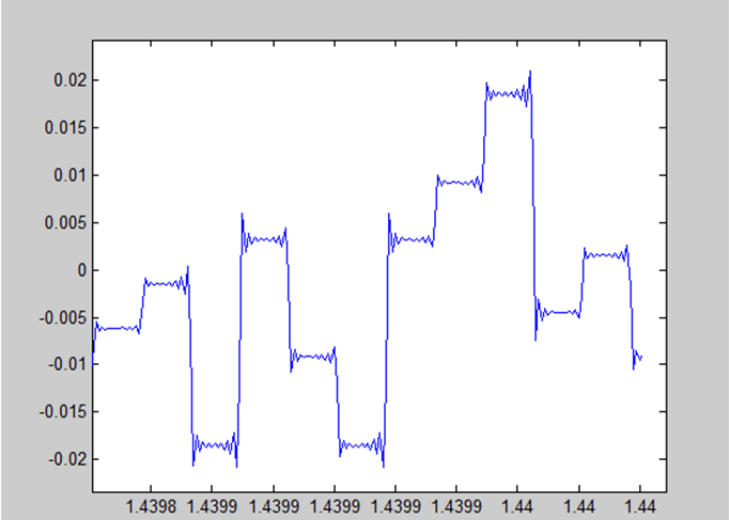


Figure 5 : Distortion Spikes Without Pulse Shaping

For the binary signal in Figure 1, a modulation technique called **phase shift keying (PSK)** is applied. PSK is a digital modulation technique that uses two phases to represent a 0 bit or 1 bit. The binary signal is referred to as a **BPSK** (binary phase shift keying) signal that has a phase of 0 for the "0" bit and π for the "1" bit. Remember that phase is represented as $e^{j\varphi}$. Hence, for BPSK:

$$e^{j(0)} = 1$$
$$e^{j(\pi)} = -1$$

A **constellation diagram** is a way of representing the real (called **in-phase**) and imaginary (called **quadrature-phase**) parts of a digitally modulated signal on the same plot. When mapped to a constellation, the real parts of a signal are measured along the x-axis and the imaginary parts are measured along the y-axis. Figure 6 shows the constellation diagram of a baseband BPSK signal.

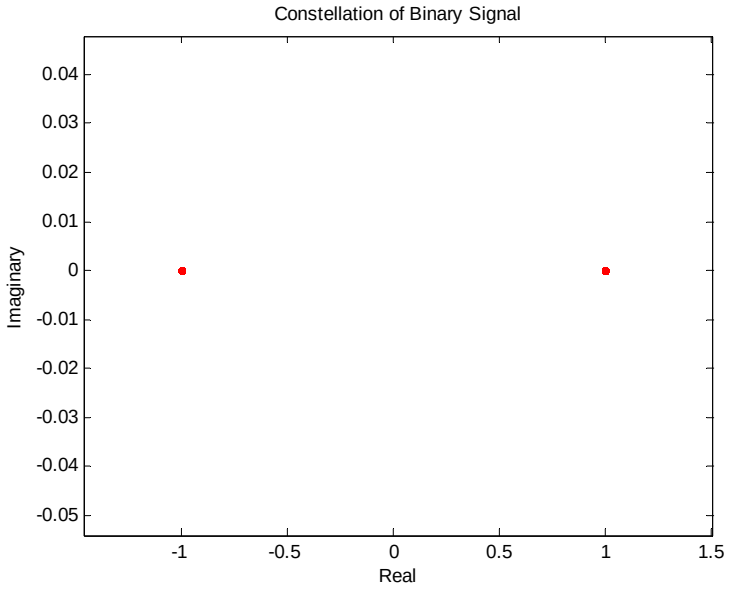


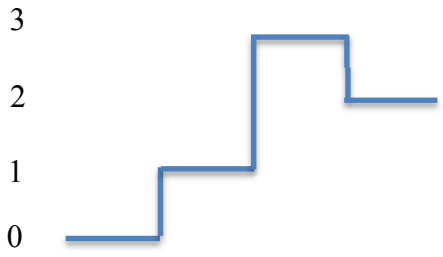
Figure 6. BPSK Constellation

The other signals used are **QPSK** (Quadrature Phase Shift Keying) and **16 QAM** (16 Quadrature Amplitude Modulation) signals. A binary signal transmits 1 bit per symbol, a QPSK signal transmits 2 bits per symbol, and a 16 QAM signal transmits 4 bits per symbol. Figure 7 demonstrates the constellation diagrams of each.

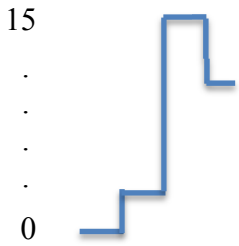
Notice that in QPSK modulation there are 4 integer levels, while 16 QAM has 16 integer levels. Every level represents some combination of real and imaginary numbers when mapped to a point on the constellation diagram. A table is shown below each of the constellations to represent one of these possible combinations for QPSK and 16 QAM.

Figure 7. BPSK, QPSK, 16 QAM.

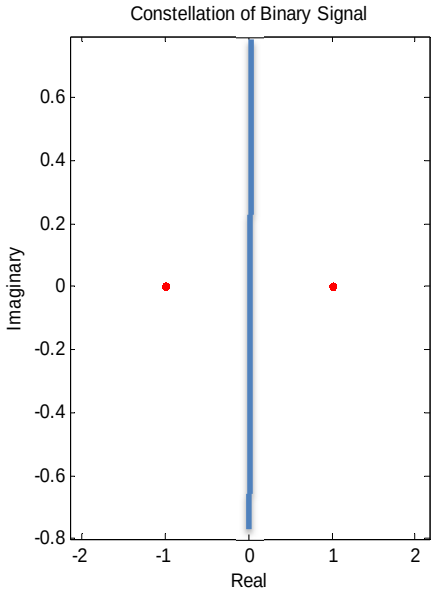
BPSK Symbol

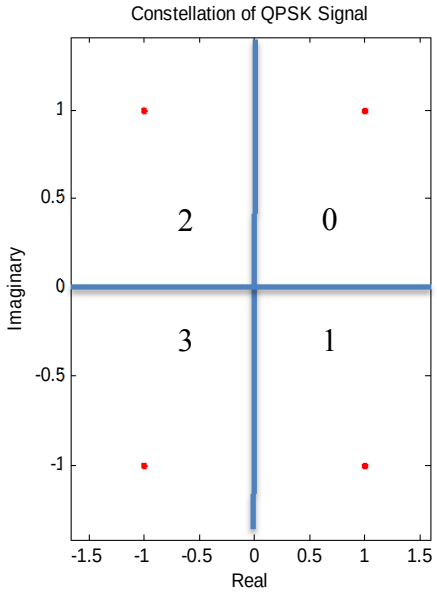


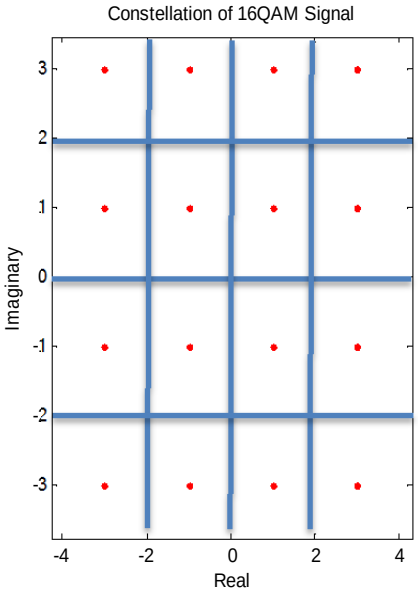
QPSK Symbol



16QAM Symbol







Symbol Value	Modulated
0	+1
1	-1

Symbol Value	Modulated
0	1+ j
1	1- j
2	-1 + j
3	-1 - j

Symbol Value	Modulated
0	3+3j
1	3+1j
2	3–1j
3	3–3j
4	1+3j
5	1+1j
6	1–1j
7	1–3j
8	-1+3j
9	-1+1j
10	-1–1j
11	-1–3j
12	-3+3j
13	-3+1j
14	-3–1j
15	-3–3j

In the two sub-carrier case from Figure 3, the center of the second sub-carrier is at 15000 Hz, which is the exact location of where the first sub-carrier has zero magnitude. Two signals are orthogonal when:

$$\int_{t_0}^{t_0+T} z_1(t)\, z_2(t)^* dt = 0$$

Let us define the following variables, where 'i' is an index variable.

$$z_1(t) = x_i(t)\cos(2\pi f_1 t)$$

$$z_2(t) = y_i(t)\cos(2\pi(f_1 + \Delta f_i)t)$$

The orthogonality equation becomes

$$z(t) = \int_{t_0}^{t_0+T} x_i(t)\, y_i(t)\cos(2\pi f_1 t)\cos(2\pi(f_1 + \Delta f_i)t)dt$$

Making the following assumptions, $z(t) = 0$.

- $\frac{T}{f_1}$ *must be equal to an integer*
- x_i *and* y_i *must be digital signals with symbol time* T
- $\Delta f_i = \frac{i}{T}$

Because each of these sub-carriers are orthogonal, the data from any of the sub-carriers can be fully recovered even though they are overlapping in frequency. This is one characteristic that makes using orthogonal sub-carriers useful. The spectrum in Figure 8 shows three sub-carriers in an OFDM system.

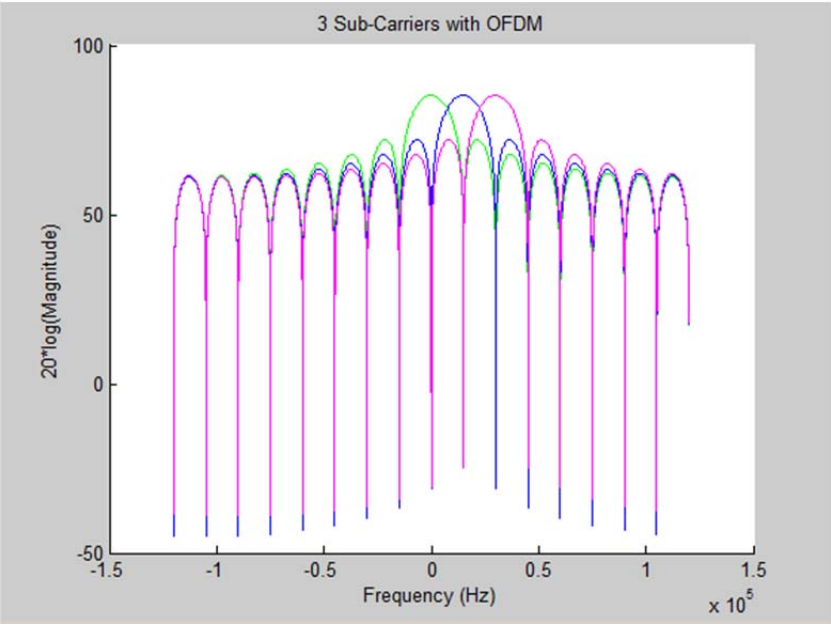


Figure 8. 3 Sub-Carriers with OFDM

To recover the data of a single sub-carrier in Figure 8, start from the beginning of the returned signal sequence. Note there are three signals overlapping each other in the time domain. Multiply the sequence by $e^{-j2\pi ft}$, where f is the frequency center of the sub-carrier being demodulated.
At the transmitter, rectangular pulse shaping was used. As previously discussed, the received signal can be passed through a matched filter to recover the transmitted bits. A matched filter can be modeled as an integrate-and-dump function with period T_S in Matlab.

Figure 9 demonstrates the combined signal of three sub-carriers with OFDM. The matched filter was modeled as an integrate-and-dump filter. The arrows in this figure represent the integration periods used for the integrate-and-dump filter model. The recovered signal is illustrated in Figure 10.

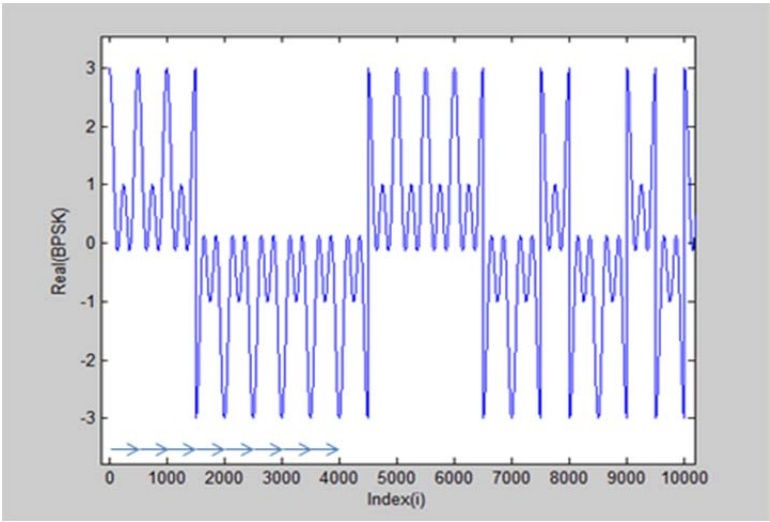


Figure 9. Signal After Frequency Shift.

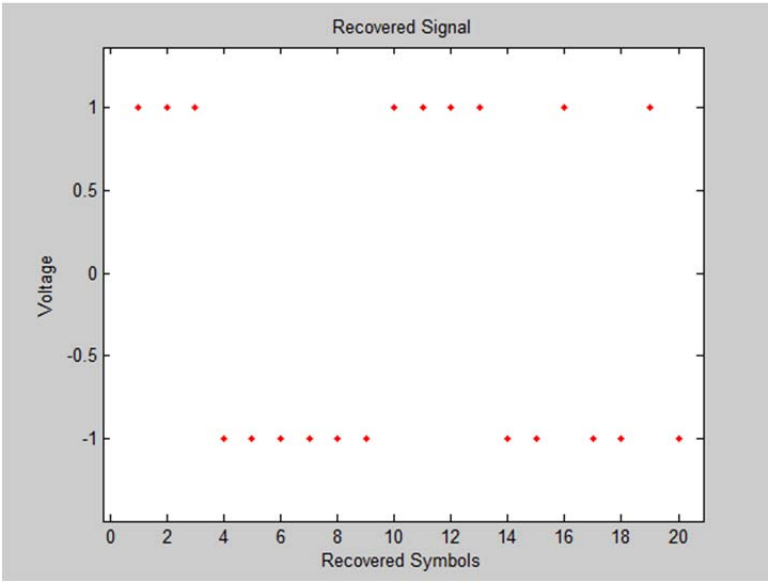


Figure 10. Recovered Signal.

In real-world communications the channel introduces **noise**. Noise will sometimes result in bit errors when recovering the original transmission. A measure of effectiveness of a signal with additive noise is called a **Signal-to-Noise Ratio (SNR).** A signal-to-noise ratio is defined as the ratio of the power contained by the system divided by the power contained by noise. SNR can be seen mathematically as:

$$SNR = S_{power}/N_{power}$$

A high SNR will result in fewer bit errors while a low SNR will result in more bit errors. For example, assume the noise power is constant. If the amount of power in the signal is increased, there is a smaller chance that the noise power will affect how well the signal is recovered. Figure 11 illustrates these signal and noise powers.

The noise will interfere with the passband signal. After demodulation, there is a likelihood that the bit will be incorrectly demodulated due to noise. The percentage of bit errors is called the **Bit Error Rate (BER).** Figure 12 displays a BPSK, QPSK, and 16QAM signal, each with equivalent SNR on a constellation diagram. If the additional noise power causes a bit to fall within the wrong quadrant of the constellation diagram, then that bit will be demodulated incorrectly. Notice that the 16QAM signal appears more susceptible to noise, but transmits more bits per symbol.

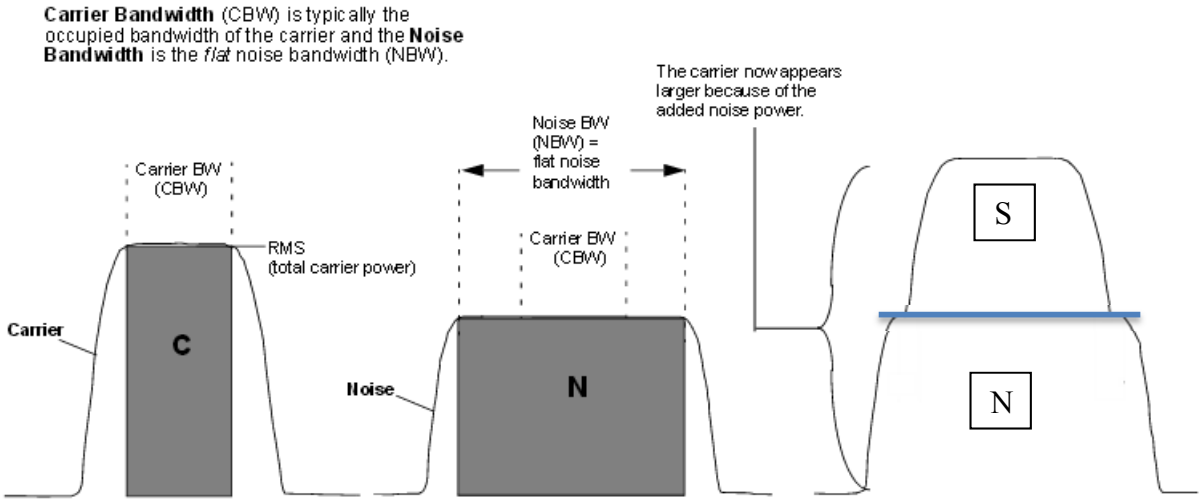


Figure 11. SNR Visual.

