Design and Simulation of Orthogonal Frequency Division Multiplexing (OFDM)

Final Report

GURU GOBIND SINGH INDRAPASTHA UNIVERSITY

Submitted by: Dheeman Barua
University School Of Information Technology
B.Tech (IT)

Acknowledgements

The Project is intended to give each student experience in completing a sophisticated design project that spans most of the senior year. Planning, management of time, allocation of responsibility, documentation, and presentation of the results are integrated with the technical design task. The students work with one faculty advisors who have expertise in the project research area. The student is fully responsible for the design project, with the advisor(s) acting as guide and mentor. In this project Prof.B.V.R Reddy has given his full support and knowledge to make this project. I wish to acknowledg him for his valuable guidance during project formulation and completion. I wish to adcknowledge all research foundations to let me work on their premises . I wish to adcknowledge university library at GGSIPU for providing me with all the books which were needed .

Finally, I am greatful to all the people for their willing support cooperation and assistance

Abstract

MATLAB program has been written to investigate Orthogonal Frequency Division Multiplexing (OFDM) communication systems. This program can be used for future researchers simulating systems that are too theoretically complex to analyze. Single-carrier QAM and multi-carrier OFDM are compared to demonstrate the strength of OFDM in multipath channels. Two graphical user interface demonstrations show some of the basic concepts of OFDM.

About the Project

This project will focus on Orthogonal Frequency Division Multiplexing (OFDM) research, simulation, and implementation. OFDM is a modulation technique especially suitable for wireless communication due to its resistance to inter-symbol interference (ISI). After researching OFDM, simulation in MATLAB will be completed. The main part of this project will be using the simulation results as a guide to implement OFDM on a DSP board. Depending on the DSP hardware used, the MATLAB code will need to be converted to either C or Simulink. This makes our code compatible with the software tool for that particular DSP board. To test the DSP code, we will verify that the input and output vectors of the MATLAB simulation and DSP implementation correspond.

Importance

With the rapid growth of digital wireless communication in recent years, the need for high-speed mobile data transmission has increased. New modulation techniques are being implemented to keep up with the desire more communication capacity. Processing power has increased to a point where OFDM has become feasible and economical. Since many wireless communication systems being developed use OFDM, it is a worthwhile research topic. Some examples of current applications using OFDM include DSL, DAB (Digital Audio Broadcasting), HDTV broadcasting, IEEE 802.11 (wireless networking standard).

Project Description

This project consists of MATLAB simulation and DSP implementation. Using a DSP design tool such as dSpace and Simulink or TMSC60 tools, a real time OFDM transmitter will be built. Figure shows a simplified flowchart of the MATLAB simulation code.

Content

Introduction

The Project is intended to give each student experience in completing a sophisticated design project that spans most of the senior year. Planning, management of time, allocation of responsibility, documentation, and presentation of the results are integrated with the technical design task. The students work with one faculty advisors who have expertise in the project research area. The student is responsible for the design project, with the advisor(s) acting as guide and mentor.

A common problem found in high-speed communication is inter-symbol interference (ISI). ISI occurs when a transmission interferes with itself and the receiver cannot decode the transmission correctly. For example, in a wireless communication system such as that shown in Figure 1, the same transmission is sent in all directions.

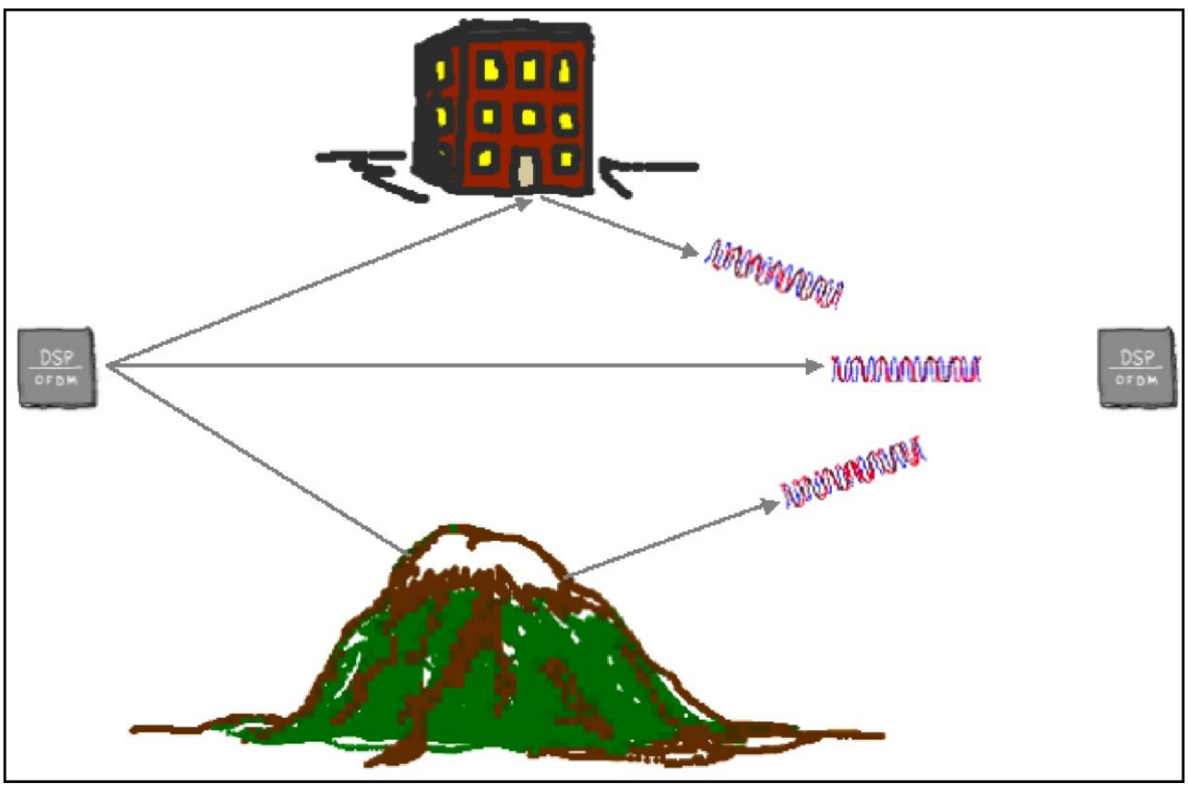


Figure 1: Multipath Demonstration

Because the signal reflects from large objects such as mountains or buildings, the receiver sees more than one copy of the signal. In communication terminology, this is called multipath. Since the indirect paths take more time to travel to the receiver, the delayed copies of the signal interfere with the direct signal, causing ISI.

Theory

This project will focus on Orthogonal Frequency Division Multiplexing (OFDM) research and simulation. OFDM is especially suitable for high-speed communication due to its resistance to ISI. As communication systems increase their information transfer speed, the time for each transmission necessarily becomes shorter. Since the delay time caused by multipath remains constant, ISI becomes a limitation in high-data-rate communication [1]. OFDM avoids this problem by sending many low speed transmissions simultaneously. For example, Figure 2 shows two ways to transmit the same four pieces of binary data.

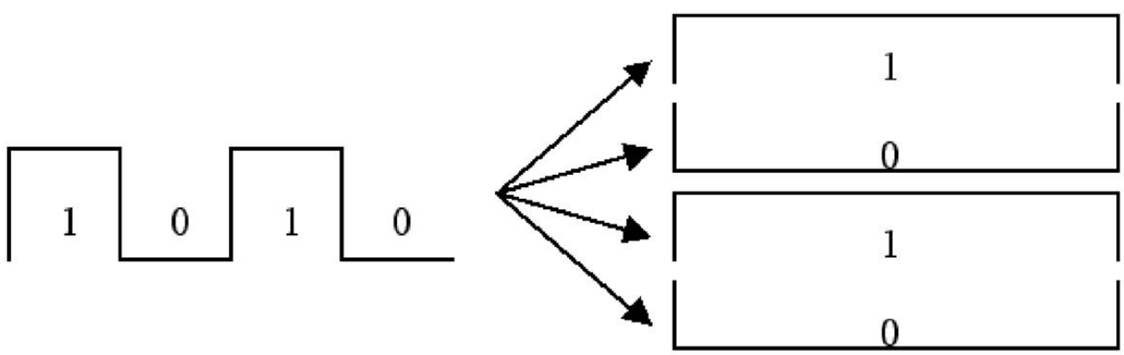


Figure 2: Traditional vs. OFDM Communication

Suppose that this transmission takes four seconds. Then, each piece of data in the left picture has a duration of one second. On the other hand, OFDM would send the four pieces simultaneously as shown on the right. In this case, each piece of data has a duration of four seconds. This longer duration leads to fewer problems with ISI. Another reason to consider OFDM is low-complexity implementation for high-speed systems compared to traditional single carrier techniques [2].