Modified From--
Agilent Technologies N5161A/62A/81A/82A/83A
MXG Signal Generators User's Guide
Figure 9-6
Page 256

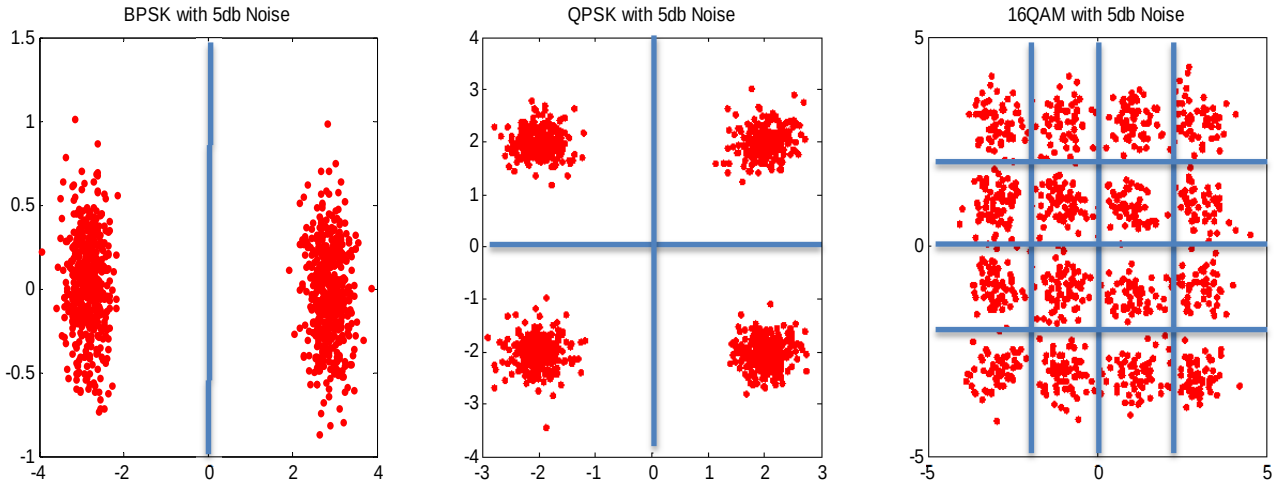


Figure 12. Constellations with 5 dB Noise

Being able to do basic SNR unit conversions is important. SNR is typically measured in decibels (dB). An SNR in decibels is mathematically represented as:

$$SNR\ (dB) = 10\log_{10}(\frac{S\ Watts}{N\ Watts})$$

For example, say that an SNR of 3 dB is desired for a signal, and it is known that there is 1 Watt of noise power. The amount of signal power required to establish this SNR is to be calculated . To find the required signal power, one must use the equation above:

$$10log_{10}\left(\frac{S\ Watts}{1\ Watt}\right) = 3\ dB$$

Given that an SNR of 3 dB is desired, the equation above can be solved for S.

$$S = 1\ Watt * 10^{(3/10)} = \sim 2\ W$$

Note what this implies while measuring power in dB: Every time the SNR is increased by 3dB, the signal power is doubled. For the example with a noise power of 1 Watt, this can be shown as:

$$10\log_{10}\left(\frac{2W}{1W}\right) = 3\ dB;\ \ 10\log_{10}\left(\frac{4W}{1W}\right) = 6\ dB;\ \ \ 10\log_{10}\left(\frac{8W}{1W}\right) = 9dB,\ \ \ldots.$$

Transmissions from unsynchronized systems on nearby frequencies also create interference. For experimentation, two systems are defined. The first system (**target** system) will consist of the three orthogonal 16-QAM sub-carriers. The second system (**covert** system) will transmit a covert signal. It is assumed that neither of these systems are aware of the other.

A covert signal has low power and is unsynchronized with the target system. For experimentation, the covert signal is modulated using QPSK and is centered at 45kHz in OFDM with the target system. Desynchronization results in interference between the target and covert system. A covert signal will appear the same as sub-carrier in the frequency domain, as is shown in Figure 13. The covert signal is shown in red. Note that the covert signal has less power than any of the target system sub-carriers.

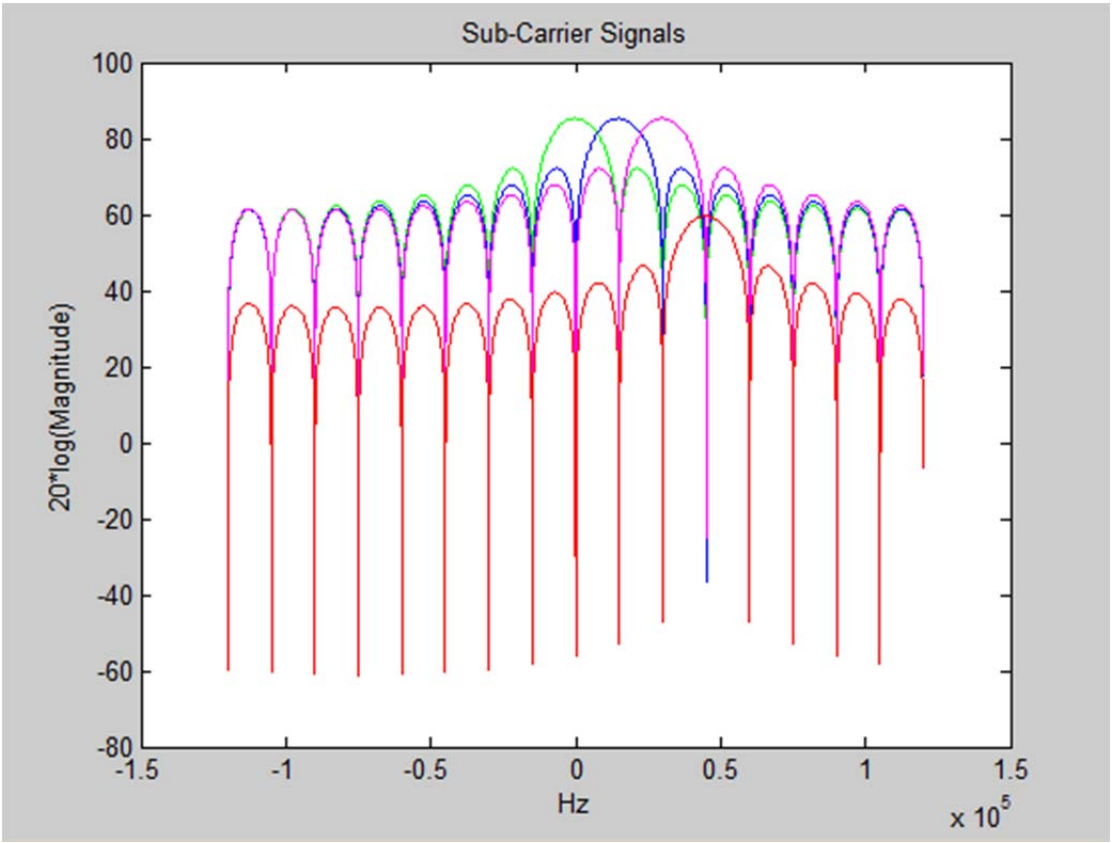


Figure 13. Target System and Covert System with OFDM

Rayleigh fading is now introduced to both systems. Rayleigh fading is the power attenuation that results from the propagation environment of a wireless signal. The power attenuation results from any objects which interfere with the signal transmission in free space such as buildings or walls. Rayleigh fading attenuation can be approximated according to a Rayleigh distribution. A Rayleigh distribution R($\sigma = 1$) is shown in Figure 14.

A Rayleigh distribution can be formed from a variable that has independent real and imaginary components that both follow a normal distribution N $(0, \sigma^2)$. The magnitude of this variable will be approximately Rayleigh-distributed.

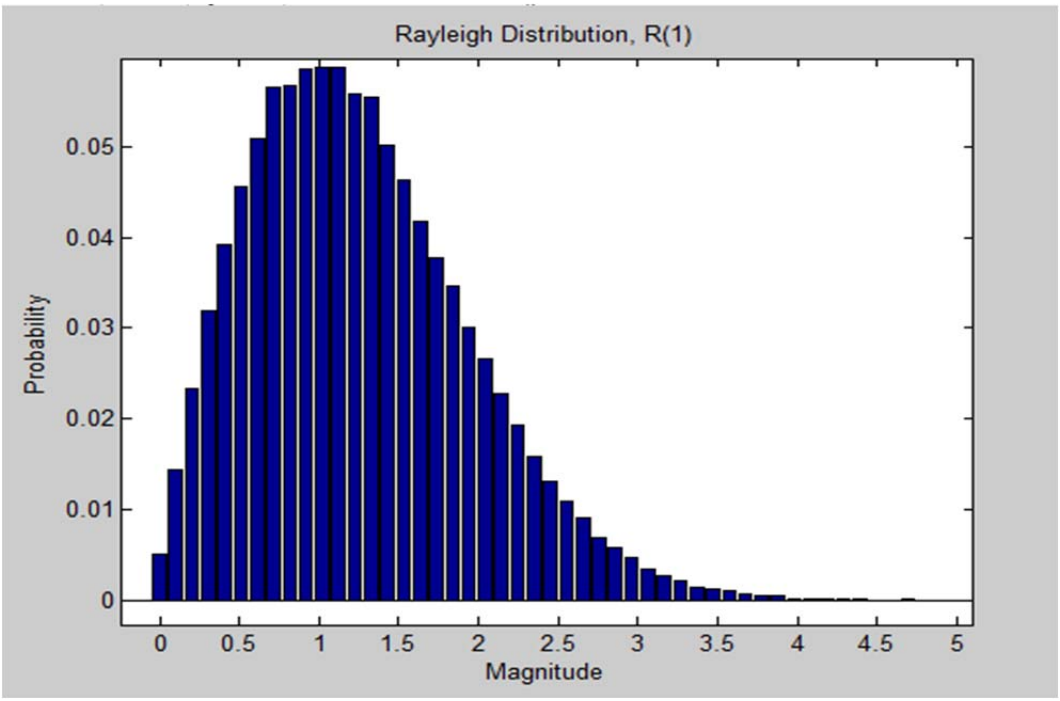


Figure 14. Rayleigh Distribution

III. Real-Time System Modeling

a. General Notes

This section will describe techniques used to correctly recover real-time signals using the signal generator (SG) and signal analyzer (SA). The reader is advised to go through the code in the folder *Matlab_System_Model* before reading this section. The directory is listed below. This code has all of the steps required to model the Target and Covert systems with Rayleigh fading using only Matlab (No signal is generated with hardware). There are detailed comments explaining each step. The logic used in this code is partially recycled in the machine analysis code.

The directory for the mentioned code is:
P:\covert_com\Equipment Measurement\Jon*Matlab_System_ModelCode***

b. Additional Theory

The following describes the process of recovering a signal from the target or covert system. During signal creation, a **reference signal** is added to the beginning or end of either the covert or target system. During signal analysis, a method is applied that uses the reference signal to find the beginning of the covert and target sequence. Finding the beginning of a sequence is necessary to analyze the BER of each system.
The reference signal is also used for resynchronizing phase. Phase desynchronization occurs when Matlab reads a signal from the SA.

The reference signal consists of 4 symbols of zeros, X symbols of the function $\cos(2\pi f \Delta t)$, followed by 4 more symbols of zeros. The reference signal length is X+8 symbols long. The larger the number of symbols X chosen for the cosine, the more likely that the beginning of the sequence will be determined correctly.
The reference signal is added to the beginning or end of the real portion of a signal. The imaginary portion of the signal must be equally long to avoid array length errors in Matlab; therefore, X+8 symbols of zeros are added to the beginning or end of the imaginary portion of each signal.

The time t for an X symbol cosine function to transmit should be X/T_S seconds. The cosine frequency was chosen to be 45kHz (the frequency center of the rightmost sub-carrier shown in Figure 13). If the target and covert systems are operating independently using two signal generators, only the system being analyzed for BER should have a reference signal in order to avoid complications during BER analysis.
A reference signal is shown in Figure 15.

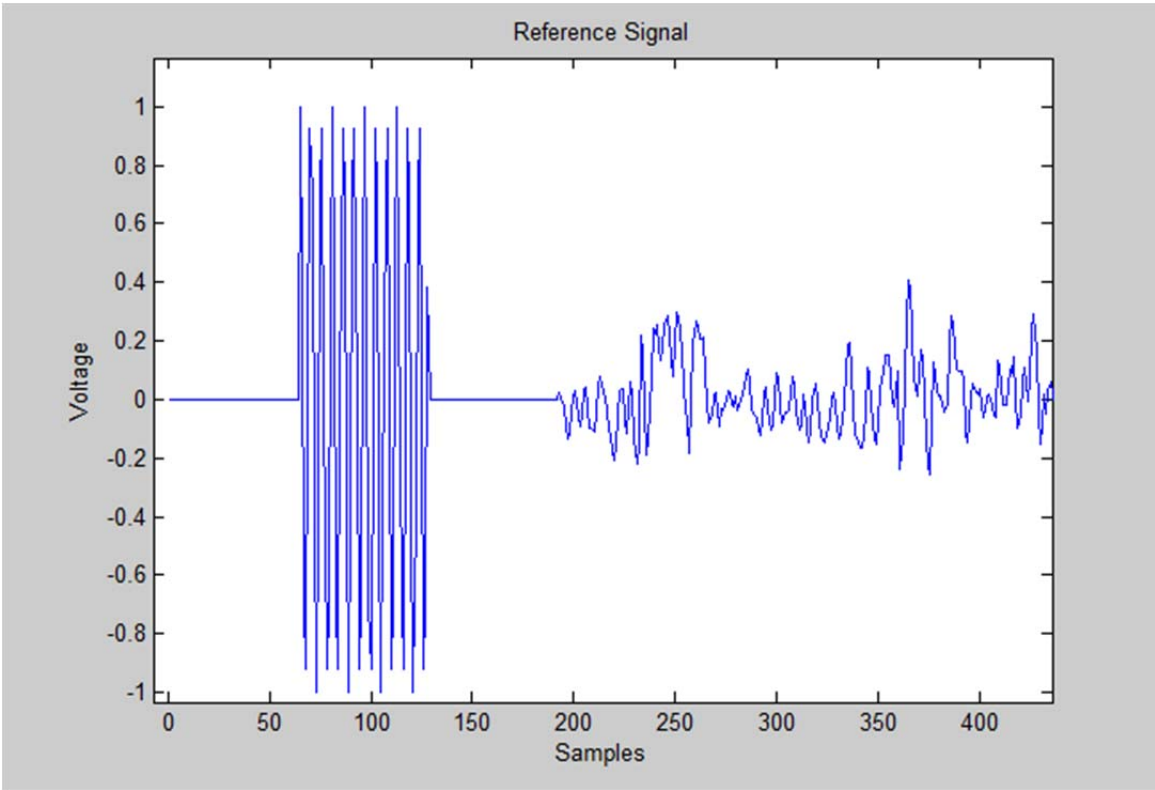


Figure 15 : 60 kHz Reference Signal
Cosine Length of 4 symbols

When trying to find the beginning of the covert or target signals, first find the correlation plot by convolving the entire read-in signal (which contains multiple sequences and reference signals in the time domain) with another exact copy of the reference signal, $\cos(2\pi f \Delta t)$. If the systems are correctly phase synchronized, a large spike in the convolution plot will occur at the indices of every reference signal as shown in Figure 16. The amount of unwanted phase present depends on when the signal is read from the SA using the Matlab code. Phase can be synchronized by multiplying the signal by a phase correction $e^{-j\phi}$. **Phase synchronization** occurs when the most maximum magnitude returns from the Matlab function $\max(\text{Correlation\_Signal})$. The user must find the correct value of ϕ within two decimal places, as there is only one correct value between 0 and 2π.

(This process will become clearer when the step by step process is described in the instrument practice section.) After phase resynchronization occurs, the beginning of the sequence can be found. The maximum correlation overlap occurs at the same array index where the reference signal reverts from $\cos(2\pi ft)$ back to 4 symbols of zeros.
The beginning of the sequence is located 4 symbols to the right of the max correlation index point.

In most cases, the index determined to be the beginning of the signal is correct. However, this reference signal approach is not exactly perfect. The index can be off by around ± 2 samples.

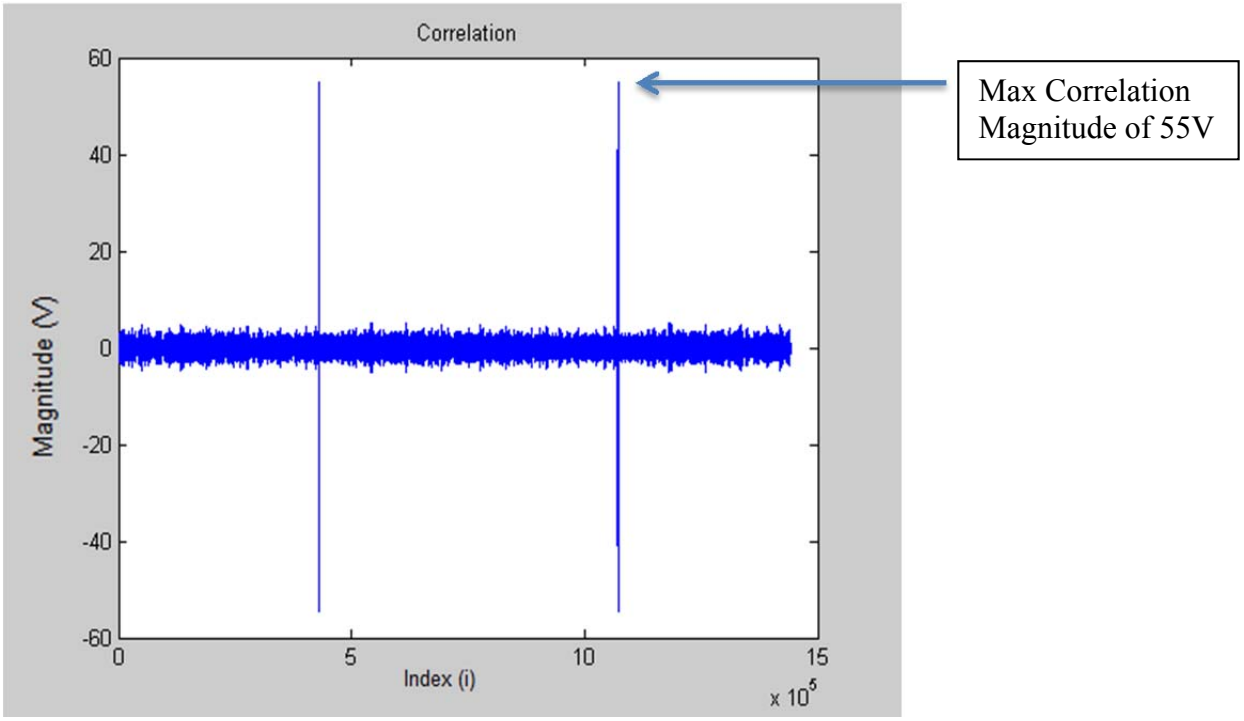


Figure 16 : 60 kHz Reference Signal

IV. One Signal Generator Practice Case – Target System Only

a. General Notes

In order to learn the general logic applied for using a single SG and SA, a simple case will be applied and examined. Only the Target system (without Rayleigh fading) will be created, downloaded, and received. This practice is for teaching the logic and calibrations required for instrument use only. The signal will not be analyzed to calculate a BER for this practice case.

b. Creating the Waveform

It is recommended that the reader implements the code along with this practice case in Matlab. The first step towards creating the Target system is to create the bit sequence of a single sub-carrier. Because 16 QAM is being utilized, uniformly distributed random integers between 0 and 15 are generated. Assume NumSym is 20000 symbols per sequence.

```
SubSeq = randi([0 15], 1, NumSym); % NumSym is the number of symbols in
                                   % one sequence. Each symbol is
                                   % represented by 1 sample integer
```

Once the integer sequence is created with 1 sample/symbol, it should be extended to have, for example, 16 samples/symbol. This can be done with the repmat( ) command. This code will return an array that has $320{,}000\,\frac{\text{samples}}{\text{sequence}} = 20{,}000\,\frac{\text{symbols}}{\text{sequence}} * 16\,\frac{\text{samples}}{\text{symbol}}$

```
Signal_a =  repmat(SubSeq(1,:),SamSym, 1); %Where SamSym is the desired
                                           %number of Samples/Symbol
```

Now that every symbol has been up-sampled, the entire sequence will be modulated to 16QAM using the qammod( ) function.

```
Signal_a = qammod(Signal_a, ModType);
                                    % Modulates the sequence to 16QAM
                                    % ModType = 2^(N bits/symbol)
                                    % ModType = 2^4 = 16 for 16QAM
                                    %
                                    % The sequence is stored. This
                                    % sequence represents the sub-
                                    % carrier centered at 0kHz in
                                    % frequency.
                                    % x(t)*e^(j*2*pi*0*t) = x(t)
```

One sub-carrier is now created. In the frequency domain this sub-carrier is already centered at 0kHz because $x(t) * e^{j2\pi(0)t} = x(t)$. The steps used to create the first sub-carrier must repeated to create the other two sub-carriers. The two new sub-carriers will need to be frequency shifted.

Two more sequences are created to represent the sub-carriers centered at 15kHz and 30kHz. The sequences are frequency shifted by $e^{j2\pi ft}$ to re-center the subcarriers.

```
t = linspace(0,NumSym/15000,length(signal_b));
% The time required to transmit NumSym symbols is NumSym/Ts.

SubSeq = randi([0 15], 1, NumSym);
Signal_b =  repmat(SubSeq(1,:),SamSym, 1);
Signal_b = qammod(Signal_b, ModType);
Signal_b = Signal_b .* exp(2j*pi*30000)*t);

SubSeq = randi([0 15], 1, NumSym);
Signal_c =  repmat(SubSeq(1,:),SamSym, 1);
Signal_c = qammod(Signal_c, ModType);
Signal_c = Signal_c .* exp(2j*pi*45000*t);

TotalSig = Signal_a + Signal_b + Signal_c;
% The transmission signal is created by combining the sequences of
% each sub-carrier. The power of TotalSig MUST still be normalized.
```

At this point, the signal for the Target system is created. The signal power has not been scaled down for safe use on the SG, and a reference signal must be added.
The following code will scale the total mean power of the signal to one. If power is not scaled appropriately, the SG could be damaged upon downloading the signal. To avoid this problem, a signal check-list is provided on the next page.

```
% The transmission signal must be normalized to have a power of 1.
% First, the mean power of the unscaled signal must be found.
% Mean power is ideally equal to |Mean Voltage|^2 in comms systems
UnscaledPow = mean(abs(TotalSig).^2);

% TotalSig (volts) is divided by the sqrt(UnscaledPow) (mean voltage)
% This normalizes the mean transmission power of the sequence to 1.
TotalSig = TotalSig./sqrt(UnscaledPow);
```

Now that the signal has normalized power, the reference signal is added to the real portion of TotalSig. Zeros are added to the imaginary portion of TotalSig to maintain equal array length for real and imaginary portions of the signal.

```
Ref = zeros(1,SamSym*12);
RefSig(SamSym*4:SamSym*8-1)=
                        cos(2*pi*60000*(linspace(0,4/15000,SamSym*4)));
% The reference signal is X = 12 symbols long.
% The cosine is X-8 = 4 symbols long. The frequency ‘f’ is 60kHz.

inphase = [real(TotalSig) RefSig];
quadrature = [imag(TotalSig) Ref];
TotalSig = inphase + 1i.*quadrature;

save(‘OFDMpractice’, ‘TotalSig’);
```

c. Downloading the Waveform

The transmission signal is now ready to be downloaded. Before downloading any signal, the signal must abide by the following checklist:

IMPORTANT: The following commands must return as true in Matlab before downloading the waveform onto either SG. This is to avoid any potential problems with a power overload, which can damage the SG.

1) mean(TotalSig) $\leq$ 0.05
2) mean(abs(TotalSig).^2) $\leq$ 1

If the above conditions are true for the signal, then copy and paste a saved . mat file of the transmission signal into the folder *DownloadAndRead.* Then open up the file *Download_WF_SG*.

The directory for the mentioned code is:
P:\covert_com\Equipment Measurements\Jon\Equipment_Measuring_Analysis\ OneSignalGenerator\DownloadAndRead\Download_WF_SG

<u>The following variables need to be changed within the file *DownloadAndRead* before running the code:</u>

load('File_name.mat') -- This should be changed to the . mat file copied in

IQData = *TotalSig;* -- The signal was saved under the variable *TotalSig*

addressMXG = '10.1.1.22' or '10.1.1.23';

-- These are the IP addresses of the signal generators. If '10.1.1.22' is used, the signal will be downloaded to the MXG signal generator. If '10.1.1.23' is used, the signal will be downloaded to the EXG signal generator.

outputPower = -20 ; -- The output power in dBm. For this example, -20 dBm is a safe output power.

sampleRate = 240e3; -- It is important to understand why this sample rate was chosen. This has to do with the number of samples per symbol chosen (16 samples/symbol) in relation to the desired one-sided bandwidth (15 kHz). If we multiply 15,000 symbols/second*16 samples/symbol, we obtain 240,000 samples/second. If a different number of samples per symbol is chosen, this should be recalculated. In all preset code described in future sections, this will remain 240e3 and should not be changed.

One of the two signal generators will have a black cable plugged in behind it, connecting the signal generator to the signal analyzer. This cable synchronizes either the MXG or EXG to the signal analyzer. Only one signal generator can be synchronized at a time. For explanation purposes, the MXG is assumed to be synchronized. Because the EXG frequency calibration is slightly offset while unsynchronized, the signal to be analyzed must be downloaded to the MXG signal generator. The cable is shown in Figure 17.

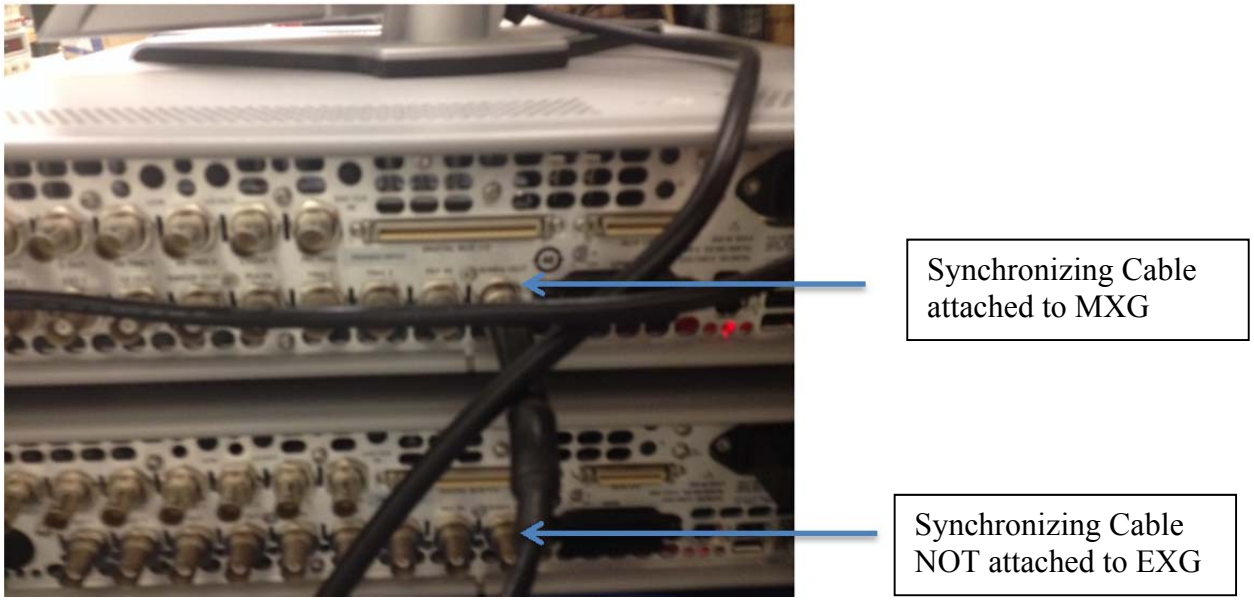


Figure 17: The Synchronizing Cable

The signal should be downloaded to '10.1.1.22' (MXG). The output power on the MXG should be set to -20 dBm (This is an acceptable output power for this example) and the center frequency should be 1 GHz. Look at the MXG and SA to double check that there are no warnings or errors. The power should never be above 0 dBm on the SA, as shown in Figure 20. Warning pictures appear in Figures 18 and 19.

(IMPORTANT) If there are any errors concerning power, turn <u>off</u> 'RF On/Off' for at least 30 seconds and determine the cause of the error.

Figure 18: PXA Errors Shown on Status Panel

Modified from --
Agilent Technologies
Instrument Messages
Agilent X-Series
Page 25

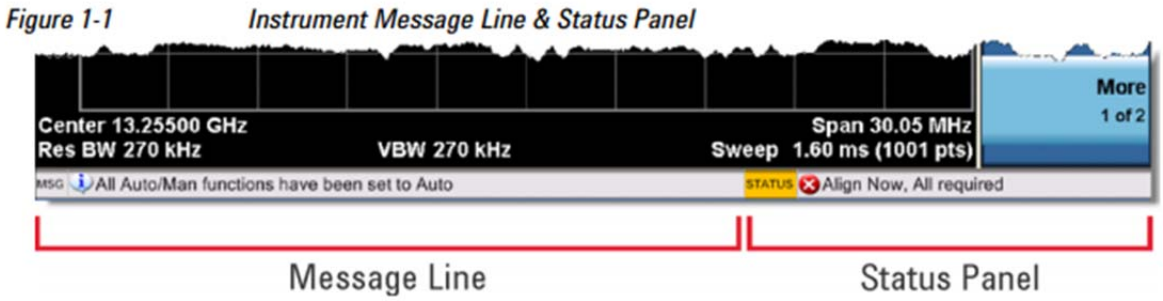


Figure 19: Error Message Area of MXG/PXG

Modified from --
Agilent Technologies
N5161A/62A/81A/82A/83A
MXG Signal Generators
User's Guide
Front Panel Display –
N5181A/82A/83A MXG

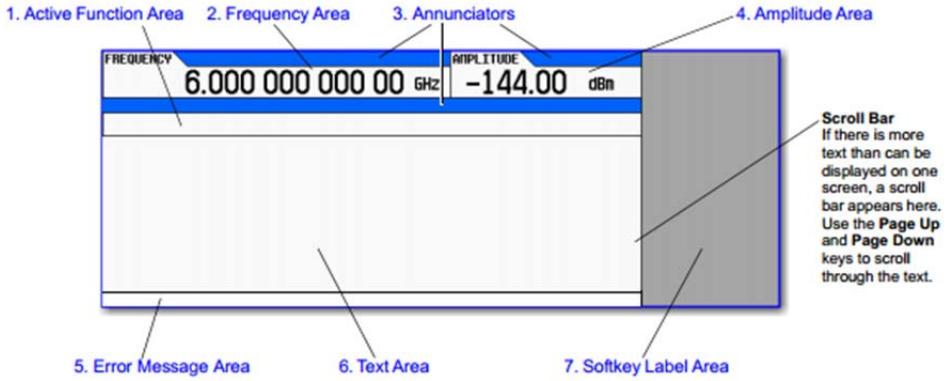
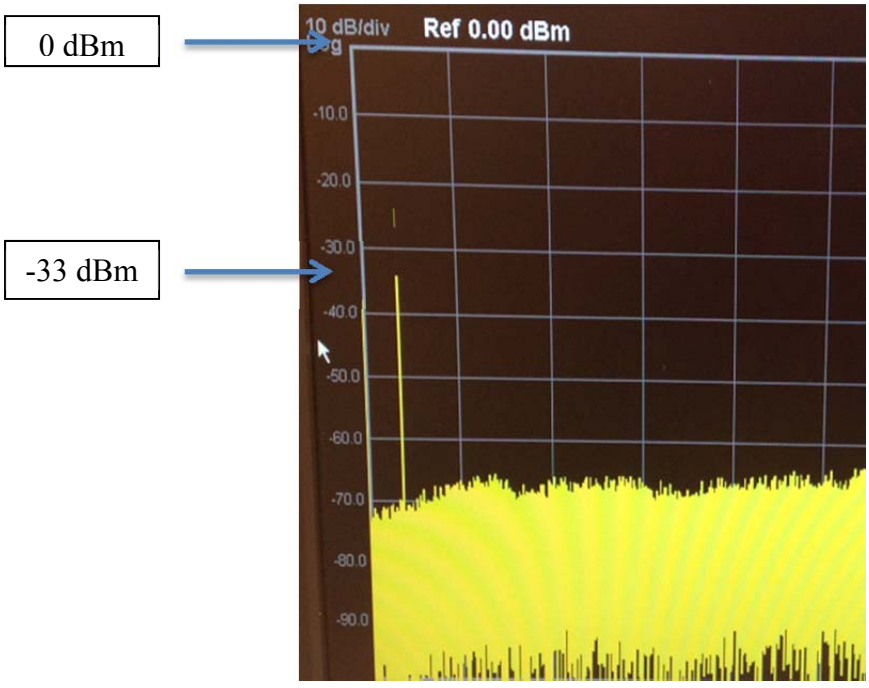


Figure 20: Typical Signal at -33 dBm (Safe Power Level)



d. Adding Noise to the Signal

Once the signal has been downloaded with the correct sample rate and power output, the MXG is able to add noise to the signal. First, on the SA, click:

FREQ → Auto Tune → SPAN → 500 kHz

This will cause the SA to show a picture of the Target system on the screen, as shown in Figure 21. The entire signal must be seen to add noise properly. In order to cut out the static noise and get a better picture, on the SA click:

Trace/Detector → Trace Average

This gives a better image of the signals appearance. This should be undone before adding noise to the system. To undo this, click:

Trace/Detector → Clear Write

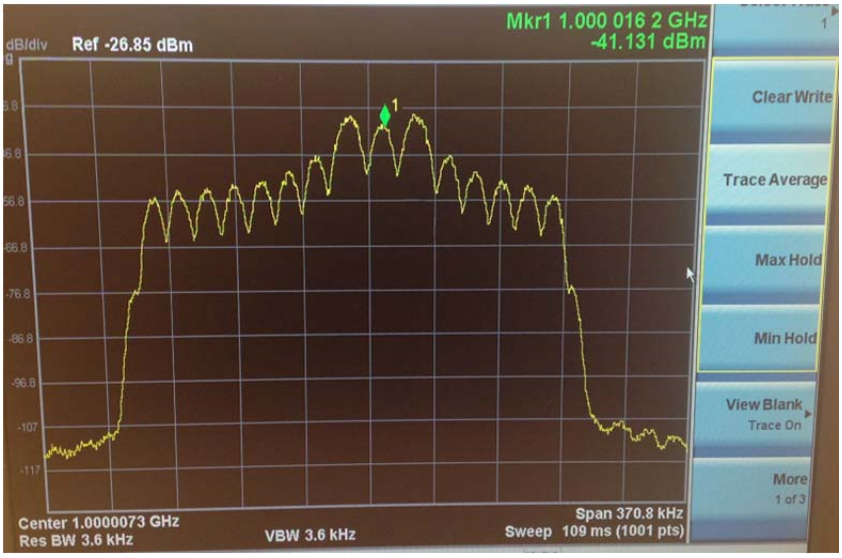


Figure 21 : Target system on the Signal Analyzer With Trace Average

Recommended: The RF output is best turned off before the ALC is turned off. The RF output is best turned off before changing noise settings. If RF is on, the SA could accidentally be harmed from a power spike without the user noticing.

Before adding Noise on the MXG, the Auto-Leveling Control (ALC) must be turned off. The ALC can be turned off on the MXG by clicking:

Cancel -> Turn off 'RF On/Off' -> AMPTD -> Turn off 'ALC'.

This turns off the waveform output to the SA, and then turns off the Auto Level Control. If the ALC is on, it will result in bad readings.

Now that there is a good image of the signal on the SA, noise can be added. To get to the AWGN main screen, on the MXG click:

Mode -> Dual Arb ->Arb Setup -> Real-Time AWGN Setup

Double check that the setting *Real-Time AWGN* is turned off. Adding noise to the signal means adding power to the signal. It is very important that the settings are adjusted correctly. Always be particularly careful when setting low Carrier-to-Noise-Ratios. These ratios typically are 10 dB or less, depending on the settings. The AWGN Setup screen is shown in Figure 22.

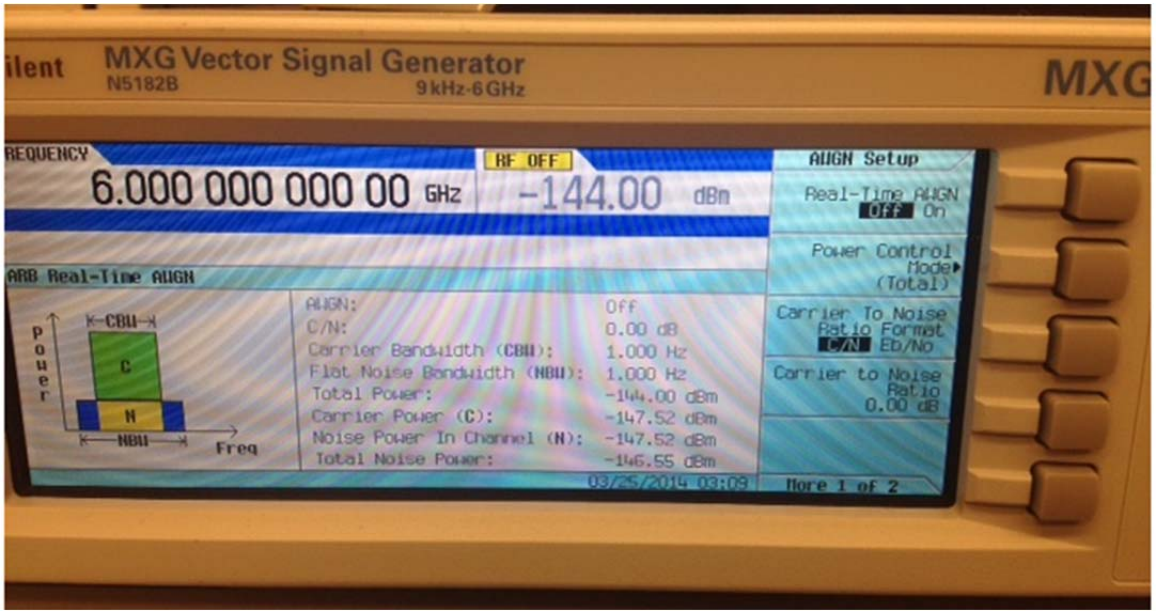


Figure 22: AWGN Setup Main Screen

From this screen, one can set the Carrier-to-Noise-Ratio. CNR and SNR are the same. For this example, set the CNR to 25dB. The carrier and noise bandwidth must also be set before turning the Real-Time AWGN option on.