Significance

With the rapid growth of digital communication in recent years, the need for high-speed data transmission has increased. New multicarrier modulation techniques such as OFDM are currently being implemented to keep up with the demand for more communication capacity. Multicarrier communication systems "were first conceived and implemented in the 1960s, but it was not until their all-digital implementation with the FFT that their attractive features were unraveled and sparked widespread interest for adoption in various single-user and multiple access (MA) communication standards" [2]. The processing power of modern digital signal processors has increased to a point where OFDM has become feasible and economical. Examining the patents, journal articles, and books available on OFDM, it is clear that this technique will have an impact on the future of communication. See the references section (starting on page 21) for a condensed bibliography and list of patents related to this topic. Since many communication systems being developed use OFDM, it is a worthwhile research topic. Some examples of current applications using OFDM include GSTN (General Switched Telephone Network), Cellular radio, DSL & ADSL modems, DAB (Digital Audio Broadcasting) radio, DVB-T (Terrestrial Digital Video Broadcasting), HDTV broadcasting, HYPERLAN/2 (High Performance Local Area Network standard), and the wireless networking standard IEEE 802.11 [1] [3] [4].

Simulation Design

This project consists of research and simulation of an OFDM communication system. Figure 3 shows a simplified flowchart of the MATLAB simulation code.

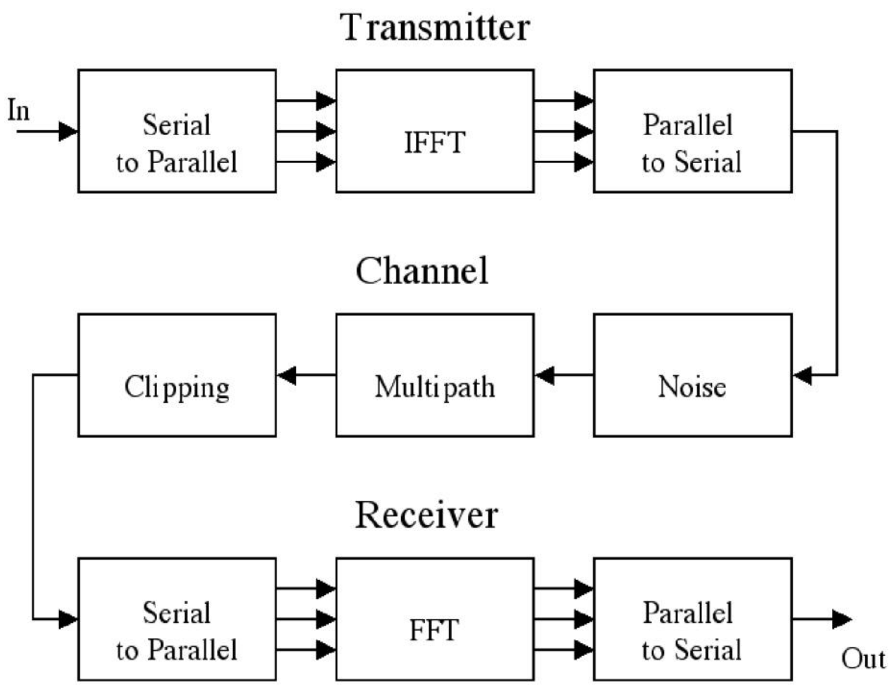


Figure 3: OFDM Simulation Flowchart

The transmitter first converts the input data from a serial stream to parallel sets. Each set of data contains one symbol, S_i, for each subcarrier. For example, a set of four data would be [S_0 S_1 S_2 S_3]. Before performing the Inverse Fast Fourier Transform (IFFT), this example data set is arranged on the horizontal axis in the frequency domain as shown in Figure 4.

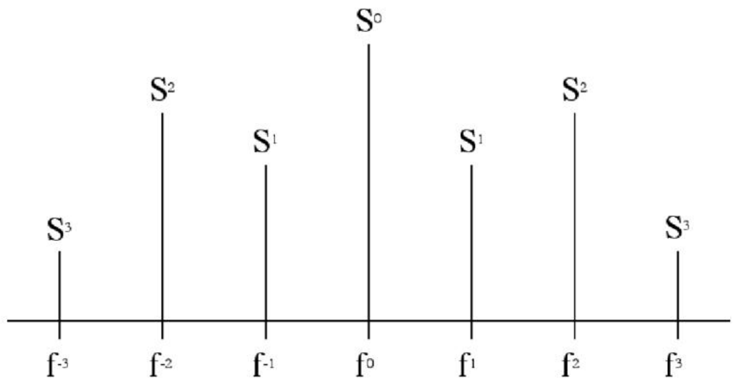


Figure 4: Frequency Domain Distribution of Symbols

This symmetrical arrangement about the vertical axis is necessary for using the IFFT to manipulate this data. An inverse Fourier transform converts the frequency domain data set into samples of the corresponding time domain representation of this data. Specifically, the IFFT is useful for OFDM because it generates samples of a waveform with frequency components satisfying orthogonality conditions. Then, the parallel to serial block creates the OFDM signal by sequentially outputting the time domain samples.

The channel simulation allows examination of common wireless channel characteristics such as noise, multipath, and clipping [5]. By adding random data to the transmitted signal, simple noise is simulated. Multipath simulation involves adding attenuated and delayed copies of the transmitted signal to the original. This simulates the problem in wireless communication when the signal propagates on many paths. For example, a receiver may see a signal via a direct path as well as a path that bounces off a building. Finally, clipping simulates the problem of amplifier saturation. This addresses a practical implementation problem in OFDM where the peak to average power ratio is high.

The receiver performs the inverse of the transmitter. First, the OFDM data are split from a serial stream into parallel sets. The Fast Fourier Transform (FFT) converts the time domain samples back into a frequency domain representation. The magnitudes of the frequency components correspond to the original data. Finally, the parallel to serial block converts this parallel data into a serial stream to recover the original input data.

Results

The MATLAB simulation accepts inputs of text or audio files as well as binary, sinusoidal, or random data. It then generates the corresponding OFDM transmission, simulates a channel, attempts to recover the input data, and performs an analysis to determine the transmission error rate. In order to compare OFDM to a traditional single carrier communication system, a 16-QAM simulation can be performed. These simulations are dynamic, allowing the user to set parameters determining the characteristics of the communication system. Two simple demonstrations of OFDM communication were developed with a graphical user interface (GUI) following the style of MATLAB toolbox demonstrations. These allow someone to quickly learn the basic concepts of OFDM communication.

The first demonstration, basicgui (or basicgui_win), introduces the process of creating an OFDM symbol. It shows a simple example of using the Fourier transform to send binary data on four frequencies. The following screenshots show the demo sequence with explanations in the text box.