Click *More 1 of 2*. Here the Carrier Bandwidth and Flat Noise Bandwidth can be set. In order to determine what the Carrier Bandwidth should be, one must examine the signal analyzer with full view of the signal. Move Marker 1 to the right side of the signal bandwidth. Take note of the Bandwidth location of this marker, as shown in Figure 23.

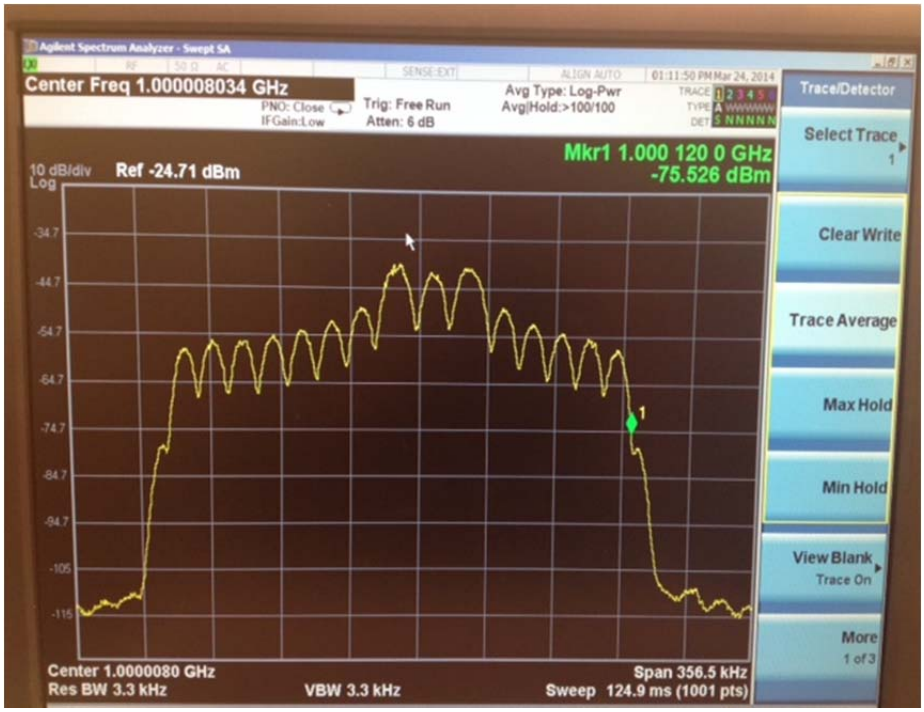


Figure 23: Finding the Carrier Bandwidth

Notice that the marker lands about 120 kHz above the 1 GHz center. On the left side, the carrier will land about 120 kHz left of the center. This means the total Carrier Bandwidth that will be set should be around 240 kHz. Once the Carrier Bandwidth is determined, just divide the Carrier Bandwidth by 1.25 to get the correct Flat Noise Bandwidth. This is simply due to instrument parameters, and there is no particular reason for the 1.25 scaling of bandwidth. Therefore, the Flat Noise Bandwidth should be about 192 kHz.

When these three variables for Noise are set and the AWGN option is turned on (Do not turn it on yet), the power level will be changing on the screen. The power should still not fluctuate above 0 dBm.

(IMPORTANT) If there are any errors concerning power, turn <u>off</u> 'RF On/Off' for at least 30 seconds and determine the cause of the error.

Always be particularly careful when setting low Carrier-to-Noise-Ratios. These ratios typically are 10 dB or less, depending on the settings. They are more likely to create large power fluctuations. If an SNR is very low relative to the system settings, the total output power of the signal should be decreased on the MXG using the button sequence:

AMPTD → Type in negative output power in dBm.
(Make sure to include a negative sign for this new power in dBm e.g. -20 dBm)

Once the option *Real-Time AWGN* is turned on, one must remember to turn on '*RF On/Off* ' for the signal to appear on the analyzer.

e. Reading the Signal

Now that the signal is downloaded and the correct SNR value has been set on the MXG, it is time to read in the signal from the SA into Matlab for post-processing.
In the *DownloadAndRead* folder, there is a file labeled *Reading_SA_IQ*. Within this script, there are two variables to consider changing, which are:

Bandwidth: Signal Bandwidth (Hz) -This is set to 192 kHz for this particular example. This is equal to the sample rate set on the download program (240 kHz) divided by 1.25; This number is due to instrument scaling only. If the signal was downloaded at X kHz, then it must be read in at X/1.25 kHz.

Measurement time: Time (seconds) - 6 seconds is enough for this case. The required time can be determined by observing the number of samples per sequence and the ARB sample clock samples/second.
(Samples/Sequence) / (Samples/Second) = (Seconds/Sequence);

Once these variables are set, run the code. It will automatically take the readings and store them under the variable IQData. Save the IQData under a name that is representative of the data at hand.

V. One Signal Generator Case – Total System

a. General Notes

This section will discuss how to model the complex system discussed using Matlab, the signal analyzer, and *one* of the signal generators. The target and covert systems are created, downloaded to one signal generator as a combined signal, are read back from the analyzer, and the BER of the covert and target system are found. The results from this section are inexact due to issues setting SNR options for the covert and target signal.

The multiple sets of code required for generating and analyzing the both systems can be found in the directory below for the one and two SG cases.

P:\covert_com\Equipment Measurement\Jon*Equipment_Analysis*

b. Creating the Waveform

From the directory above, select the folder labeled *One Signal Generator*. Open up the file *equipMain*.

Within this code, there are two variables to consider changing. The variable *SymRate* is set to be an integer from 1 to 4 and represents the symbol rate. Because the target system uses 16 QAM and the Covert system QPSK, if the two systems are transmitted with the same symbol length, the Covert system will be transmitting data at half the rate of the target system. This is because 16 QAM transmits 4 bits per symbol and QPSK transmits 2 bits per symbol. Thus:

SymRate = 1, 2, 3, 4;
1 : Rb/2 -- The systems have the same symbol lengths
2 : Rb/4 -- The covert symbol length is half that of the target symbol length
3 : Rb/8 -- The covert symbol length is a quarter that of the target symbol length
4 : Rb/16 -- The covert symbol length is an eighth that of the target symbol length

Therefore, *SymRate* is determined by which transmission rate is desired for the particular experiment under analysis.

cSNRo = const + 25; -- The 25 should be changed to whichever SNR in dB is under analysis for the covert signal. When the code is run, the main Matlab screen will show the Carrier-to-Noise-Ratio that is to be entered on the synchronized signal generator for correct BER analysis

Once these variables are set for the case under analysis, run the code. The set of signals needed for downloading the waveform will be saved automatically to a . mat file. This file is labeled by default as XmitSig. mat , but the file name should be changed to represent the settings under which the readings were taken.

c. Downloading the Waveform

IMPORTANT: The following commands must return as true in Matlab before downloading the waveform onto either SG. This is to avoid any potential problems with a power overload, which can damage the SG.

1) mean(TotalSig) ≤ 0.05
2) mean(abs(TotalSig).^2) ≤ 1

Double click on the created . mat file. The array *OFDMsig* should be checked for the above conditions.

If the above conditions are true for the transmission signal, then copy and paste the . mat file into the folder *DownloadAndRead.* Then open up the file *Download_WF_SG*. The following three variables need to be changed within the file:

load('File_name.mat') -- This should be changed to the . mat file name.

IQData = *OFDMsig;* -- This is for the one SG case, and doesn't vary.

.

addressMXG = '10.1.1.22' or '10.1.1.23';

-- These are the IP addresses of the signal generators. If '10.1.1.22' is used, the signal will be downloaded to the MXG signal generator. If '10.1.1.23' is used, the signal will be downloaded to the EXG signal generator.

One of the two signal generators will have a black cable plugged in behind it, connecting the signal generator to the signal analyzer. This cable synchronizes either the MXG or EXG to the signal analyzer. Only one signal generator can be synchronized at a time. For explanation purposes, the MXG is assumed to be synchronized. Because the EXG frequency calibration is slightly offset while unsynchronized, the signal to be analyzed must be downloaded to the MXG signal generator. The cable is shown in Figure 17.

The signal should be downloaded to '10.1.1.22' (MXG). The output power on the MXG should be set to -20 dBm (This is an acceptable output power for this example) and the center frequency should be 1 GHz. Look at the MXG and SA to double check that there are no warnings or errors. The power should never be above 0 dBm on the SA, as shown in Figure 20. Warning panels are shown in Figures 18 and 19.

(IMPORTANT) If there are any errors concerning power, turn <u>off</u> 'RF On/Off' for at least 30 seconds and determine the cause of the error.

Recommended: The RF output is best turned off before the ALC is turned off. The RF output is best turned off before changing noise settings. If RF is on, the SA could accidentally be harmed from a power spike without the user noticing.

Before adding Noise on the MXG, the Auto-Leveling Control (ALC) must be turned off. The ALC can be turned off on the MXG by clicking:

Cancel -> Turn off 'RF On/Off' -> AMPTD -> Turn off 'ALC'.

This turns off the waveform output to the SA, and then turns off the Auto-Level Control. If the ALC is left on, it will result in bad readings.

e. Add Noise to the Signal

The following button sequence should be selected to get to the AWGN settings:

Mode -> Dual Arb -> Arb Setup -> Real-Time AWGN Setup

Double check that the setting *Real-Time AWGN* is turned off. From the AWGN Setup screen, the *Carrier-to-Noise-Ratio* can be set to the value displayed in Matlab after running *equipMain* to generate the signal. For a 25db Covert SNR , the Carrier-to-Noise-Ratio displayed is 20.39 dB.

Click *More 1 of 2* on the screen displayed in Figure 22. From here, set the *Carrier Bandwidth* to *250 kHz* and *Flat Noise Bandwidth* to *190 kHz*. For the one SG case using the provided code, these values are both constant regardless of other settings.

The Carrier Bandwidth is 250 kHz to encompass both the OFDM and Covert Systems. Flat Noise should be approximately equal to the Carrier Bandwidth divided by 1.25 due to machine scaling.

Now that the three settings described are adjusted, turn *Real-Time AWGN* on. Then turn on the 'RF On/Off' output, and view the SA. The power level should be changing on the screen, however the power should still not be above 0 dBm.

(IMPORTANT) If there are any errors concerning power, turn <u>off</u> 'RF On/Off' for at least 30 seconds and determine the cause of the error.

e. Reading the Signal

Now that the signal is downloaded and the correct SNR value has been set on the MXG, it is time to read in the signal from the SA into Matlab for post-processing.
In the *DownloadAndRead* folder, there is a file labeled *Reading_SA_IQ*. Within this script, there are two variables to consider changing, which are:

Bandwidth: Signal Bandwidth (Hz) -- This is set to 192 kHz while using the provided code. This should be equal to the rate set on the download (240 kHz) divided by 1.25; this number is due to machine scaling.

Measurement time: Time (seconds) -- 6 seconds is enough for this case. The time required can be determined by observing the number of samples in one sequence and the ARB sample clock (samples/second) and cancelling units.

Once these variables are set, run the code. It will automatically take the readings and store them under the variable IQData. Save the IQData under a name that is representative of the data at hand.

f. Analyzing the Signal

The user should now have two . mat files, one with the original transmission signal and one with the IQData read in from the signal analyzer. Open the file ***equipRecover.*** The target and covert signals must be analyzed for BER separately, but the analysis will use the same code script (the program will be run twice). To select which signal is under analysis, variables within *equipRecover* will be changed according to their side notes on the next page.

The following variables need to be adjusted in the equipRecover code:

For the two lines that have the command *load(' ')*, enter the file names of the two . mat files representing the transmission signal and the received signal.

modType = 4 or 16 --16 if target is under analysis, 4 if covert is under analysis.

R_b = 1, 2, 4, 8 -- Used to determine the covert symbol length in calculations:
-- Rb/2 : 1
-- Rb/4 : 2
-- Rb/8 : 4
-- Rb/16: 8

phSync = PhaseResynchronization -- . This is the phase correction for the machinery, $e^{j\phi}$. This is the first value that must be determined by trial and error. Move this up or down in intervals of 0.5 and run the program. If the correlation gets higher, then the user is moving in the correct direction. This should be fine-tuned to the second decimal place to get the absolute highest correlation value. Once the maximum value is found, check that (phSync + π) does not return a higher correlation value.

nSub = 0, 1, 2, 3 -- This value indicate which of the sub-carriers is under analysis. If the target system is being analyzed, 0, 1, or 2 should be selected (the sub-carriers at 0kHz , 15kHz, and 30kHz respectively). If the covert system is being analyzed, this value should be set to 3 (sub-carrier at 45kHz).

seqStart = Index of Max Correlation; Used to calculate starting point of the sequence

-- Once the maximum correlation has been found i.e. the phase is resynchronized, use the Matlab coordinate finder to find the index of the maximum correlation spike. Choose the first correlation spike from the left, at the top of the spike. When the coordinate is found, right click the coordinate finder marker and click 'Select Text Update function', and then select 'NewCallback.m'. This will give the exact index of the maximum correlation value. This index of the maximum correlation value should be entered for SeqStart.

Once SeqStart is set to the correct value and the phase is re-synchronized, the program usually can calculate the BER successfully without adjusting the variables listed below. While the target system is under analysis, additional magnitude and phase synchronizations are sometimes required.

startShift = -8 → +8 maximum -- This is to account for the sample offset that the maximum correlation index may be off by. The index of the beginning of the signal is adjusted.. This is almost exclusively used for the target signal analysis and low SNRs.

magV = Magnification of Received Constellation diagram

-- This extra variable will only affect results when the target system (16QAM) is under analysis. Because 16QAM is sensitive to the amplitudes read from a sample, the top right corner of the constellation (if viewable) needs to be centered at 3+3j. This variable allows magnification of the amplitude.

phAdjust = Secondary Phase Correction of Received Constellation

-- One can often see the distinctive points that represent each integer level after demodulation. Sometimes these points are rotated so that the demodulation deciphers the values incorrectly. To rotate them to the right or left, adjust this value until the points are correctly centered. Once this last variable has been adjusted, the suspected BER should be presented on the main Matlab screen. This variable is typically only needed while analyzing the target system.

intRange = Integration Range -- This is typically left as a constant 0. However, for testing purposes, this variable was included to see how the BER is affected by the integration range over received symbol.

VI. Two Signal Generator Case – Total System

a. General Notes

This section will discuss how to model the complex system discussed using Matlab, the signal analyzer, and *two* signal generators. The target and covert system are created, downloaded onto their respective signal generators, are read back from the analyzer, and the BER of the covert and target system are found.

The multiple sets of code required for generating and analyzing the both systems can be found in the directory below for the one and two SG cases.

P:\covert_com\Equipment Measurement\Jon*Equipment_Measuring_Analysis*

b. Creating the Waveform

Select the folder labeled *Two Signal Generators*, then open up the script *equipMain*.

Decide which system to analyze- the target or covert system. Only one of these systems can be analyzed at a time. In order to analyze both the target and covert system for a selected covert SNR, two seperate signals must be generated using this code. One transmission signal will be tested to analyze the target system BER, and the other transmission signal will be tested to analyze the covert system BER.

Within the script *equipMain*, there are three variables to consider changing. These variables are *Target_or_Cov, SymRate,* and the covert SNR under the variable *cSNRo.*

The variable *Target_or_Cov* should be set to 0 or 1:
0: The target system BER is analyzed from the received signal
1: The covert system BER is analyzed from the received signal

The variable *SymRate* should be set to an integer from 1 to 4 to represent the desired symbol rate of the covert system. The target system uses 16 QAM and the covert system uses QPSK. If the two systems are transmitted with the same symbol length, the Covert system will naturally be transmitting data at half the rate of the target system. This is because 16 QAM transmits 4 bits per symbol and QPSK transmits 2 bits per symbol.

SymRate = 1, 2, 3, 4;
1 : Rb/2 -- The systems half the same symbol lengths
2 : Rb/4 -- The covert symbol length is half that of the target symbol length
3 : Rb/8 -- The covert symbol length is a quarter that of the target symbol length
4 : Rb/16 -- The covert symbol length is an eighth that of the target symbol length

cSNRo = const + 25; -- The 25 should be changed to whatever SNR in dB is under analysis for the covert signal.

Once these three variables are set for the case under analysis, run the code. The transmission signal will be saved automatically in a . mat file.

c. Downloading the Waveform

IMPORTANT: The following commands must return as true in Matlab before downloading the waveform onto either SG. This is to avoid any potential problems with a power overload, which can damage the SG.

1) mean(TotalSig) ≤ 0.05
2) mean(abs(TotalSig).^2) ≤ 1

Double click on the created . mat file. The array *uC* and *cD* should be checked for the above conditions.

If the above conditions are true for the transmission signal, then copy and paste the . mat file into the folder *DownloadAndRead.* Then open up the file *Download_WF_SG*. The following three variables need to be changed:

load('File_name.mat') -- This should be changed to the . mat file name.

IQData = *cD* or *uC;* -- If *cD* is selected, signal for target system analysis will be downloaded. If *uC* is selected, signal for covert system analysis will be downloaded.

addressMXG = '10.1.1.22' or '10.1.1.23';

-- These are the IP addresses of the signal generators. If '10.1.1.22' is used, the signal will be downloaded to the MXG signal generator. If '10.1.1.23' is used, the signal will be downloaded to the EXG signal generator. The signal under analysis will be downloaded to the synchronized signal generator (MXG in this case).

One of the two signal generators will have a black cable plugged in behind it, connecting the signal generator to the signal analyzer. This cable synchronizes either the MXG or EXG to the signal analyzer. Only one signal generator can be synchronized at a time. For explanation purposes, the MXG is assumed to be synchronized. Because the EXG frequency calibration is slightly offset while unsynchronized, the signal to be analyzed must be downloaded to the MXG signal generator. For this example, the covert signal will be under analysis and downloaded to the MXG. The exact steps one should take are listed on the next page.

If the covert system is under analysis, then *uC* should be downloaded to '10.1.1.22' (MXG). The output power for the covert signal will vary for each SNR according to Figure 24 on the next page. To change the output power of the signal generator, click the button AMPTD and type in the power. Look at the MXG and SA to double check that there are no warnings or errors. The power should never be above 0 dBm on the SA, as shown in Figure 20. Warning panels are shown in Figures 18 and 19.

(IMPORTANT) If there are any errors concerning power, turn <u>off</u> 'RF On/Off' for at least 30 seconds and determine the cause of the error.

The Auto-Leveling Control (ALC) must be turned off. The ALC can be turned off on the MXG by clicking:

Cancel -> Turn off 'RF On/Off' -> AMPTD -> Turn off 'ALC'.

After the MXG and PXA are checked for power and the ALC is turned off, on the MXG turn off 'RF On/Off'.

Afterwards, *cD* should be downloaded to '10.1.1.23'. The Target signal power will be set to -20 dBm and the center frequency will be 1 GHz for any case. Again, check that the output power on the SA is not above 0 db and that there are no errors. The target system should be centered at 1 GHz in frequency. However, because the EXG frequency offset is slightly off, it must be realigned manually to the exact target system frequency. Leave on 'RF On/Off' for the EXG.

(IMPORTANT) If there are any errors concerning power, turn <u>off</u> 'RF On/Off' for at least 30 seconds and determine the cause of the error.

Recommended: The RF output is best turned off before the ALC is turned off. The RF output is best turned off before changing noise settings. If RF is on, the SA could accidentally be harmed from a power spike without the user noticing.

Before realigning the frequency on the EXG, the Auto-Leveling Control (ALC) must be turned off. The ALC can be turned off on the EXG by clicking:

Cancel -> Turn off 'RF On/Off' -> AMPTD -> Turn off 'ALC'.

To realign the frequency offset of the EXG generator, first turn off all modulations so the signal appears steady on the PXA. To do this, click:

Turn on 'RF On/Off' -> Turn off 'Mod On/Off'

On the PXA, then click:

Freq Channel -> Auto Align.

This will show the signal center on the PXA. The center frequency on the analyzer display may be around 1.000000743 GHz. This is not the desired 1 GHz frequency of the target system. To fix this, on the EXG click *FREQ* and type in the result of the following calculation for the frequency in GHz.

1 GHz - 0.000000743 GHz = .999999257 GHz
Double check that the signal is recentered at 1 GHz on the SA. If not, then correct the amount of frequency offset on the EXG again.

Now that each system is centered at 1 GHz and the output power is set for each instrument, turn on the 'Mod On/Off' for the EXG. Turn on 'RF On/Off' for only the EXG. Check the power level on the SA. Now turn on 'RF On/Off' for the MXG. Again check that the power levels are below 0db. Once both machines are on, the power is in the correct range, and the center frequencies are set, noise can be added.

Covert @ X dB	Covert (MXG) Power (dBm)
25	-24
20	-29
15	-34
10	-39
5	-44
0	-49

Figure 24 : Table of Covert Powers used for Two Machine Case

d. Add Noise to the Signal

After the signal has been downloaded with all of the correct settings and it has been observed that there are no error warnings, one should add noise to the signal. The AWGN noise should be added only to the machine that is transmitting the target signal. For this example, noise is added to the machine with the target system (EXG).

Recommended: The RF output is best turned off before the ALC is turned off. The RF output is best turned off before changing noise settings. If RF is on, the SA could accidentally be harmed from a power spike without the user noticing.

The following button sequence should be selected to get to the AWGN settings:

Mode -> Dual Arb ->Arb Setup -> Real-Time AWGN Setup

Double check that the setting *Real-Time AWGN* is turned off. From this screen, the *Carrier-to-Noise-Ratio* can be set. Regardless of the desired covert SNR, this option will be set to 21.38 dB on the EXG. The covert signal's output power will be changed on the MXG to adjust the covert SNR.

Click *More 1 of 2*. Set the *Carrier Bandwidth* to *250 kHz* and *Flat Noise Bandwidth* to *190 kHz*. These are constant while using the provided code.
Carrier Bandwidth is 250 kHz to encompass both the OFDM and Covert Systems.
Flat Noise Bandwidth should be approximately equal to the Carrier Bandwidth divided by 1.25 due to machine scaling.

Now that the three noise settings are adjusted, turn *Real-Time AWGN* on and view the SA. The power level should be changing from the screen, however the power should still not be above 0 dBm.

(IMPORTANT) If there are any errors concerning power, turn <u>off</u> 'RF On/Off' for at least 30 seconds and determine the cause of the error.

e. Reading the Signal

Now that the signal is downloaded and the correct SNR value has been set on the signal generator, the signal can be read from the signal analyzer into Matlab.
In the *DownloadAndRead* folder, there is a script labeled *Reading_SA_IQ*.
In this script, there are two variables to consider changing before reading the signal:

Bandwidth: Signal Bandwidth (Hz) -- This is set to 192 kHz while using provided code. This should be equal to the rate set on the download (240 kHz) divided by 1.25; this number is due to machine scaling.

Measurement time: Time (seconds) -- 6 seconds is enough for this case. The time required can be calculated by observing the number of samples in one sequence and the ARB sample clock (samples/second) and cancelling units.

Once these variables are set, run the program and save the variable IQData under a name that is representative of the received signal settings, including which system is under analysis.

f. Analyzing the Signal

The user should now have two . mat files, one containing the transmission signal and one containing the received signal. Open the file *TwoEquipRecover.* The target and covert signals must be analyzed for BER separately, but they will use the same program (the program will be run twice). To select which signal is under analysis, variables within Two*EquipRecover* will be changed according to their side notes listed below.

The following variables need to be adjusted within the TwoEquipRecover code:

For the two lines that have the command *load(' ')*, enter the file names of the two . mat files described.

OFDM_Cov = 0 or 1 -- 0 if the target is under analysis, 1 if the covert is under analysis

R_b = 1,2,4,8 -- Used to determine the covert symbol length in calculations:
-- Rb/2 : 1
-- Rb/4 : 2
-- Rb/8 : 4
-- Rb/16: 8

phSync = PhaseResynchronization -- . This is the phase correction for the machinery, $e^{j\phi}$. This is the first value that must be determined by trial and error. Move this up or down in intervals of 0.5 and run the program. If the correlation gets higher, then the user is moving in the correct direction. This should be fine-tuned to the second decimal place to get the absolute highest correlation value. Once all of the following settings are adjusted, check that (phSync + π) does not return a better BER.

nSub = 1,2,3 -- This value indicates which of the sub-carriers is under analysis. If the target system is being analyzed, 1, 2, or 3 should be selected (the sub-carriers at 0kHz , 15kHz, and 30kHz respectively). If the covert system is being analyzed, this value does not affect anything.

seqStart = Index of Max Correlation; Used to calculate starting point of the sequence

-- Once the maximum correlation has been found i.e. the phase is resynchronized, use the Matlab coordinate finder to find the index of the maximum correlation spike. Choose the first correlation spike from the left, at the top of the spike. When the coordinate is found, right click the coordinate finder marker and click 'Select Text Update function', and then select 'NewCallback.m'. This will give the exact index of the maximum correlation value. This index of the maximum correlation value should be entered for SeqStart.

Once SeqStart is set to the correct value and the phase is re-synchronized, the program usually can calculate the BER successfully without adjusting the variables listed below. While the target system is under analysis, additional magnitude and phase synchronizations are sometimes required.

startShift = -8 → +8 maximum -- This is to account for the sample offset that the maximum correlation index may be off by. The index of the beginning of the signal is adjusted. correct BER can be recovered. This is almost exclusively used for the target signal analysis and low SNRs.

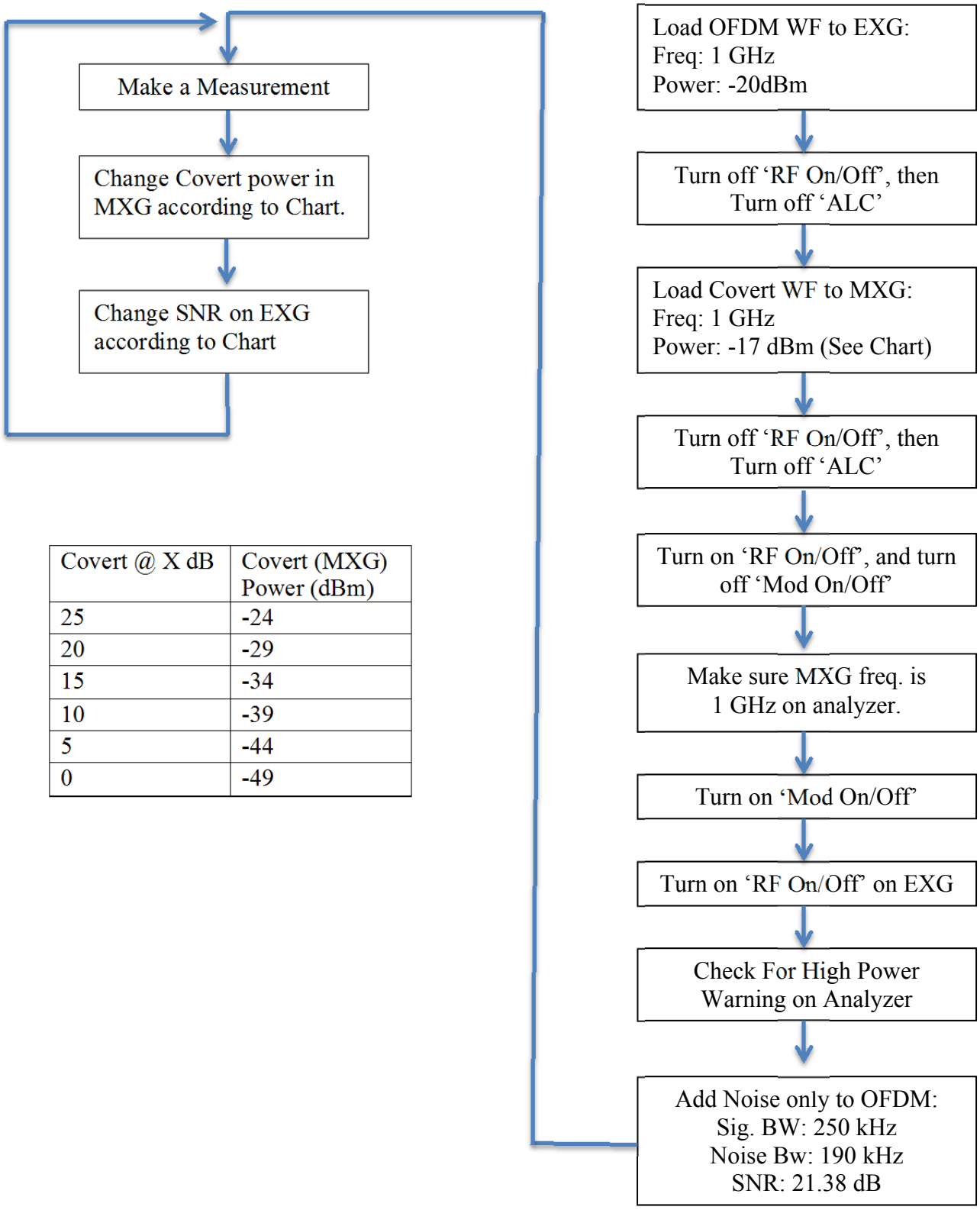
magV = Magnification of Received Constellation diagram

-- This extra variable will only affect results when the target system (16QAM) is under analysis. Because 16QAM is sensitive to the amplitudes read from a sample, the top right corner of the constellation (if viewable) needs to be centered at 3+3j. This variable allows magnification of the amplitude.

phAdjust = Secondary Phase Correction of Received Constellation

-- One can often see the distinctive points that represent each integer level after demodulation. Sometimes these points are rotated so that the demodulation deciphers the values incorrectly. To rotate them to the right or left, adjust this value until the points are correctly centered. Once this last variable has been adjusted, the suspected BER should be presented on the main Matlab screen.

Figure 25: Block Diagram of Two Machine Total System Case.
MXG Assumed to be Synchronized.
Covert System Under Analysis



Covert @ X dB	Covert (MXG) Power (dBm)
25	-24
20	-29
15	-34
10	-39
5	-44
0	-49

VII. Results

The results of the experimental testbed for the one signal-generator case are shown in Figure 26. For different data rates, the comparisons to the theoretical results have varying levels of agreement. For $\frac{Rb}{2}$, $\frac{Rb}{4}$, the emulated values obtained were closely matched to the computed results. The emulated results for $\frac{Rb}{8}$ and $\frac{Rb}{16}$ do not match the theoretical curves, but they do follow the correct trend. Using only one signal generator caused complications in accurately determining the desired SNR for each the target and covert signals. These power issues were solved by using two signal generators.

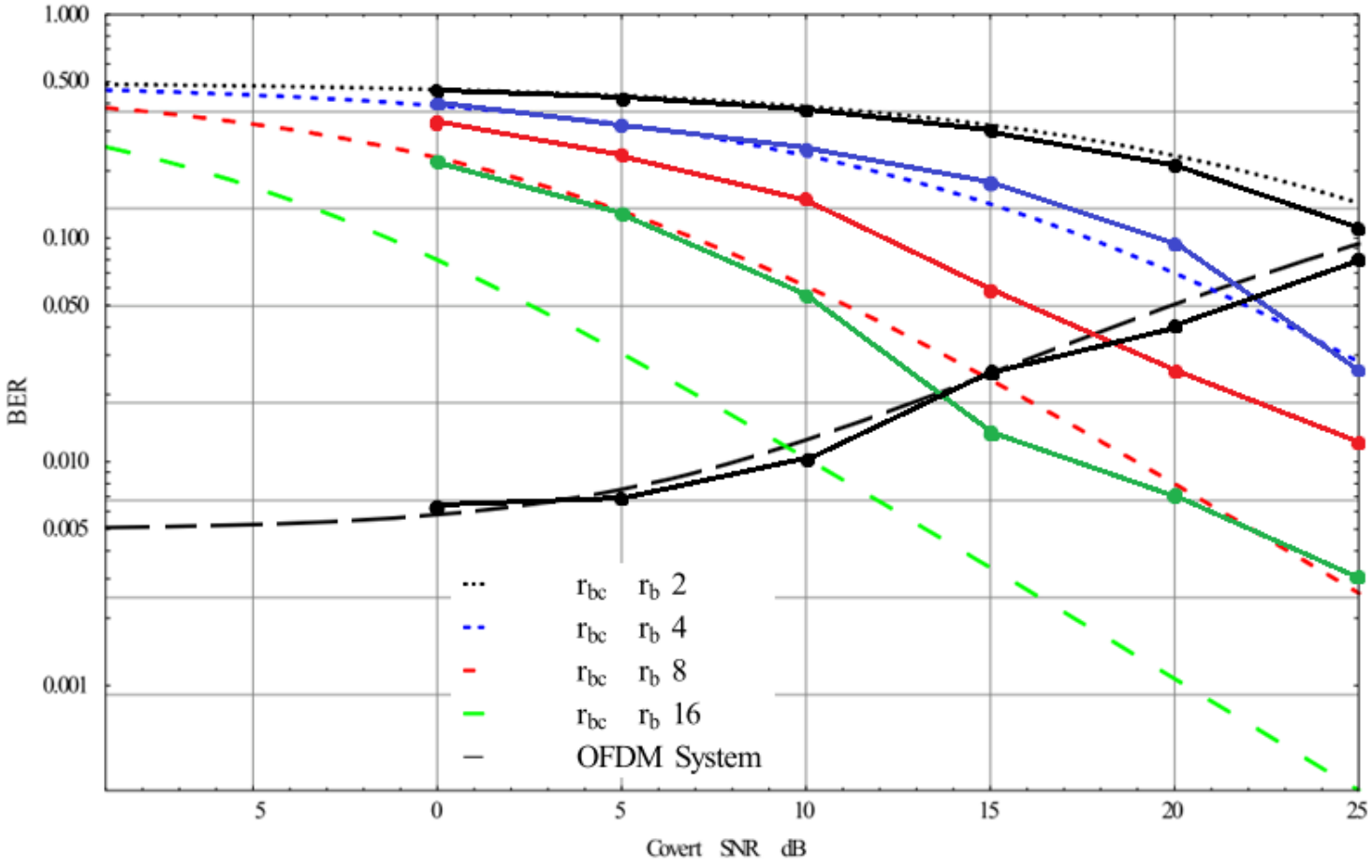


Figure 26: Theoretical vs. Emulated Results Using One Signal Generator

The results of the experimental testbed for the two signal-generator case are shown in Figure 27. These results follow the expected trends. These emulated results are most like those found in a real-time system.