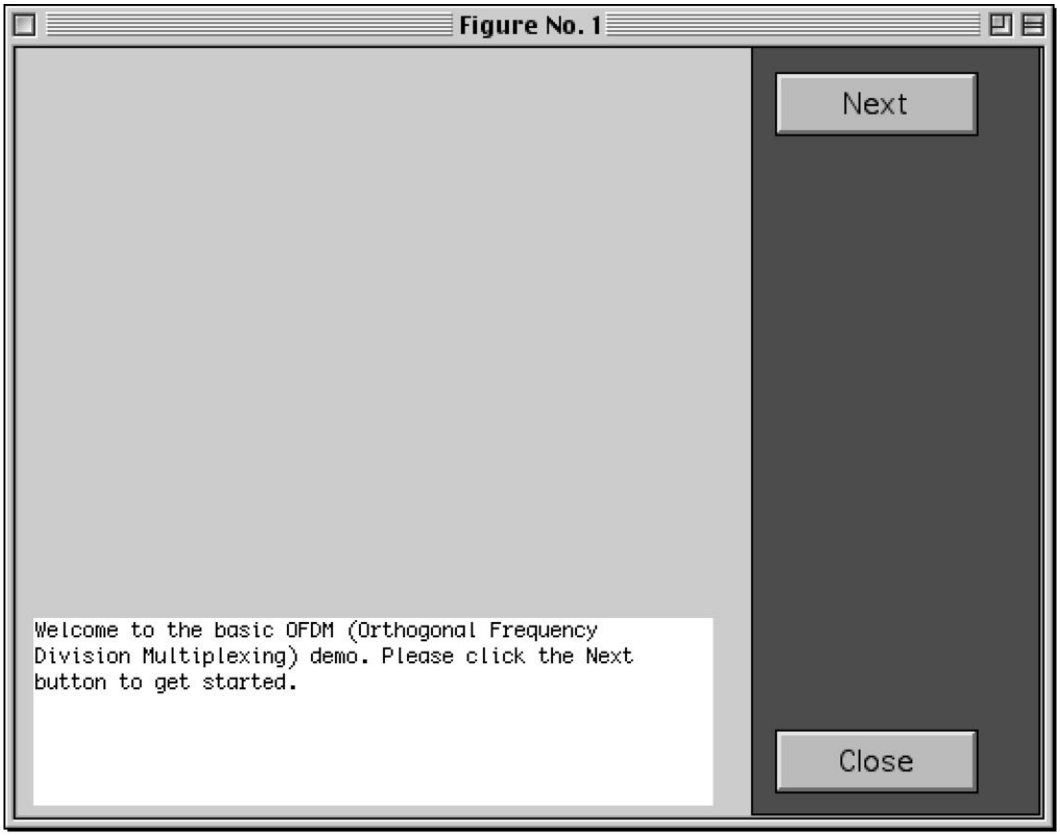


Figure No. 1
Next
Welcome to the basic OFDM (Orthogonal Frequency Division Multiplexing) demo. Please click the Next button to get started.
Close

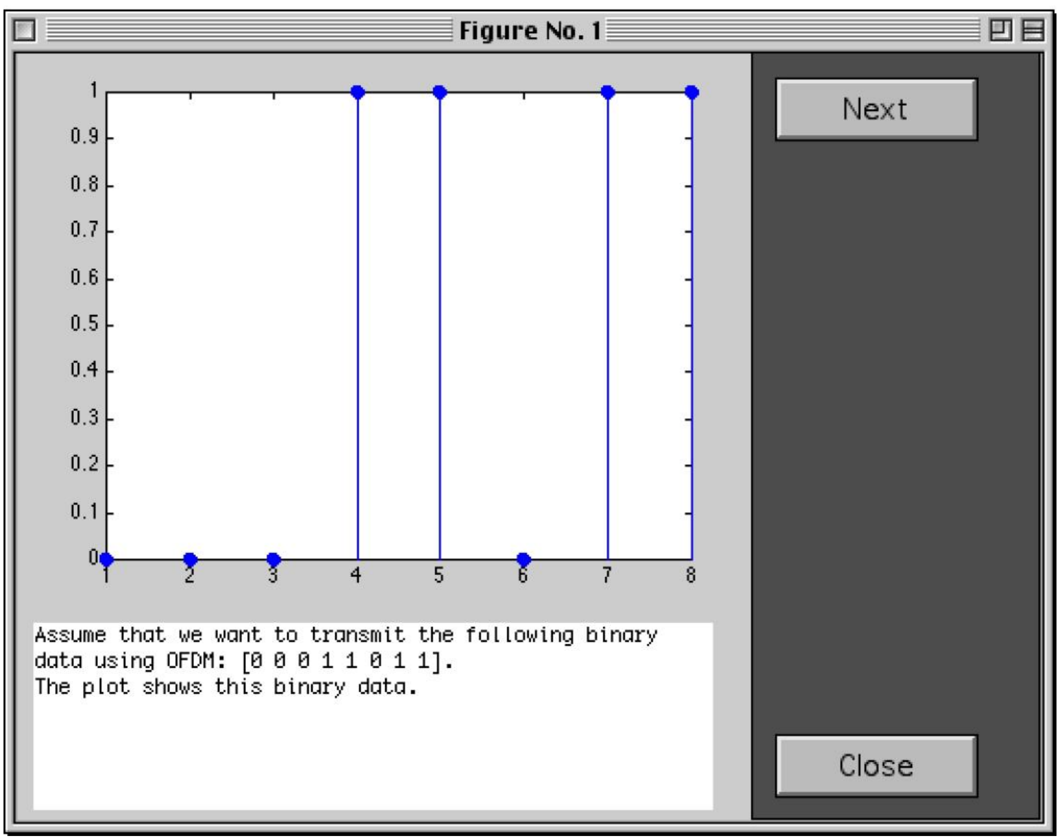


Figure No. 1
Next
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
Assume that we want to transmit the following binary data using OFDM: [0 0 0 1 1 0 1 1].
The plot shows this binary data.
Close

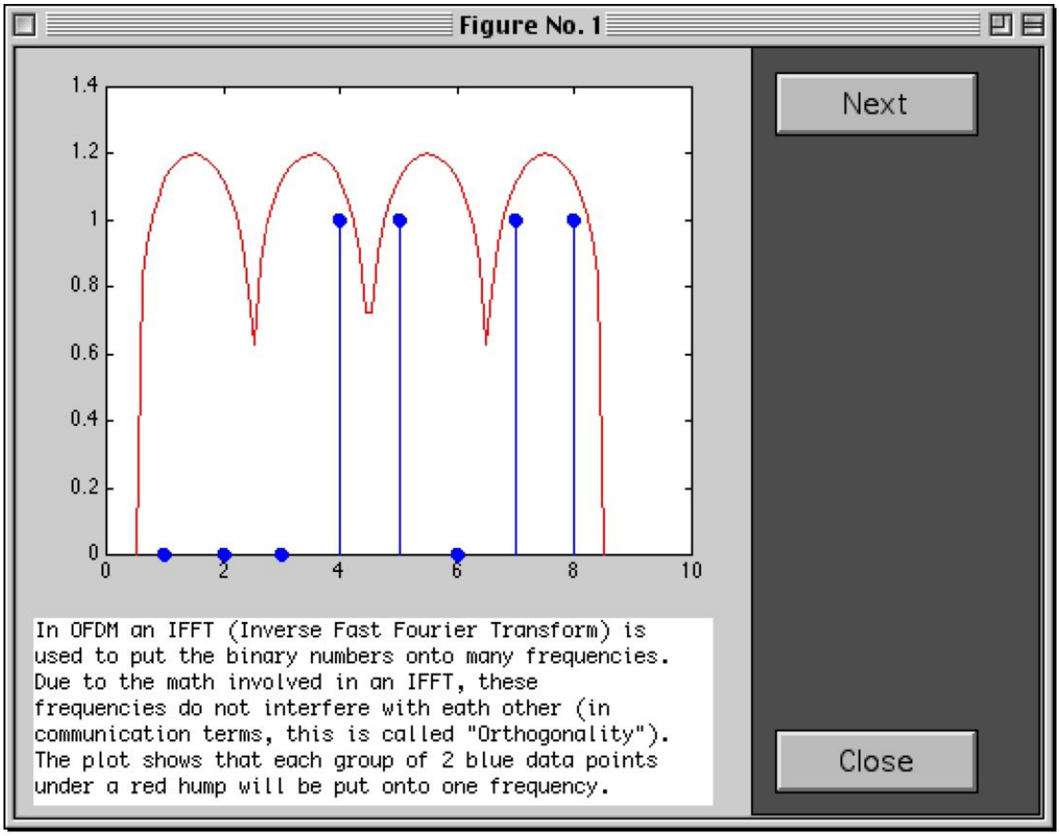


Figure No. 1
Next
Close
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
In OFDM an IFFT (Inverse Fast Fourier Transform) is
used to put the binary numbers onto many frequencies.
Due to the math involved in an IFFT, these
frequencies do not interfere with eath other (in
communication terms, this is called "Orthogonality").
The plot shows that each group of 2 blue data points
under a red hump will be put onto one frequency.