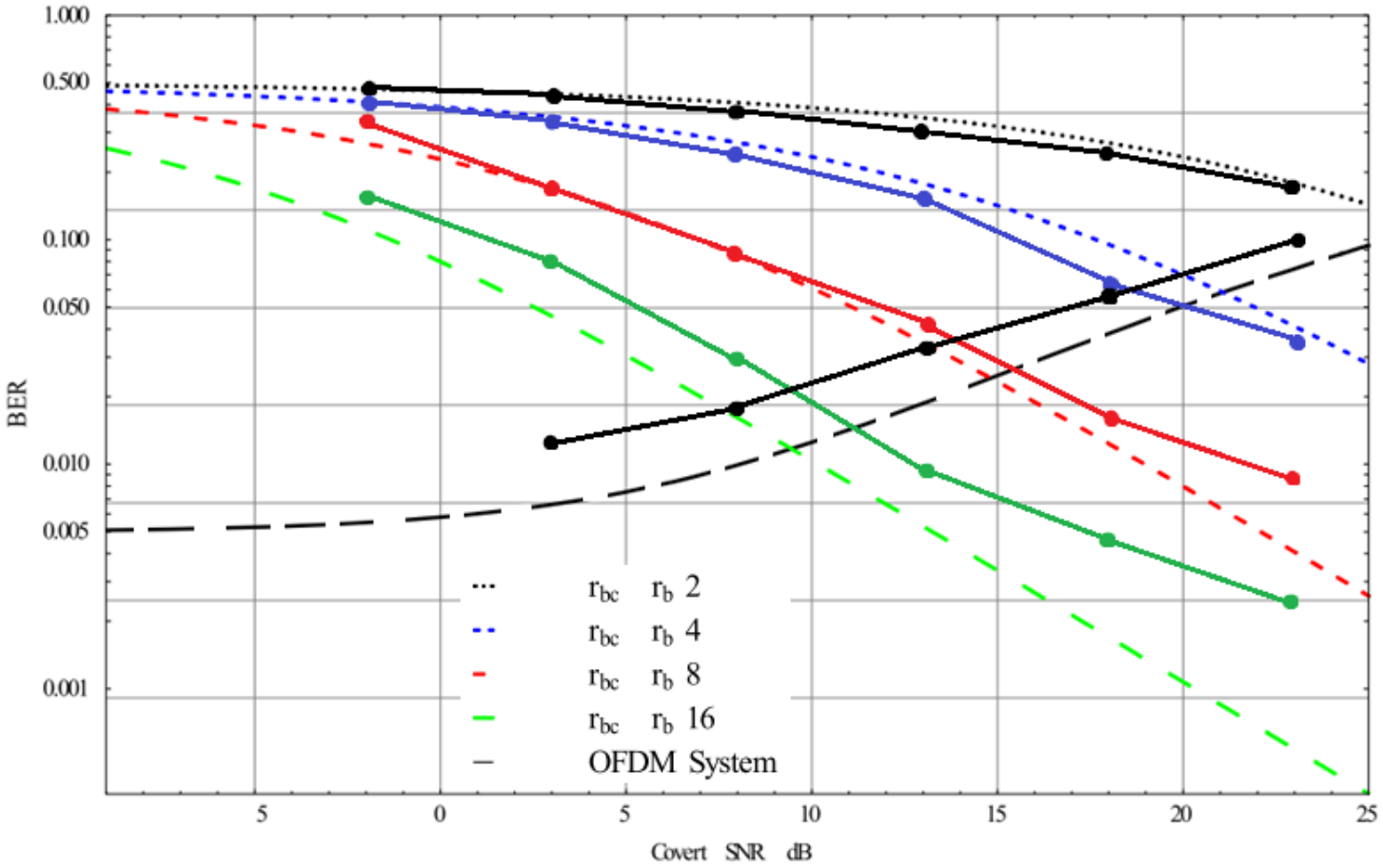


Figure 27: Theoretical vs. Emulated Results Using Two Signal Generators

VIII. Index of Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Matlab Simulation Code    %
% Written by: Jonathan Owen %
% Total Covert/Target System%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear; close all; clc;
randn('seed', 4);

for x = -10:25 %Covert SNR run through from __ to __

%% Generate the Sub-Carriers
SamSym = 16; % Samples / Symbol of the Sub-carrier signals
SamSymC = 16;  % Samples / Symbol of the Covert signal
SubsN = 3; % 3 Sub-Carriers excluding Covert signal
Offset = 8; % Covert time offset at half of Target symbol- SamSym/2
EsNodBSubs = 26.15; %Constant SNR for Target system
EsNodBCov = x;  %Changin SNR for Covert system, See For loop.
NumSym = 40000; %OFDM Number of Symbols
NumSymCov = 40000; %Covert Number of Symbols
%For different data rates, change NumSymCov by Half, Double SamSymC
%Currently at Rb/2 because the OFDM signal is 16QAM and the Covert
%signal is QPSK.

BWcov = 15000;
%Bandwidth is 15000 for Rb/2, 7500 for Rb/4, 3750 for Rb/8, so on.
%This is because the time it takes for one symbol to transmit becomes
%doubled, quadrupled, etc.

SubSeq = randi([0 15], 1, NumSym);
%NumSym is the number of symbols in the sequence.
%NumSym symbols are randomly generated from integers 0 to 15 (16QAM);
%Note that this is 1 sample/symbol.

for ii= 1:SubsN;
symSeq(ii, :) = SubSeq(1,:);
end
%Logs the original sequence transmitted for each OFDM signal
%for BER comparison later.
```

```
SubsD =  repmat(symSeq(1,:),SamSym, 1); %Where SamSym is designated
                                        %Samples/Symbol
SubsD = qammod(SubsD, 16);
signal_00k(1,:) = SubsD(1:end);
% Each subcarrier needs SamSym samples/symbol; Repmat replicates
% the original sequence SamSym times into new rows.
%
% Modulate SubsD to be 16QAM real and imaginary parts.
%
% Create a new vector signal_00k by taking data column-wise from SubsD;

%                                       SubsD
% From                    To          o---        -
%                                     o---        -
%      o-------                       o---        -
%                                     o---  ...   -
%                                     o---        -
%                                     o---        -


for ii= 1:SubsN;
SubCar(ii,:)=  signal_00k .* exp(2j*pi*((ii-1)…
*15000)*linspace(NumSym/(BW*length(signal_00k)),NumSym/15000,…
length(signal_00k)));
end

% Each Sub-carriers is phase-offset to the correct frequency center;
% 0, 15, 30 khz
% The signal centered at 0khz is multiplied by e^(j*2*pi*f*t) to create
% signals centered at 15khz and 30khz;
% To transmit NumSym Symbols, it will take from 0 to NumSym/15000
% seconds.
% Delta t is made to be the same vector length as signal_00k, so time
% is distributed over the whole sequence.

%% Generate the Covert

SubSeq = randi([0 3], 1, NumSymCov);
%NumSymCov symbols are randomly generated from integers 0 to 3 (QPSK);
% Note that this is 1 sample/symbol.

symSeq1(1, :) = SubSeq(1,:);
%Logs the original sequence transmitted for the Covert signal
%for BER comparison later.
```

```
temp = repmat(SubSeq, SamSymC, 1);
temp = qammod(temp, 4);
signal_cov(1,:) = temp(1:end);
% Each subcarrier needs SamSymC samples/symbol; Repmat replicates
% the original sequence SamSymC times into new rows.
%
% Modulate SubsD to be 16QAM real and imaginary parts.
%
% Create a new vector signal_cov by taking data column-wise from SubsD;
%                                        SubsD
% From                    To          o---        -
%                                     o---        -
%      o-------                       o---        -
%                                     o---  ...   -
%                                     o---        -

SubCar(SubsN+1,:)= signal_cov .* exp(2j*pi*((SubsN)*15000)*…
linspace(NumSymCov/(BWcov*length(signal_cov)),NumSymCov/BWcov,…
length(signal_cov))));
% The covert is phase-offset to the correct frequency center;
% 45 khz
% To transmit NumSymC Symbols, it will take from 0 to NumSymC/BWcov
% seconds.
% See BWcov variable set at beginning for explanation of why it isn't
% 15000 constant
% Delta t is made to be the same vector length as signal_cov, so time
% is distributed uniformly over the whole sequence.


%% Normalize power with reference to noise
% Note that this section is mainly for Theoretical modeling, not for
% the SA or SG modeling. It's still good to know.

No = .00002; % Assume some Noise power constant, in Watts for any one
             % sample

Spower = db2pow(EsNodBSubs)*No;
%Converts the Target S/N power from dB, Then Multiplies by No
%Watts/Sample; Results in Desired Power/Sample for Target
Cpower = db2pow(EsNodBCov)*No;
%Converts the Covert S/N power from dB, Then Multiplies by No
%Watts/Sample; Results in Desired Power/Sample for Covert
```

```
% In order to set the correct SNR's, we must first normalize the
% randomly generated signals so the mean power is 1. Then we increase
% the power of the signals to match the SNR desired.

pws00 = mean(abs(SubCar(1,:)).^2); % Find mean power of Sub-carrier 1
pws15 = mean(abs(SubCar(2,:)).^2); % Find mean power of Sub-carrier 2
pws30 = mean(abs(SubCar(3,:)).^2); % Find mean power of Sub-carrier 3
pwc45 = mean(abs(SubCar(4,:)).^2); % Find mean power of Covert Signal

% We divide the original signals by the sqrt(Power),
% which is the average voltage because power in comms systems is V^2

% Normalizes all signal powers to have an average power of 1.
SubCar(1,:) = SubCar(1,:)./sqrt(pws00);
SubCar(2,:) = SubCar(2,:)./sqrt(pws15);
SubCar(3,:) = SubCar(3,:)./sqrt(pws30);
SubCar(4,:) = SubCar(4,:)./sqrt(pwc45);

%Spower and Cpower represent what power the Signals needs have relative
%to the SNR, in Watts/Sample
%Multiplying something with an average power of 1 by the sqrt(Spower)
%will give the signal the SNR power desired.
signal_a = SubCar(1,:).*sqrt(Spower); % Sample* sqrt(V^2)/sample =Volts
signal_b = SubCar(2,:).*sqrt(Spower);
signal_c = SubCar(3,:).*sqrt(Spower);
signal_cov = SubCar(4,:).*sqrt(Cpower);


%% Add Fading
%-------------
% Sub-carriers

% We create NumSym Normal constants as X1I and X1Q
% sqrt(1/2) because only positive half of Fading is generated.
X1I    = sqrt(1/2)*randn(1,NumSym);
X1Q    = sqrt(1/2)*randn(1,NumSym);

% We take the magnitude of the Normalized curves to get a Rayleigh
% distribution
alpha1 = sqrt(X1I.^2+X1Q.^2);
R1new     = alpha1;

% We make each individual constant the same sample length of 1 symbol.
% The Rayleigh constants vector length should be the same as the
% Sub-carrier vector length.
ttemp = repmat(R1new, SamSym, 1);
R1(1,:) = ttemp(1:end);




%--------------
```

```
% Covert
% We create NumSymCov Normal constants as X2I and X2Q
% sqrt(1/2) because only positive half of Fading is generated.
X2I = sqrt(1/2)*randn(1,NumSymCov);
X2Q = sqrt(1/2)*randn(1,NumSymCov);

% We take the magnitude of the Normalized curves to get a Rayleigh
% distribution
alpha2 = sqrt(X2I.^2+X2Q.^2);
R2new     = alpha2;

% We make each individual constant the same sample length of 1 symbol.
% The Rayleigh constants vector length should be the same as the Covert
% vector length.
ttemp = repmat(R2new, SamSymC, 1);
R2(1,:) = ttemp(1:end);

%--------------
% Each signal is multiplied by its respective Rayleigh fading
signal_a  = R1.*(real(signal_a) + 1i*imag(signal_a));
signal_b  = R1.*(real(signal_b) + 1i*imag(signal_b));
signal_c  = R1.*(real(signal_c) + 1i*imag(signal_c));
signal_cov = R2.*(real(signal_cov) + 1i*imag(signal_cov));

%% Covert Time Shift
%We need to time shift both the real and imaginary parts of the Covert
%signal. Therefore, we split the real and imaginary parts.
inphase = real(signal_cov);
quadrature = imag(signal_cov);

%Moves 'Offset' samples from the end to the beginning of the signal.
%This represents our Desynchronization. Real part.
temp = inphase(end-(Offset-1):end);
inphase(end-(end-Offset-1):end) = inphase(1:end-Offset);
inphase(1:Offset)= temp;

%Moves 'Offset' samples from the end to the beginning of the signal.
%This represents our Desynchronization. Imaginary part.
temp = quadrature(end-(Offset-1):end);
quadrature(end-(end-Offset-1):end) = quadrature(1:end-Offset);
quadrature(1:Offset)= temp;

%Recombines the covert signal.
signal_cov = inphase + 1i* quadrature;
```

```
%% After all properties are added, the signals are added together.
TotalSig = signal_a + signal_b + signal_c + signal_c
%% Frequency domain For Matlab Double Check
% IF YOU ARE RUNNING THIS CODE, It is highly recommended you comment
% this part of the code out.

%  These have lengths of 8192 because of how the FFT is done in Matlab.
 freq_signal_a = zeros(1, 8192);
 freq_signal_b = zeros(1, 8192);
 freq_signal_c = zeros(1, 8192);
 freq_signal_cov = zeros(1, 8192);

%% The FFT is Calculated using Bartletts method.
% This puts each individual symbol of the signals into a variable
% newsignal. The FFT is done on newsignal, and the result is added into
% freq_signal_x. This will give us the Fourier Transform;

for ii = 1:(length(signal_a)/SamSym);
    newsignal = signal_a(1, ((ii-1)*SamSym+1):((ii-1)*SamSym+SamSym));
    freq_signal_a = abs(fft(newsignal, 8192)) + freq_signal_a;

    newsignal = signal_b(1, ((ii-1)*SamSym+1):((ii-1)*SamSym+SamSym));
    freq_signal_b = abs(fft(newsignal, 8192)) + freq_signal_b;

    newsignal = signal_c(1, ((ii-1)*SamSym+1):((ii-1)*SamSym+SamSym));
    freq_signal_c = abs(fft(newsignal, 8192)) + freq_signal_c;

    newsignal= signal_cov(1, ((ii-1)*SamSym+1):((ii-1)*SamSym+SamSym));
    freq_signal_cov = abs(fft(newsignal, 8192)) + freq_signal_cov;
end

% After doing an fftshift, one wants to plot from -F/2 to F/2.
% To find F, we multiply 16 samples/symbol by 15000 symbols/sec to get
% 240000 samples/sec.
% This calculation will be done FREQUENTLY while using the Signal
% Analyzer and Generator to set the ARB Sample Clock speed.

freq_range = linspace(-.120e6, .120e6, length(freq_signal_a));

% Plots the FFT of each indiviual signal
figure(6);
freq_signal_a = fftshift(freq_signal_a);
freq_signal_b = fftshift(freq_signal_b);
freq_signal_c = fftshift(freq_signal_c);
freq_signal_cov = fftshift(freq_signal_cov);
hold on
plot (freq_range, 20*log10(abs(freq_signal_a)), 'g');
plot (freq_range, 20*log10(abs(freq_signal_b)), 'b');
plot (freq_range, 20*log10(abs(freq_signal_c)), 'm');
plot (freq_range, 20*log10(abs(freq_signal_cov)), 'r');
title('Sub-Carrier Signals')
xlabel('Frequency (Hz)')
ylabel('Magnitude (M)')
hold off
```

```
% Repeat all of the above steps to see how the fft will look of the
% Total Signal. Note it won't look much like the above fft because the
% overlapping adds up. As a sanity check,
% each hump should be about 15000 Hz from the other.
%
freq_signal_e = zeros(1, 8192);

for ii = 1:(length(TotalSig)/SamSym);
    newsignal = TotalSig(1, ((ii-1)*SamSym+1):((ii-1)*SamSym+SamSym));
    freq_signal_e = fft(newsignal, 8192) + freq_signal_e;
end

freq_range = linspace(-.120e6, .120e6, length(freq_signal_e));

freq_signal_e = fftshift(freq_signal_e);
figure(5)
plot (freq_range, 20*log10(abs(freq_signal_e)), 'g');


%% Add Noise
% This technique was used to generate noise. It is only needed
% for Matlab simulations.
% Sqrt(X/2) because only positive half of noise power is considered.
% Sqrt(X/2) noise power is then distributed on a normal curve, i.e.
% positive and negative values. This distribution represents Sqrt(X).
%
% No = .00002 Watts for each sample.
% SamSym*No = Desired Watts per symbol
% The Desired Watts per Symbol distributed to each sample on a normal
% distribution.
RSig = real(TotalSig) + sqrt(SamSym*No/2)*randn(1,length(TotalSig));
ISig = imag(TotalSig) + sqrt(SamSym*No/2)*randn(1,length(TotalSig));

TotalSig = RSig + 1i.*ISig;


                          %%%%%%%%%%%%%%%
%                          RECEIVERS END
%                          %%%%%%%%%%%%%%%

%% Differentiate Signal A
% First we must remove the fading of the signal. This is the same for
% all sub-carriers and carries through to signal B and C
TotalSigSubs = TotalSig ./ R1;

% Next, we must remove the phase shift for sub-carrier 1
temp = TotalSigSubs .* …
exp(-2j*pi*0*linspace(NumSym/(BW*length(signal_a)),NumSym/15000, …
length(signal_a)));
```

```
% Because there seems to be no reliable integration method in Matlab,
% we take the mean over each symbol. The results will be off by a
% scaling factor.
% HOWEVER, this is fixed right after by Normalizing the signal, and
% then bringing it back up to the power it should be at.
integ1= [];
m=1;
for i= 1:SamSym:length(temp)
integ1(1,m) = mean( temp(i:(i-1)+ SamSym));
m = m+1;
end

%This is for re-scaling the recovered signal. It is the inverse of what
% was done while power scaling. See power scaling for explanation.
integ1 = integ1./sqrt(Spower);
integ1 = integ1.*sqrt(pws00);

% Plots the Constellation of Sub-carrier 1.
% figure(1);
% plot(integ1, 'r.');
% hold on
% plot([1 3], [1, 3], 'g.');
% title('Sub-Carrier Centered at 0 khz');
% hold off

%% Differentiate Signal B
% We must remove the phase shift for sub-carrier 2
temp = TotalSigSubs .* …
exp(-2j*pi*15000*linspace(NumSym/(BW*length(signal_b)),…
NumSym/15000,length(signal_b)));

% Because there seems to be no reliable integration method in Matlab,
% we take the mean over each symbol. The results will be off by a
% scaling factor.
% HOWEVER, this is fixed right after by Normalizing the signal, and
% then bringing it back up to the power it should be at.
integ2= [];
m=1;
for i= 1:SamSym:length(temp)
integ2(1,m) = mean( temp(i:(i-1)+ SamSym));
m = m+1;
end

%This is for re-scaling the recovered signal. It is the inverse of what
% was done while power scaling. See power scaling for explanation.
integ2 = integ2./sqrt(Spower); %change signal power for correct SNR
integ2 = integ2.*sqrt(pws15); %normalize signal power

% Plots the Constellation of Sub-carrier 2.
% figure(2);
% plot(integ2, 'r.');
% hold on
% plot([1 3], [1, 3], 'g.');
% title('Sub-Carrier Centered at 15 khz');
% hold off
```

```
%% Differentiate Signal C
% We must remove the phase shift for sub-carrier 3
temp = TotalSigSubs .* …
exp(-2j*pi*15000*linspace(NumSym/(BW*length(signal_c)),…
NumSym/15000,length(signal_c)));

% Because there seems to be no reliable integration method in Matlab,
% we take the mean over each symbol. The results will be off by a
% scaling factor.
% HOWEVER, this is fixed right after by Normalizing the signal, and
% then bringing it back up to the power it should be at.
integ3= [];
m=1;
for i= 1:SamSym:length(temp)
integ3(1,m) = mean( temp(i:(i-1)+ SamSym));
m = m+1;
end

%This is for re-scaling the recovered signal. It is the inverse of what
%was done while power scaling. See power scaling for explanation.
integ3 = integ3./sqrt(Spower); %change signal power for correct SNR
integ3 = integ3.*sqrt(pws30); %normalize signal power

% Plots the Constellation of Sub-carrier 3.
% figure(3);
% plot(integ3, 'r.');
% hold on
% plot([1 3], [1, 3], 'g.');
% title('Sub-Carrier Centered at 30 khz');
% hold off

%% Differentiate Covert Signal
%We must first inverse the time shift. This is simply moving the offset
%from the beginning back to the end of the signal, using the same
%process.
inphase1= real(TotalSig);
quadrature1 = imag(TotalSig);

temp = inphase1(1:Offset);
inphase1(1:end-Offset) = inphase1(end-(end-Offset-1):end);
inphase1(end-(Offset-1):end) = temp;

temp = quadrature1(1:Offset);
quadrature1(1:end-Offset) = quadrature1(end-(end-Offset-1):end);
quadrature1(end-(Offset-1):end) = temp;

TotCov = inphase1 + 1j* quadrature1;
```

```
% The fading is removed from the time shifted signal.
TotCov = (TotCov)./ R2;
%
% The phase offset is removed.
temp = TotCov .* …
exp(-2j*pi*45000*linspace(NumSymCov/(BWcov*length(signal_cov)), …
NumSymCov/BWcov,length(signal_cov)));

% Because there seems to be no reliable integration method in Matlab,
% we take the mean over each symbol. The results will be off by a
% scaling factor.
% HOWEVER, this is fixed right after by Normalizing the signal, and
% then bringing it back up to the power it should be at.
integ4= [];
m=1;
for i= 1:SamSymC:length(temp)
integ4(1,m) = mean( temp(i:(i-1)+ SamSymC));
m = m+1;
end

%This is for re-scaling the recovered signal. It is the inverse of what
%was done while power scaling. See power scaling for explanation.
integ4 = integ4./sqrt(Cpower); %change signal power for correct SNR
integ4 = integ4.*sqrt(pwc45); %normalize signal power

% Plots the Constellation of Covert signal.
% figure(4);
% plot(integ4, 'r.');
% hold on
% plot([1 3], [1, 3], 'g.');
% title('Covert Centered at 45 khz');
% hold off

%% BER
modType = 16;
modTypeC = 4;

% All of the recovered data is demodulated for calculating bit error
rate.
deData1 = qamdemod(integ1 , modType);
deData2 = qamdemod(integ2 , modType);
deData3 = qamdemod(integ3 , modType);
deData4 = qamdemod(integ4 , modTypeC);
%
numBit1 = 0;
errR1 = 0;
numBit2 = 0;
errR2 = 0;
numBit3 = 0;
errR3 = 0;
```

```
% The Known transmitted signal is demodulated
b_dData1 = de2bi(symSeq(1, :)', log2(modType), 'left-msb')';

% The Received signal is demodulated
b_deData1 = de2bi(deData1.', log2(modType), 'left-msb')';
b_deData2 = de2bi(deData2.', log2(modType), 'left-msb')';
b_deData3 = de2bi(deData3.', log2(modType), 'left-msb')';

% Calculating BER for Sub-Carriers.
[temp1, temp2] = biterr(b_deData1, b_dData1);
numBit1 = numBit1 + temp1;
errR1 = errR1 + temp2;
BE_subs1 = (errR1);
clear temp1 temp2;
%
[temp1, temp2] = biterr(b_deData2, b_dData1);
numBit2 = numBit2 + temp1;
errR2 = errR2 + temp2;
BE_subs2 = (errR2);
clear temp1 temp2;
%
[temp1, temp2] = biterr(b_deData3, b_dData1);
numBit3 = numBit3 + temp1;
errR3 = errR3 + temp2;
BE_subs3 = (errR3);
clear temp1 temp2;

% Calculates the BER average for ALL sub-carriers
BE_subs = mean([BE_subs1, BE_subs2, BE_subs3])

%--------------
numBitCov =0;
errR= 0;
% The Known transmitted signal is converted to binary
b_dData = de2bi(symSeq1(1, :)', log2(modTypeC), 'left-msb')';
% The Received signal is converted to binary
b_deData = de2bi(deData4.', log2(modTypeC), 'left-msb')';
% Calculating BER for Cov
[temp1, temp2] = biterr(b_deData, b_dData);
numBitCov = numBitCov + temp1;
errR = errR + temp2;

numBitCov;
BE_cov = errR;

%The BER data is stored.
DataT(x+11,:) = [BE_subs BE_cov]
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 16 QAM Signal Generation       %
% Written by: Jonathan Owen      %
% Target System Only: One SG     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SubSeq = randi([0 15], 1, NumSym); % NumSym is the number of symbols in
                                   % one sequence. Each symbol is
                                   % represented by 1 sample integer

Signal_a =  repmat(SubSeq(1,:),SamSym, 1); %Where SamSym is the desired
                                           %number of Samples/Symbol
Signal_a = qammod(Signal_a, ModType);
                                    % Modulates the sequence to 16QAM
                                    % ModType = 2^(N bits/symbol)
                                    % ModType = 2^4 = 16 for 16QAM
                                    %
                                    % The sequence is stored. This
                                    % sequence represents the sub-
                                    % carrier centered at 0kHz in
                                    % frequency.
                                    % x(t)*e^(j*2*pi*0*t) = x(t)

t = linspace(0,NumSym/15000,length(signal_b));
% The time required to transmit NumSym symbols is NumSym/Ts.

SubSeq = randi([0 15], 1, NumSym);
Signal_b =  repmat(SubSeq(1,:),SamSym, 1);
Signal_b = qammod(Signal_b, ModType);
Signal_b = Signal_b .* exp(2j*pi*30000)*t);

SubSeq = randi([0 15], 1, NumSym);
Signal_c =  repmat(SubSeq(1,:),SamSym, 1);
Signal_c = qammod(Signal_c, ModType);
Signal_c = Signal_c .* exp(2j*pi*45000*t);

TotalSig = Signal_a + Signal_b + Signal_c;

% The transmission signal is created by combining the sequences of
% each sub-carrier. The power of TotalSig MUST still be normalized.
% The transmission signal must be normalized to have a power of 1.
% First, the mean power of the unscaled signal must be found.
% Mean power is ideally equal to |Mean Voltage|^2 in comms systems
UnscaledPow = mean(abs(TotalSig).^2);

% TotalSig (volts) is divided by the sqrt(UnscaledPow) (mean voltage)
% This normalizes the mean transmission power of the sequence to 1.
TotalSig = TotalSig./sqrt(UnscaledPow);

Ref = zeros(1,SamSym*12);
RefSig(SamSym*4:SamSym*8-1)=
                        cos(2*pi*60000*(linspace(0,4/15000,SamSym*4)));
% The reference signal is X = 12 symbols long.
% The cosine is X-8 = 4 symbols long. The frequency ‘f’ is 60kHz.
```

```
inphase = [real(TotalSig) RefSig];
quadrature = [imag(TotalSig) Ref];
TotalSig = inphase + 1i.*quadrature;

save('OFDMpractice', 'TotalSig');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Downloading Waveform to S.G.   %%%
%%%         G. Shabsigh, Ph.D.        %%%
%%%      The University of Kansas     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all; close all; clc


% Load waveform matrix
load('stCovCurve.mat')
IQData = cD; % cD: OFDM, uC: covert

% S.G. IP Address
addressMXG = '10.1.1.23'; %23: EXG, 22: MXG

% Sampling Rate (sample/sec)
sampleRate = 240e3;

% Carrier Freq. (Hz)
centerFrequency = 1e9;

% Output Power (dBm)
outputPower = -20;



%%                              WARNING                                  %%
%% It is not recommended to change any of the code beyond this line %%


% Terminate the if the carrier freq. is below 50MHz
if(centerFrequency<50e6)
    display('ERROR: Carrier Freq. is below 50MHz')
    display('Program is terminated.')
    break;
end

% Terminate the code if power is greater than -5dBm
if(outputPower>-5)
    display('ERROR: Power is too high')
    display('Program is terminated.')
    break;
end

MXGdownloadM(addressMXG, IQData, sampleRate, centerFrequency,
outputPower)


display('Download from MATLAB is complete.')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Downloading Waveform to S.G. Function   %%%
%%%        G. Shabsigh, Ph.D.                 %%%
%%%     The University of Kansas              %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function MXGdownloadM(addressMXG, IQData, sampleRate, centerFrequency,
outputPower)
% This function downloads IQ Data to the signal generator's non-
% volatile
% memory using a TCPIP sockets connection. This function takes the
% following inputs:
% * The IP Address of the signal generator
% * The waveform which is a row vector.
% * The sample rate of the signal
% * The center frequency of the signal
% * The output power of the signal generator
%
% The waveform playback is continuous and the event1 line will be
% triggered
% at the start of each repetition of the signal
%
% Syntax: downloadWaveform('10.10.10.10', Test_IQData, 22e6, 2e9, -10)

if ~isvector(IQData)
    error('downloadWaveform: invalidInput','Invalid input data'); %#ok
else
    IQsize = size(IQData);
    % User gave input as column vector. Reshape it to row vector.
    if ~isequal(IQsize(1),1)
        warning('Wrong input detected. Automatically converting to row
vector.'); %#ok
        IQData = reshape(IQData,1,IQsize(1));
    end
end



%% Define the system parameters. These may be tuned before making the
% measurement
ArbFileName = 'MATLABSignal';       % filename for the data in the ARB
numberOfSamples = length(IQData);

%% Connection to instrument and waveform download
% Open a TCPIP/GPIB connection to the instrument
signalGeneratorObject = tcpip(addressMXG,5025);
% Set up the output buffer size
signalGeneratorObject.OutputBufferSize = numberOfSamples*9*2;
% Set output to Big Endian with TCPIP objects, because we do the
interleaving
% and the byte ordering in code. For VISA or GPIB objecs, use
littleEndian.
signalGeneratorObject.ByteOrder = 'bigEndian';
% Set the timeout
signalGeneratorObject.Timeout = 20;
% Open connection to the instrument
fopen(signalGeneratorObject);
```

```
%% Download signal
% Seperate out the real and imaginary data in the IQ Waveform
wave = [real(IQData);imag(IQData)];
wave = wave(:)';    % transpose the waveform

% Scale the waveform if necessary
tmp = max(abs([max(wave) min(wave)]));
if (tmp == 0)
    tmp = 1;
end

% ARB binary range is 2's Compliment -32768 to + 32767
% So scale the waveform to +/- 32767 not 32768
scale = 2^15-1;
scale = scale/tmp;
wave = round(wave * scale);
modval = 2^16;
% Get data from double to unsigned int
wave = uint16(mod(modval + wave, modval));

%%%%%%%%%%%%%%%%%%
% % If on a PC swap the bytes to Big Endian
% [aa, bb, cc] = computer;
% if strcmp( cc, 'L' )
%     wave = bitor(bitshift(wave,-8),bitshift(wave,8));
% end
%%%%%%%%%%%%%%%%%%

% Start from known instrument state
fprintf(signalGeneratorObject,'*RST');

% Query the instrument identity and display it.
instrumentInfo = query(signalGeneratorObject, '*IDN?');
fprintf('Instrument identification information: %s', instrumentInfo);

% Some settings commands to make sure we don't damage the instrument
fprintf(signalGeneratorObject,':OUTPut:STATe OFF');
fprintf(signalGeneratorObject,':SOURce:RADio:ARB:STATe OFF');
fprintf(signalGeneratorObject,':OUTPut:MODulation:STATe OFF');

% Write the data to the instrument
n = size(wave);
fprintf('Starting Download of %d Points...',n(2)/2);
binblockwrite(signalGeneratorObject,wave,'uint16',[':MEM:DATA "NVWFM:'
ArbFileName '",']);
fprintf(signalGeneratorObject,'');
disp('...done!');

% The previous operation could take time. Ensure that it is done before
% moving on
measureComplete = query(signalGeneratorObject,'*OPC?'); %#ok
```

```
%% Setting up the instrument to generate a selected waveform
% Clear Volatile memory waveforms

fprintf(signalGeneratorObject, ':MMEMory:DELete:WFM');
% *** DO NOT DELETE WAVEFORMS FROM NONVOLATILE MEMORY ***
% Get the available waveforms available in NonVolatile Waveform Memory
% DISP this variable to get a list of all available waveforms
% waveforms = query(signalGeneratorObject,…
% ':MMEMory:CATalog?"NVWFM:"');

% Copy the MATLABSignal to Volatile memory
fprintf(signalGeneratorObject,[':MEMory:COPY:NAME "NVWFM:' ArbFileName
'","WFM1:MATLAB_WFM"']);
% The previous operation could take time. Ensure that it is done before
% moving on
measureComplete = query(signalGeneratorObject,'*OPC?'); %#ok


%%%%%%%%%%%%%%%%%%%%%%%
% Clear all markers from the file
fprintf(signalGeneratorObject,':SOURce:RADio:ARB:MARKer:CLEar:ALL
"MATLAB_WFM",1');
% Set marker 1 (first input after filename), starting at the first
% point (second input), ending at point 1 (third input) and skipping 0.
% Refer page 295 of http://cp.literature.agilent.com/litweb/pdf/N5180-
% 90004.pdf for more info
fprintf(signalGeneratorObject,':SOURce:RADio:ARB:MARKer:SET
"MATLAB_WFM",1,1,1,0');
% Play back the selected waveform
fprintf(signalGeneratorObject, ':SOURce:RADio:ARB:WAVeform
"WFM1:MATLAB_WFM"');
% Set the scaling to 70%
fprintf(signalGeneratorObject,':SOURce:RADio:ARB:SCALing
"MATLAB_WFM",70');
%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Set the sample rate (Hz) for the signal.
% You can get this info for the standard signals by looking at the data
% in the 'waveforms' variable
fprintf(signalGeneratorObject,[':SOURce:RADio:ARB:SCLock:RATE '
num2str(sampleRate)]);
% set center frequency (Hz)
fprintf(signalGeneratorObject, ['SOURce:FREQuency '
num2str(centerFrequency)]);
% set output power (dBm)
fprintf(signalGeneratorObject, ['POWer ' num2str(outputPower)]);
% make sure output protection is turned on
fprintf(signalGeneratorObject,':OUTPut:PROTection ON');
% turn off internal AWGN noise generation
fprintf(signalGeneratorObject,':SOURce:RADio:ARB:NOISe:STATe OFF');
% turn off phase noise impairment
% Do not use phase noise command for PSG
% fprintf(signalGeneratorObject,':SOURce:RADio:ARB:PHASe:NOISe:STATe
% OFF');
% ARB Radio on
fprintf(signalGeneratorObject, ':SOURce:RADio:ARB:STATe ON');
% modulator on
fprintf(signalGeneratorObject, ':OUTPut:MODulation:STATe ON');
% RF output on
fprintf(signalGeneratorObject, ':OUTPut:STATe ON');


%% Clean up
% Close the connection to the instrument
fclose(signalGeneratorObject);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%     Reading IQ Data from Agilent S.A     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clc; clear all; close all;

% S.A. IP Address
addressPXA = '10.1.1.20';

% Center frequency of the modulated waveform (Hz)
centerFrequency = 1e9;

% Signal bandwidth (Hz)
bandwidth = 192e3;

% Measurement time (s)
measureTime = 6;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%            WARNING                   %%
%% It is not recommended to change  %%
%% any of the code beyond this line %%

pxa=tcpip(addressPXA, 5025);
set(pxa,'InputBufferSize',40e6);
set(pxa,'Timeout',25);
fopen(pxa);

fprintf(pxa, '*RST');
instrumentInfo = query(pxa, '*IDN?');
disp(['Instrument identification information: ' instrumentInfo]);

% Set up signal analyzer mode to Basic/IQ mode
fprintf(pxa,':INST:SEL BASIC');
fprintf(pxa, '*RST');


% Set the center frequency
fprintf(pxa,[':SENSe:FREQuency:CENTer ' num2str(centerFrequency)]);
% set to take one single measurement once the trigger line goes high
fprintf(pxa,':INIT:CONT OFF');
% Set the time for which measurement needs to be made
fprintf(pxa,[':WAVeform:SWE:TIME '  num2str(measureTime)]);
% Set the resolution bandwidth
fprintf(pxa,[':SENSe:WAVEform:BANDwidth:RESolution '
num2str(bandwidth)]);
% Turn off averaging
fprintf(pxa,':SENSe:WAVeform:AVER OFF');
```

```
% Set the endianness of returned data
fprintf(pxa,':FORM:BORD NORM');
% Set the format of the returned data
fprintf(pxa,':FORM:DATA REAL,32');

% % Initiate measurement
fprintf(pxa,':INIT:WAV');
% wait till measurement operation is complete
measureComplete = query(pxa,'*OPC?');

display('Measurement Initiated.')

if(measureComplete == 0)% || findstr(lastwarn, 'timeout occurred') ||
findstr(lastwarn, 'Unsuccessful'))
    % Reset and switch back to the spectrum analyzer view
    fprintf(pxa, '*RST');
    fprintf(pxa,':INSTrument:SELect SA');
    fprintf(pxa, '*RST');
    % Close and delete instrument connections
    fclose(pxa);
    delete(pxa); clear pxa;

    rstDueErr1 = 1
    break;
end



% Set the format of the returned data
fprintf(pxa,':FORM:DATA REAL,32');
% Read the IQ data
fprintf(pxa,':READ:WAV0?');
data=binblockread(pxa,'float');
% Read the additional terminator character from the instrument
fread(pxa,1);

if(isempty(data))
% || findstr(lastwarn, 'binblock') || findstr(lastwarn,'Unsuccessful'))
% Reset and switch back to the spectrum analyzer view
    fprintf(pxa, '*RST');
    fprintf(pxa,':INSTrument:SELect SA');
    fprintf(pxa, '*RST');
% Close and delete instrument connections
    fclose(pxa);
    delete(pxa); clear pxa;

    rstDueErr2 = 1
    break;
end
```

```
display('Data is Read.')