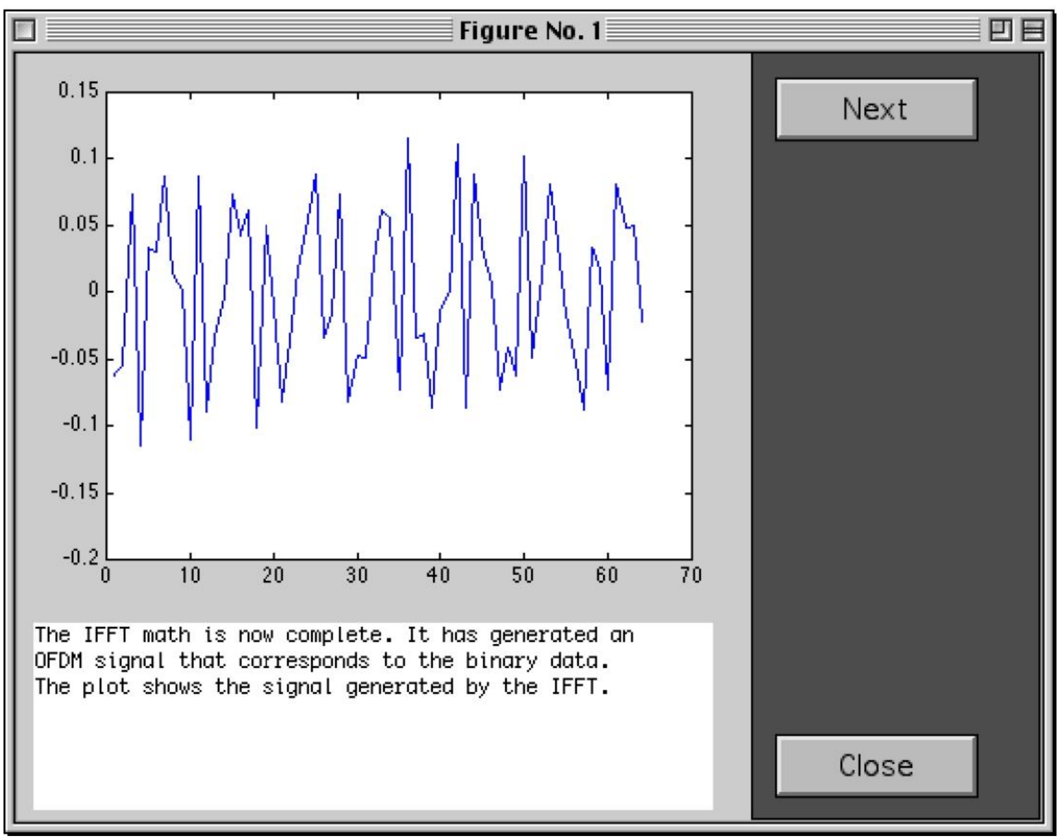


Figure No. 1
Next
Close
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
10
20
30
40
50
60
70
The IFFT math is now complete. It has generated an
OFDM signal that corresponds to the binary data.
The plot shows the signal generated by the IFFT.

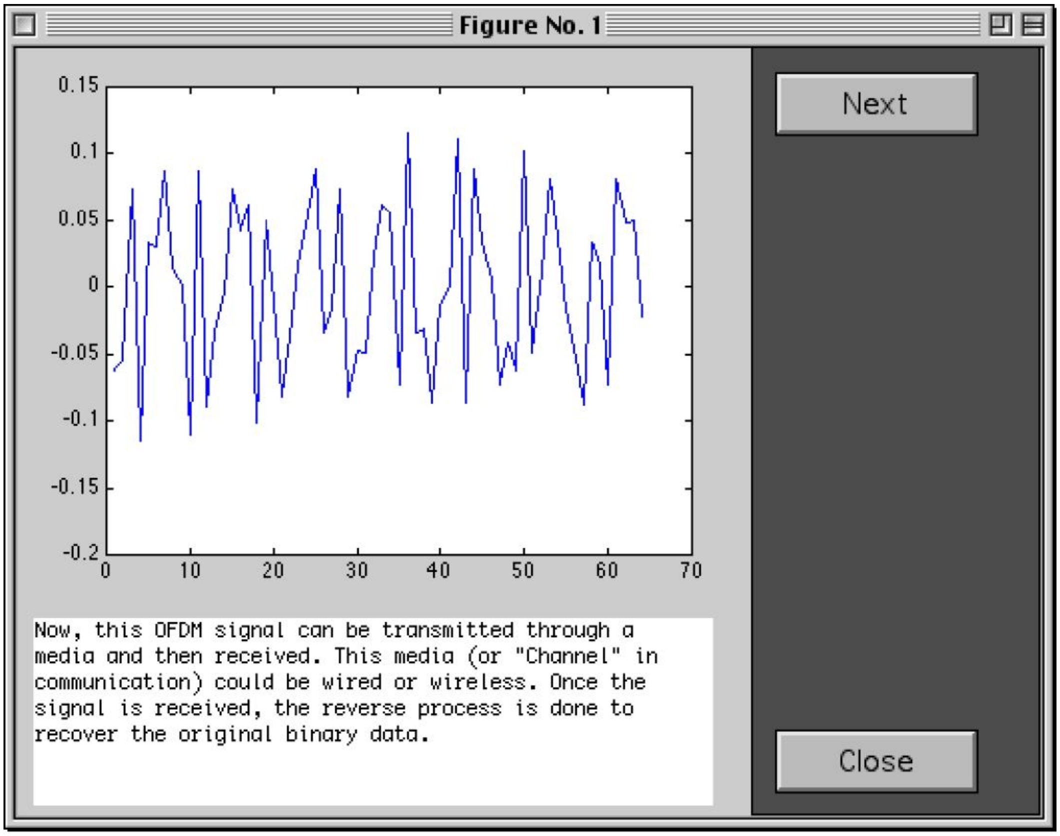


Figure No. 1
Next
Close
Now, this OFDM signal can be transmitted through a media and then received. This media (or "Channel" in communication) could be wired or wireless. Once the signal is received, the reverse process is done to recover the original binary data.

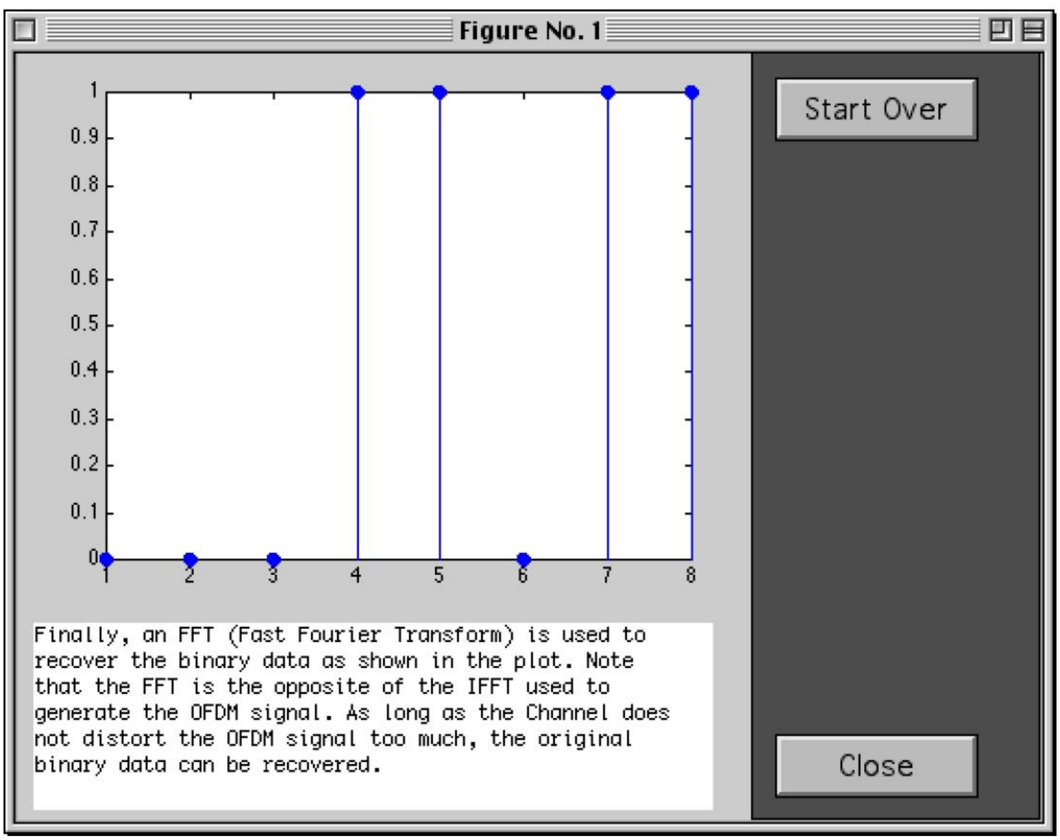
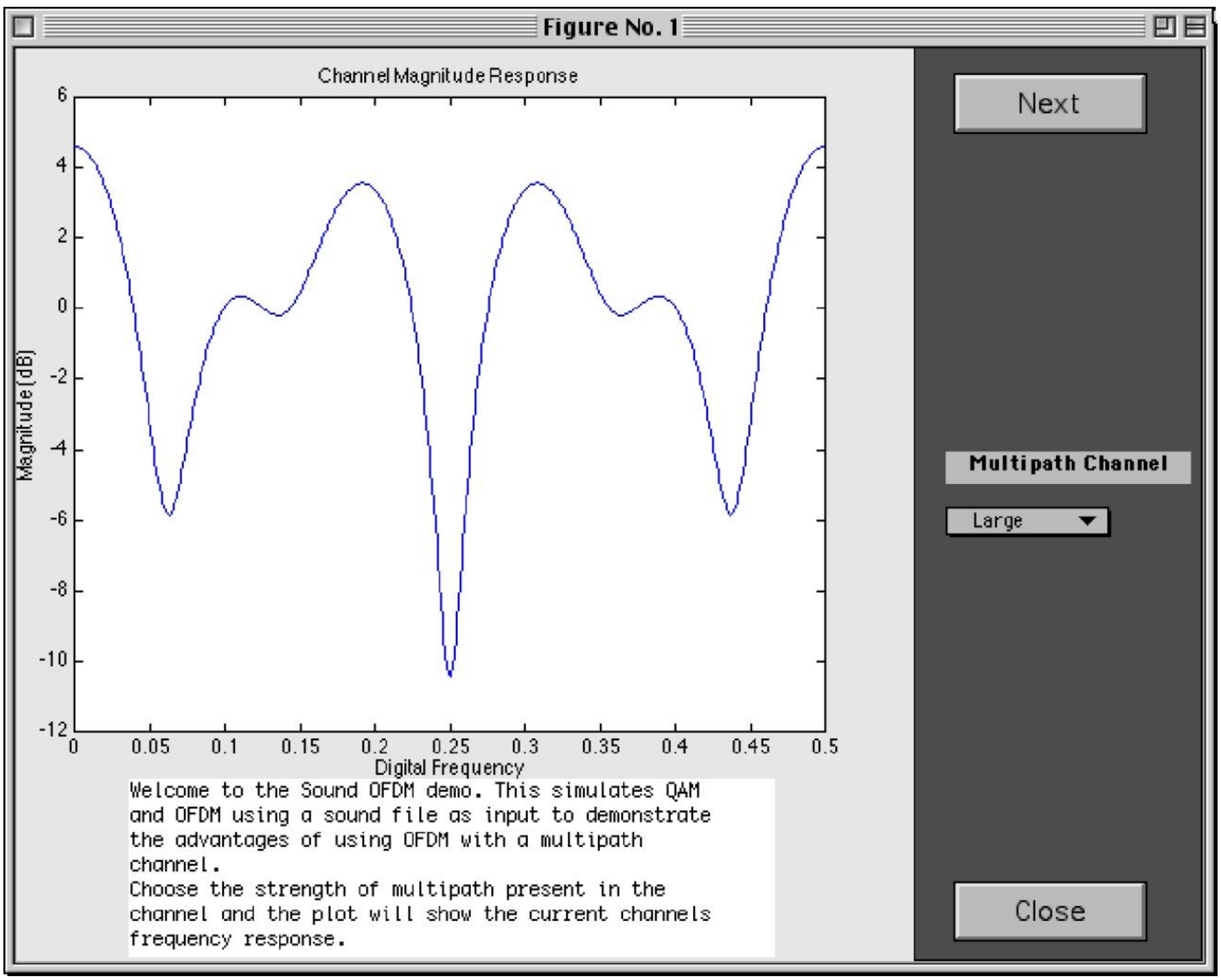


Figure No. 1
Start Over
Close
Finally, an FFT (Fast Fourier Transform) is used to recover the binary data as shown in the plot. Note that the FFT is the opposite of the IFFT used to generate the OFDM signal. As long as the Channel does not distort the OFDM signal too much, the original binary data can be recovered.

The second demonstration, soundgui (or soundgui_win), gives a more technical example. It compares OFDM to 16-QAM in a multipath channel. The user can choose no, small, or large amount of multipath. The following screenshots show the demo sequence with explanations in the text box.



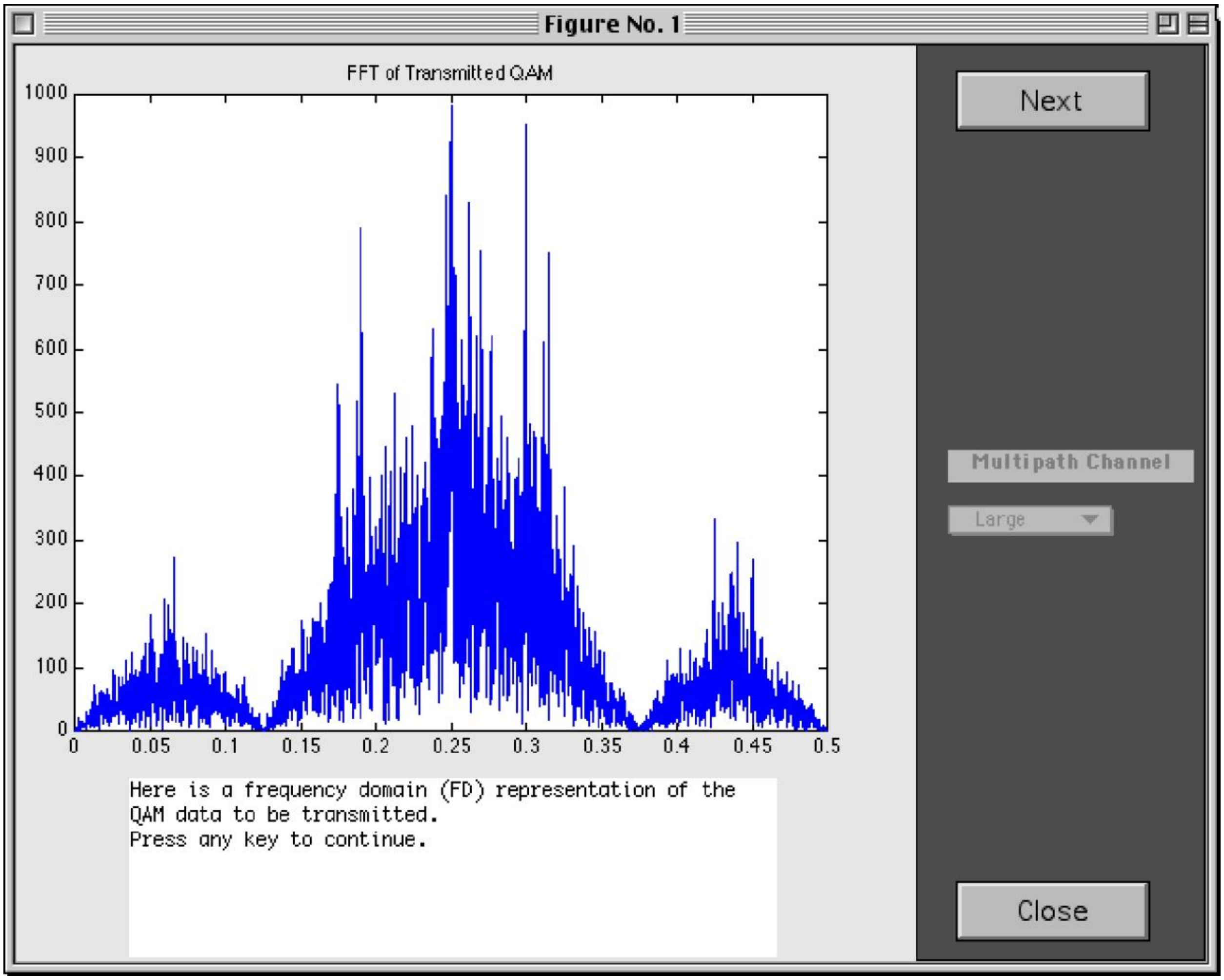


Figure No. 1
FFT of Transmitted QAM
1000
900
800
700
600
500
400
300
200
100
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Here is a frequency domain (FD) representation of the
QAM data to be transmitted.
Press any key to continue.
Next
Multipath Channel
Large
Close