% Separate the data and build the complex IQ vector.
inphase=data(1:2:end);
quadrature=data(2:2:end);
IQData=inphase+1i*quadrature;

fprintf(pxa,':FETCH:WAV1?');
signalSpec = binblockread(pxa,'float');
sampleRate = 1/signalSpec(1);
disp(['Sample Rate (Hz) = ' num2str(sampleRate)]);
disp(['Number of points read = ' num2str(signalSpec(4))]);
disp(['Max value of signal (dBm) = ' num2str(signalSpec(6))]);
disp(['Min value of signal (dBm) = ' num2str(signalSpec(7))]);

% Switch back to the spectrum analyzer view and reset
fprintf(pxa,':INST:SEL SA');
fprintf(pxa, '*RST');

% Close and delete instrument connections
fclose(pxa);
delete(pxa); clear pxa;



display('IQ Reading is Complete.')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generating OFDM/Covert Waveform %
%    for Equipment Measurement    %
%   Ghaith Shabsigh - ITTC 2013   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Main File

clear all; clc; close all;


SymRate = 1;                    % 1: rb/2;
                                % 2: rb/4;
                                % 3: rb/8;
                                % 4: rb/16
const = -26.15;                 % Constant
cSNRo = const + 25;             % <--- Replace the number by the SNR of
                                %cov.

cSymo = [40e3 20e3 10e3 5e3]; % Number of Symbols, Covert, depends on
                                %SymRate
cRepo = [16 32 64 128];         % Number of Samples/Symbol, Covert
nuSymo = 40e3;                  % Number of Symbols, OFDM


%%%%%%%%%%%%%%%%%%%%%%%%

display(['Setup S.G. SNR at: ' ...
    num2str(26.15-10*log10(3+(10^(0.1*cSNRo)))) ' dB'])


% Calling Function
equipCovert(cSymo(SymRate), cRepo(SymRate), cSNRo, nuSymo);
                                            %Generate Transmission Signal

% NOTE1: The generated Waveform will be Saved in Current Folder
% Name of WF Matrix can be changed in the function .

% NOTE2: Number of Samples per Symbol for the OFDM is 16.

% NOTE3: Covert Modulation is QPSK, OFDM Modulation is 16QAM




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Generating Covert Waveform fn. %
%    for Equipment Measurement    %
%   Ghaith Shabsigh - ITTC 2013   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function equipCovert(cSym, cRep, cSNR, nuSym)


No = 0.03;            % Constant
S = No*db2pow(26.15); % Setting up the OFDM Signal at Some Level
C = S*db2pow(cSNR);   % Attenuating Covert Signal to Achieve Cov. SNR

t = (0:(16*nuSym-1))/(16*15000); % Time Duration of the WF

                          %%%%%%%%%%%%%%%%%
                              %Target%

% Gen data
tSequence1 = randi([0 15], 3, nuSym); % QAM16
tSequence2 = randi([0 15], 3, nuSym); % QAM16
tSequence3 = randi([0 15], 3, nuSym); % QAM16

% Modulate / Up sample
temp = repmat(qammod(tSequence1(1, :), 16), 16, 1);
rD(1, :) = temp(:).';

temp = repmat(qammod(tSequence2(2, :), 16), 16, 1);
rD(2, :) = temp(:).';

temp = repmat(qammod(tSequence3(3, :), 16), 16, 1);
rD(3, :) = temp(:).';

% Frequency Shift / Up convert
f = 15000;
uD = rD.*[ones(1, 16*nuSym); exp(2i*pi*f*t); exp(4i*pi*f*t)];

clear('temp', 'rD', 'reD', 'imD')

% Power scaling each sub-carrier
pws1 = mean(abs(uD(1, :)).^2);
pws2 = mean(abs(uD(2, :)).^2);
pws3 = mean(abs(uD(3, :)).^2);

uD(1, :) = uD(1, :)*sqrt(S/pws1);
uD(2, :) = uD(2, :)*sqrt(S/pws2);
uD(3, :) = uD(3, :)*sqrt(S/pws3);

% Add Rayleigh Fading
tFading = repmat(sqrt(0.5*randn(1, nuSym).^2 + 0.5*randn(1, nuSym).^2),
16, 1);
tFading = tFading(:).';

% Combine all Sub-carriers and Fading into one signal
```

```
cD = sum(uD, 1).*tFading;

                                %%%%%%%%%%%%%%%%%%
                                    %Covert%


% Covert
% Gen data
cSequence = randi([0 3], 1, cSym); %QPSK

% Modulate / Up sample
tempz = repmat(qammod(cSequence(1, :), 4), cRep, 1);
rC(1, :) = tempz(:).';

% Frequency Shift / Up convert
fo = 45000;
uC = rC.*exp(2i*pi*fo*t);

% Power scaling covert
pwc = mean(abs(uC).^2);
uC = uC*sqrt(C/pwc);

% Add Rayleigh Fading
cFading = repmat(sqrt(0.5*randn(1, cSym).^2 + 0.5*randn(1, cSym).^2),
cRep, 1);
cFading = cFading(:).';

uC = uC.*cFading;

% Reference Signal Variables declared
fr = 45e3;
tr = (0:(16*100-1))/(16*15000);


% Combine Target and Covert Signals / Desynchronizing Covert by 8
samples
totS = uC + [cD(9:end) cD(1:8)];

% Normalize so max voltage of totS is 1.
totS = totS/max(abs(totS));

% Add Reference
OFDMsig = [totS zeros(1, 4*16) cos(2*pi*fr*tr) zeros(1, 4*16)];

% Plot Real Part of the Waveform
figure
plot(real(OFDMsig))

% Saving the WF, OFDM and Covert Generated Symbols, Target and Covert
Fading
save('XmitSig', 'OFDMsig', 'tSequence1', 'tSequence2', 'tSequence3',
'cSequence', 'tFading', 'cFading')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Demodulating Received Signal   %
%    for Equipment Measurement    %
%   Ghaith Shabsigh - ITTC 2013   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all; clc;


load('XmitSig')     % Load Measured Signal
load('RcvdSig')     % Load Generated Waveform


modType = 16;            % <--- QPSK or 16QAM
R_b = 1;                % <--- 1: rb/2;
                        %      2: rb/4;
                        %      4: rb/8;
                        %      8: rb/16;


phSync = 0;                 % <--- Phase Resynchronization,
                            %       Only one value between 0 and 2pi
                            %       gives maximum correlation spike

nSub = 0;                   % <--- Subcarrier Number; 0 --> 3
                            %      0 - 0kHz
                            %      1 - 15kHz
                            %      2 - 30kHz
                            %      3 - 45kHz

seqStart = 1;      % <--- Index of Maximum Correlation Spike Furthest
to Left
startShift = 0;         % <--- Changing Sequence Beginning

magV = 300;             % <--- Constellation Magnitude Scaling- Only
Target System Affected
phAdjust = 0;           % <--- Constellation Diagram Phase Rotation
Adjustment

intRange = 0;             % <--- Controls Integration Range Over Each
Symbol


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Phase Resynchronized
IQsig = IQData.'*exp(1i*phSync);

% Plot Signal of attempted Resynch
figure(1)
plot(real(IQsig))
title('Inphase')
```

```
% Correlation Plot to Find the Beginning of a Sequence
fr = 45e3;
tr = (0:(16*100-1))/(16*15000);
zz = conv(real(IQsig), cos(2*pi*fr*tr));
display(['Max Correlation Value: ' num2str(max(zz))])
figure(2)
plot(zz)
title('Correlation')


% First Sample of the Sequence Determined
bg = 16*4 + seqStart + startShift;

% Show Real Signal For Comparison
 rxSignal = IQsig(bg:(bg+16*40000-1));
 figure(3); plot(real(rxSignal));
 figure(4); plot(real(OFDMsig));


% Get rid of Fading
if(modType==4)
    rxSignal = rxSignal./cFading;
elseif(modType==16)
    rxSignal= [rxSignal(1,end-7:end),rxSignal(1,1:end-8)];
    rxSignal = rxSignal./tFading;
else
    display('modType != 4 || 16');
end

% Frequency Shift / Down Convert Subcarrier
t = (0:(16*40000-1))/(16*15000);
fo = nSub*15000;
downSignal = rxSignal.*exp(-2i*pi*fo*t);

% Integrate over each symbol
tempp = reshape(downSignal, 16*R_b, 40000/R_b);
intSignal = mean(tempp(1:(end-intRange), :), 1);

% Plot Constellation Diagram of the Subcarrier Under Test
figure(3)
temmp = magV*intSignal*exp(1i*ph2);
plot(temmp, '.'); hold on
plot(3, 3, 'y.'); hold off
axis([-10 10 -10 10])
title('Subcarrier Constellation Diagram')

% Demodulate the Sub-carrier Sequence Under Test
demodSignal = qamdemod(temmp, modType);
```

```
% Choosing the Correct Sequence from Loaded Data for BER Analysis
if(modType == 4)
    GdemS = cSequence(1, :);
elseif(modType == 16)
    if nSub == 0
    GdemS = tSequence1(nSub+1, :);
    elseif nSub == 1
    GdemS = tSequence2(nSub+1, :);
    elseif nSub == 2
    GdemS = tSequence3(nSub+1, :);
    end
end

% Decimal to Binary
tempp1 = de2bi(GdemS.', log2(modType), 'left-msb')'; % generated
b_dData = tempp1(:);
tempp2 = de2bi(demodSignal, log2(modType), 'left-msb')'; % captured
b_deData = tempp2(:);

% BER
[N, R] = biterr(b_dData, b_deData);

display(['BER Value: ' num2str(R)])
```

%%%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generating OFDM/Covert Waveform %
%    for Equipment Measurement    %
%   Ghaith Shabsigh - ITTC 2013   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Main File

clear all; clc; close all;


Target_or_Cov = 1;                  % 0: Target, 1: Covert

SymRate = 1;                        % 1: rb/1;
                                    % 2: rb/4;
                                    % 3: rb/8;
                                    % 4: rb/16

const = -26.15;                     % Constant

cSNRo = const + 25;           % <--- Replace with desired Covert SNR

cSymo = [40e3 20e3 10e3 5e3]; % Number of Symbols, Covert
cRepo = [16 32 64 128];       % Number of Samples/Symbol, Covert
nuSymo = 40e3;                % Number of Symbols, OFDM
strr = ['vv' 'vH' 'vQ' 'vE'];

%%%%%%%%%%%%%%%%

display('Setup S.G. SNR according to table');

% Calling Function -- Generate Transmission Signal
equipOFDM(cSymo(SymRate), cRepo(SymRate), cSNRo, nuSymo, …
            Target_or_Cov, strr((2*SymRate-1):(2*SymRate)));



% NOTE1: The generated Waveform will be Saved in Current Folder.
% Name of WF Matrix can be changed in the function .

% NOTE2: Number of Samples per Symbol for the OFDM is 16.

% NOTE3: Covert Modulation is QPSK, OFDM Modulation is 16QAM

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Generating OFDM Waveform fn.  %
%    for Equipment Measurement    %
%   Ghaith Shabsigh - ITTC 2013   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function equipOFDM(cSym, cRep, cSNR, nuSym, OFDM_Cov, stro)


No = 0.03;            % Constant
sampRate = 240e3;     % S.G. Sampling Rate
S = No*db2pow(26.15); % Setting up the OFDM Signal at Some Level
C = S*db2pow(cSNR);   % Attenuating Covert Signal to Achieve Cov. SNR

t = (0:(16*nuSym-1))/(16*15000); % Time Duration of the WF


                        %%%%%%%%%%%%%%%%%%
                            %Target%

% Gen data
tSequence1 = randi([0 15], 3, nuSym); % QAM16
tSequence2 = randi([0 15], 3, nuSym); % QAM16
tSequence3 = randi([0 15], 3, nuSym); % QAM16

% Modulate / Up sample
temp = repmat(qammod(tSequence1(1, :), 16), 16, 1);
rD(1, :) = temp(:).';

temp = repmat(qammod(tSequence2(2, :), 16), 16, 1);
rD(2, :) = temp(:).';

temp = repmat(qammod(tSequence3(3, :), 16), 16, 1);
rD(3, :) = temp(:).';

% Frequency Shift / Up convert
f = 15000;
uD = rD.*[ones(1, 16*nuSym); exp(2i*pi*f*t); exp(4i*pi*f*t)];

clear('temp', 'rD', 'reD', 'imD')

% Power scaling each sub-carrier
pws1 = mean(abs(uD(1, :)).^2);
pws2 = mean(abs(uD(2, :)).^2);
pws3 = mean(abs(uD(3, :)).^2);

uD(1, :) = uD(1, :)*sqrt(S/pws1);
uD(2, :) = uD(2, :)*sqrt(S/pws2);
uD(3, :) = uD(3, :)*sqrt(S/pws3);

% Add Rayleigh Fading
tFading = repmat(sqrt(0.5*randn(1, nuSym).^2 + 0.5*randn(1, nuSym).^2),
16, 1);
tFading = tFading(:).';
```

```
% Combine all Sub-carriers and Fading into one signal
cD = sum(uD, 1).*tFading;



                        %%%%%%%%%%%%%%%%%%
                            %Covert%


% Covert
% Gen data
cSequence = randi([0 3], 1, cSym); %QPSK

% Modulate / Up sample
tempz = repmat(qammod(cSequence(1, :), 4), cRep, 1);
rC(1, :) = tempz(:).';

% Frequency Shift / Up convert
fo = 45000;
uC = rC.*exp(2i*pi*fo*t);

% Power scaling covert
pwc = mean(abs(uC).^2);
uC = uC*sqrt(C/pwc);

% Add Rayleigh Fading
cFading = repmat(sqrt(0.5*randn(1, cSym).^2 + 0.5*randn(1, cSym).^2),
cRep, 1);
cFading = cFading(:).';

uC = uC.*cFading;

% Reference Signal Variables declared
fr = 45e3;
tr = (0:(16*100-1))/(16*15000);

% Normalize so max voltage of cD, uC is 1.
cD = cD/max(abs(cD));
uC = uC/max(abs(uC));

% Because Two SGs are used, desynchronization will occur without coding.

% Add Reference to cD

    if(OFDM_Cov == 0)
         uC = [cD zeros(1, 4*16) zeros(1, 1600) zeros(1, 4*16)];
         cD = [uC zeros(1, 4*16) cos(2*pi*fr*tr) zeros(1, 4*16)];
    elseif(OFDM_Cov == 1)
         cD = [cD zeros(1, 4*16) zeros(1, 1600) zeros(1, 4*16)];
         uC = [uC zeros(1, 4*16) cos(2*pi*fr*tr) zeros(1, 4*16)];
    end
```

```
% Plot Real Part of the Waveform
figure
plot(real(cD))


% Saving the WF, OFDM and Covert Generated Symbols, OFDM and Covert
Fading
str = [];
if(OFDM_Cov == 0)
    str = 'OFDM';
elseif(OFDM_Cov == 1)
    str = 'Co';
end


save(['st' str stro 'Curve'], 'uC', 'cD', 'tSequence1', 'tSequence2',
'tSequence3', 'cSequence', 'tFading', 'cFading')
```

%%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Demodulating Received Signal   %
%    for Equipment Measurement    %
%   Ghaith Shabsigh - ITTC 2013   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all; clc;


load('XmitSig')      % Load Measured Signal
load('RcvdSig')      % Load Generated Waveform


OFDM_Cov = 1;           % <--- 0: demod OFDM, 1: demod Covert;
R_b = 1;                % <--- 1: rb/2;
                        %      2: rb/4;
                        %      4: rb/8;
                        %      8: rb/16;


phSync = 0;             % <--- Global Rotation of the Constellation
nSub = 0;               % <--- OFDM Subcarrier Number; 0 --> 2; NOT
used for Cov.

seqStart = 1;              % <--- First Sample of the Ref. Signal
startShift = 0;            % <--- Changing Seq. Beginning

magV = 160;             % <--- Constellation Magnitude Scaling
phAdjust = 0;                % <--- Subcarrier Const. Diag. Rotation


modType = [];           % <--- QPSK or 16QAM
if(OFDM_Cov == 0)
    modType = 16;
elseif(OFDM_Cov == 1)
    nSub = 3;
    modType = 4;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Phase Resynchronized
IQsig = IQData.'*exp(1i*phSync);

% Plot Signal of attempted Resynch
figure(1)
plot(real(IQsig))
title('Inphase')

% Correlation Plot to Find the Beginning of a Sequence

fr = 45e3;
tr = (0:(16*100-1))/(16*15000);
Reff = cos(2*pi*fr*tr);
```

```
zz = conv(real(IQsig), Reff);
display(['Max Correlation Value: ' num2str(max(zz))])

figure(2)
plot(zz)
title('Correlation')

%Comment break out to continue running after Resynchronization
% break

% First Sample of the Sequence Determined
bg = 16*4 + seqStart + startShift;

% Get rid of Fading
if(OFDM_Cov == 0)
    Fd = tFading;
elseif(OFDM_Cov == 1)
    Fd = cFading;
end
rxSignal = IQsig(bg:(bg+16*40000-1))./Fd;

% Down Convert Subcarrier
t = (0:(16*40000-1))/(16*15000);

fo = nSub*15000;
downSignal = rxSignal.*exp(-2i*pi*fo*t);

% Integrate over each symbol
tempp = reshape(downSignal, 16*R_b, 40000/R_b);
intSignal = mean(tempp(1:end, :), 1);

% Plot Constellation Diagram of the Subcarrier Under Test
figure(3)
temmp = magV*intSignal*exp(1i*phAdjust);
plot(temmp, '.'); hold on
plot(3, 3, 'y.'); hold off
% axis([-5 5 -5 5])
title('Subcarrier Constellation Diagram')

% Demodulate
demodSignal = qamdemod(temmp, modType);

% Choosing the Correct Sequence from Loaded Data for BER Analysis
if(modType == 4)
    GdemS = cSequence(1, :);
elseif(modType == 16)
    if nSub == 0
    GdemS = tSequence1(nSub+1, :);
    elseif nSub == 1
    GdemS = tSequence2(nSub+1, :);
    elseif nSub == 2
    GdemS = tSequence3(nSub+1, :);
    end
end
```

```
% Decimal to Binary
tempp1 = de2bi(GdemS.', log2(modType), 'left-msb')'; % generated
b_dData = tempp1(:);
tempp2 = de2bi(demodSignal, log2(modType), 'left-msb')'; % captured
b_deData = tempp2(:);

% BER
[N, R] = biterr(b_dData, b_deData);

display(['BER Value: ' num2str(R)])
```

%%