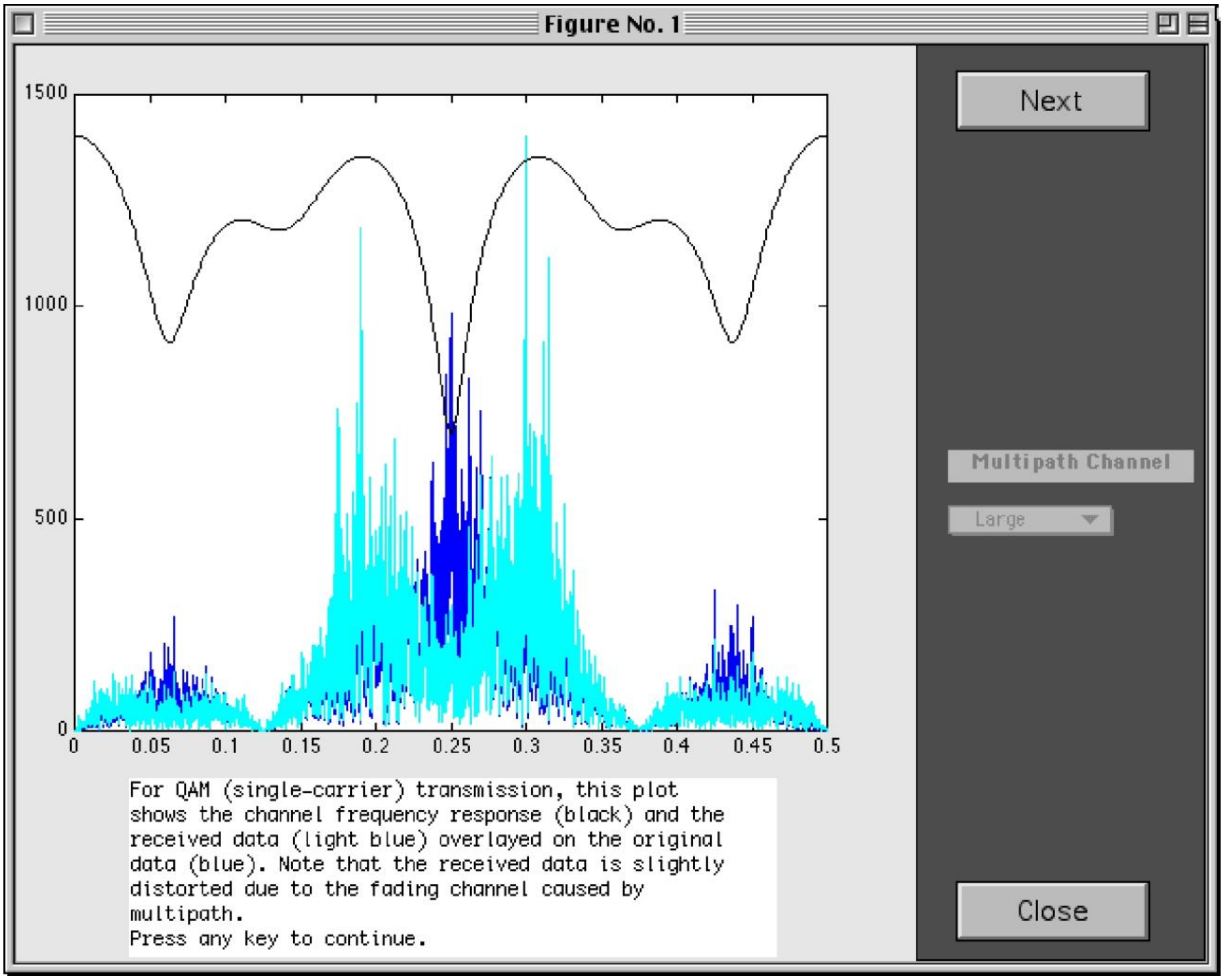


Figure No. 1
Next
Multipath Channel
Large
Close
1500
1000
500
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
For QAM (single-carrier) transmission, this plot shows the channel frequency response (black) and the received data (light blue) overlayed on the original data (blue). Note that the received data is slightly distorted due to the fading channel caused by multipath.
Press any key to continue.

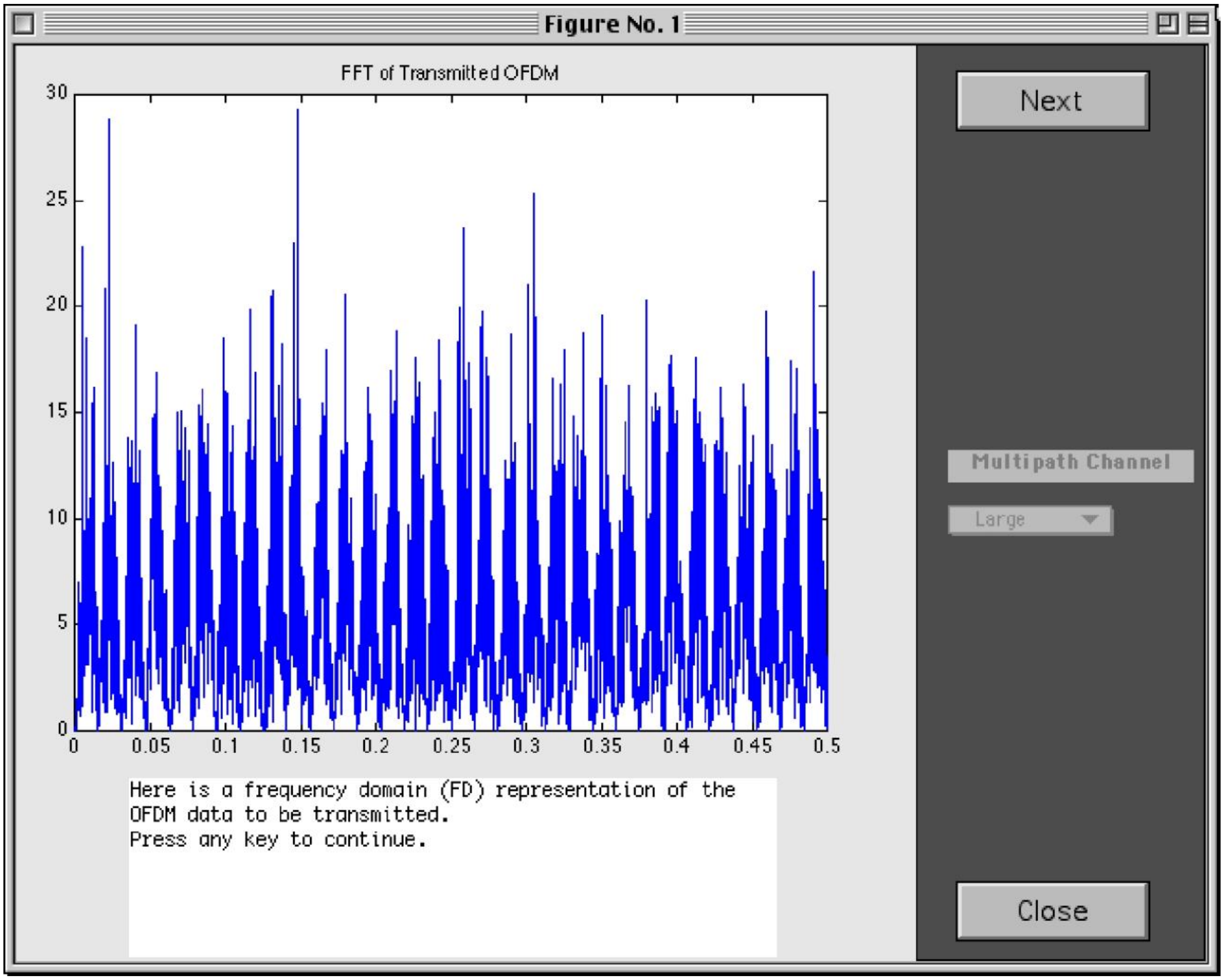


Figure No. 1
FFT of Transmitted OFDM
30
25
20
15
10
5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Here is a frequency domain (FD) representation of the
OFDM data to be transmitted.
Press any key to continue.
Next
Multipath Channel
Large
Close

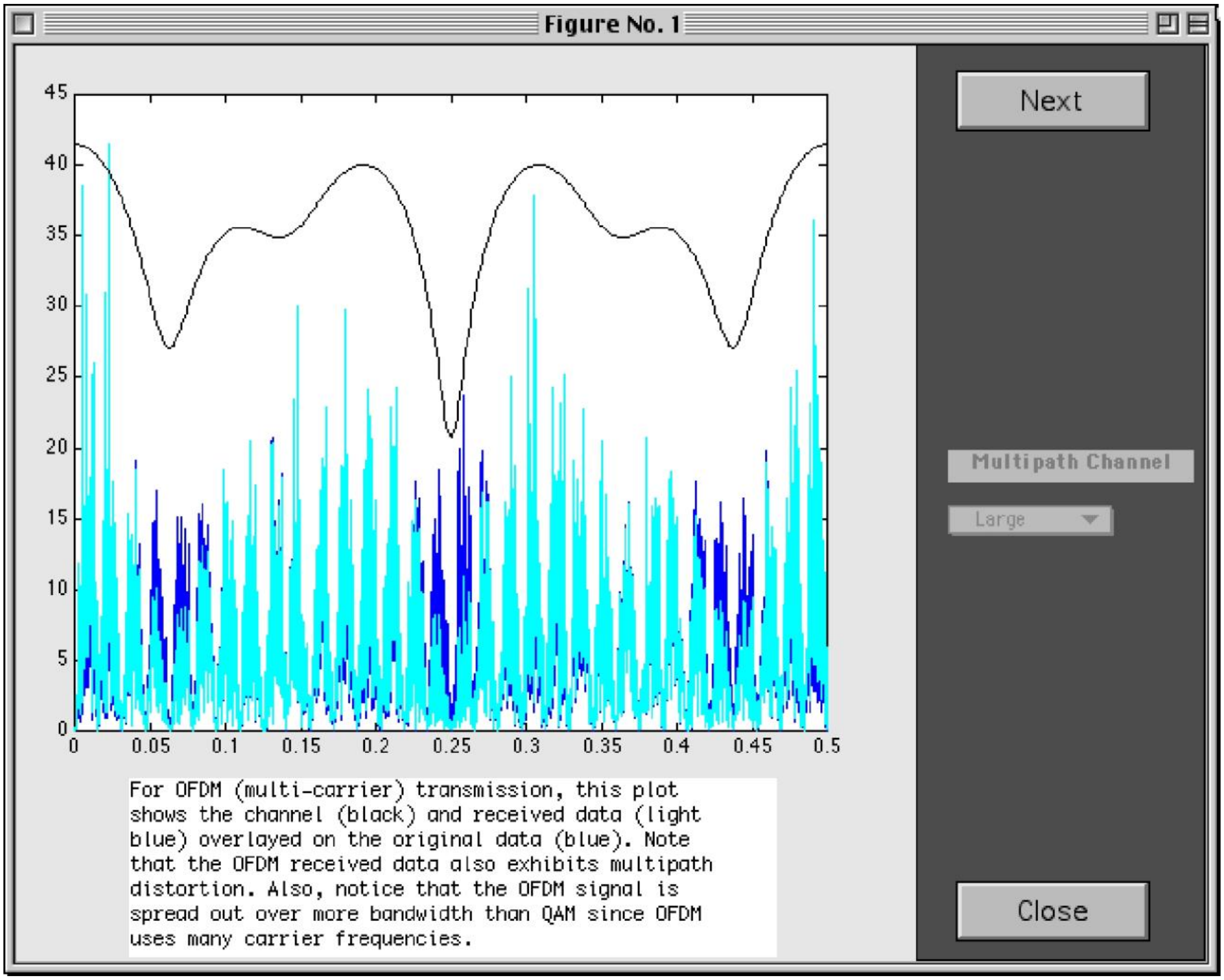
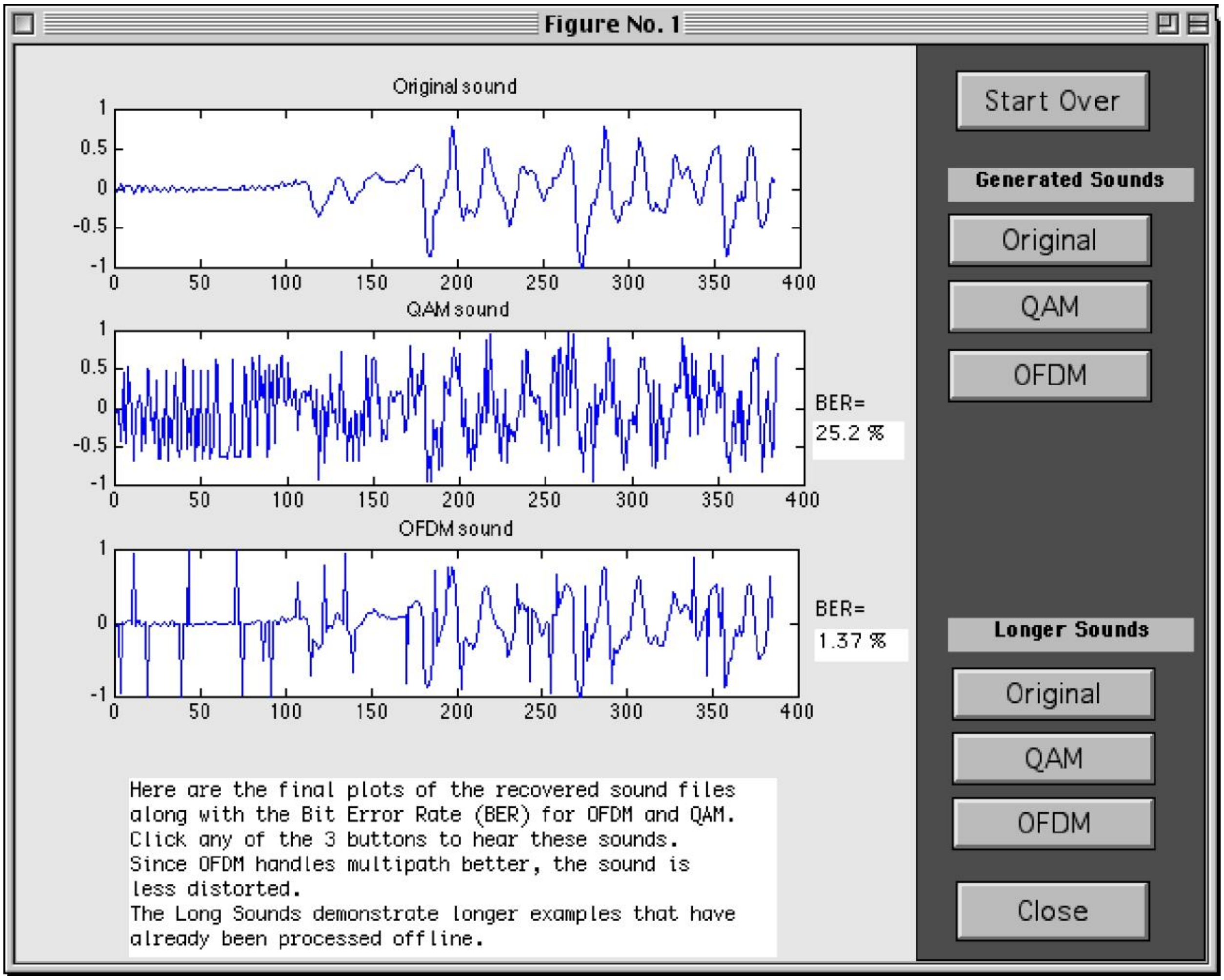


Figure No. 1
Next
Multipath Channel
Large
Close
45
40
35
30
25
20
15
10
5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
For OFDM (multi-carrier) transmission, this plot shows the channel (black) and received data (light blue) overlayed on the original data (blue). Note that the OFDM received data also exhibits multipath distortion. Also, notice that the OFDM signal is spread out over more bandwidth than QAM since OFDM uses many carrier frequencies.



The two GUI demonstrations utilize the complete simulation code, but not all of its capabilities. By modifying the setup.m m-file, users can adjust parameters such as the fft_size, num_carriers, input types, and channel characteristics. It also allows detailed analysis of the communication system. Plots showing OFDM input and output, 16-QAM input and output, and the received 16-QAM signal constellation are generated. See Figures 5, 6, and 7 for examples of these plots.

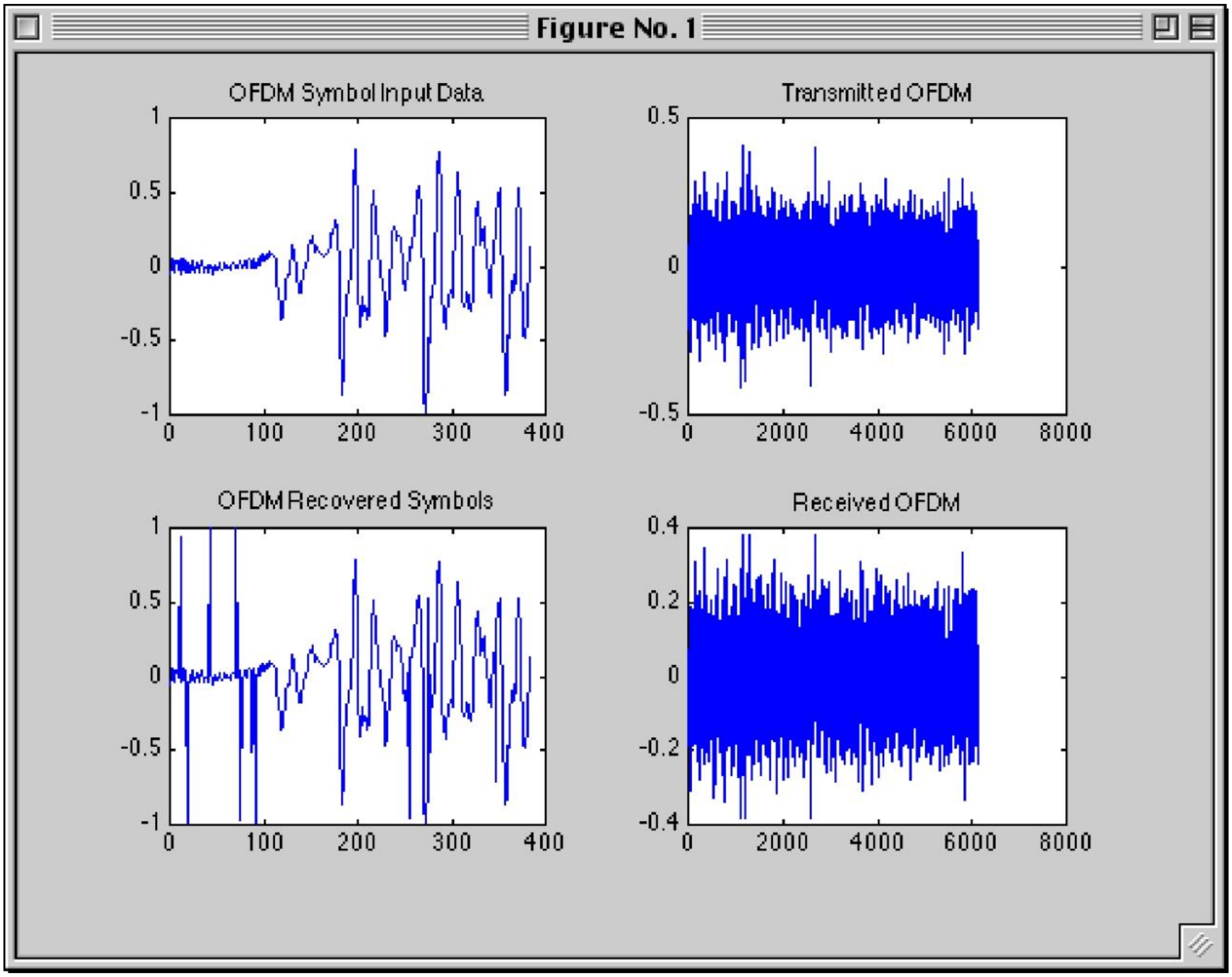


Figure 5: OFDM Input and Output

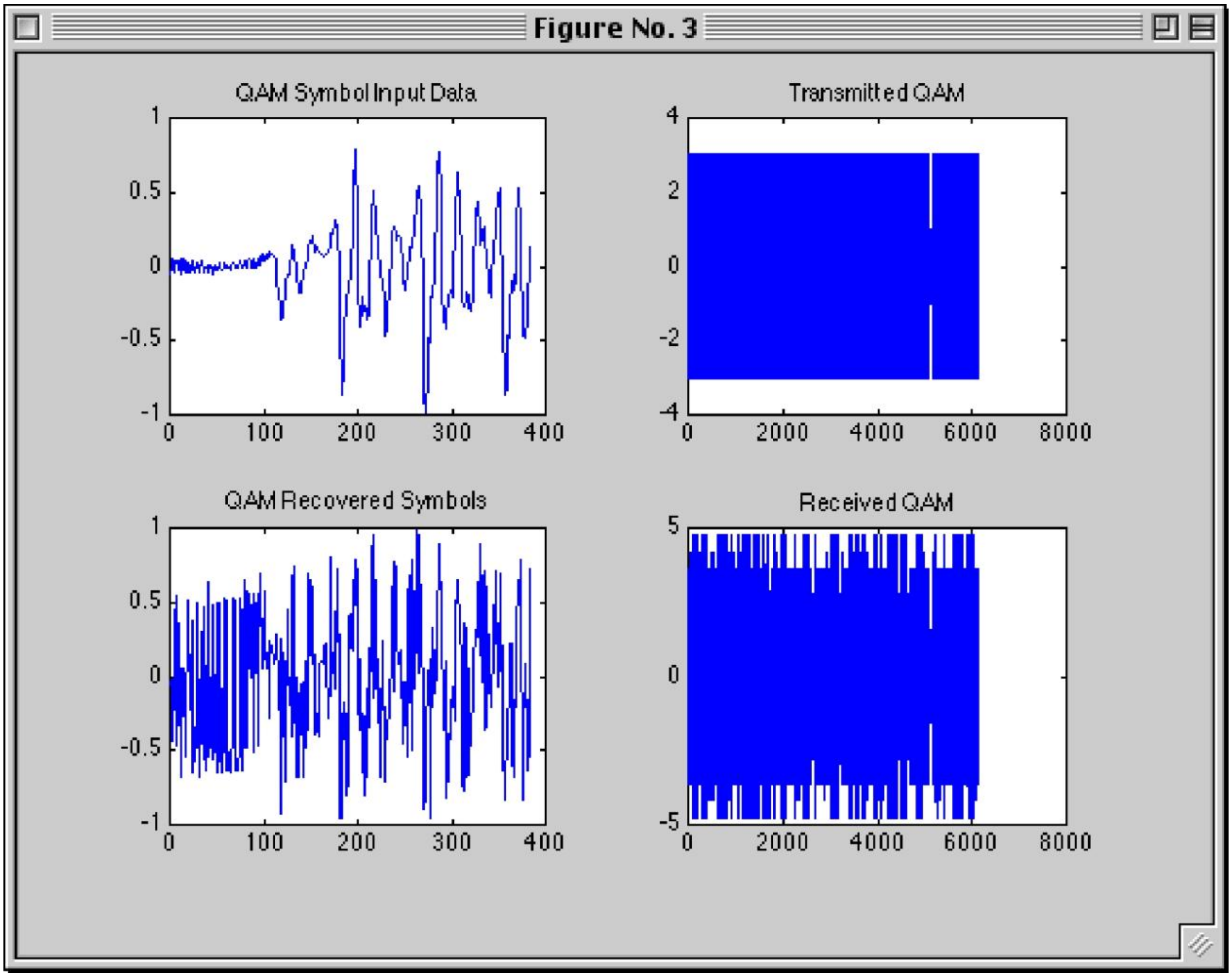


Figure 6: 16-QAM Input and Output

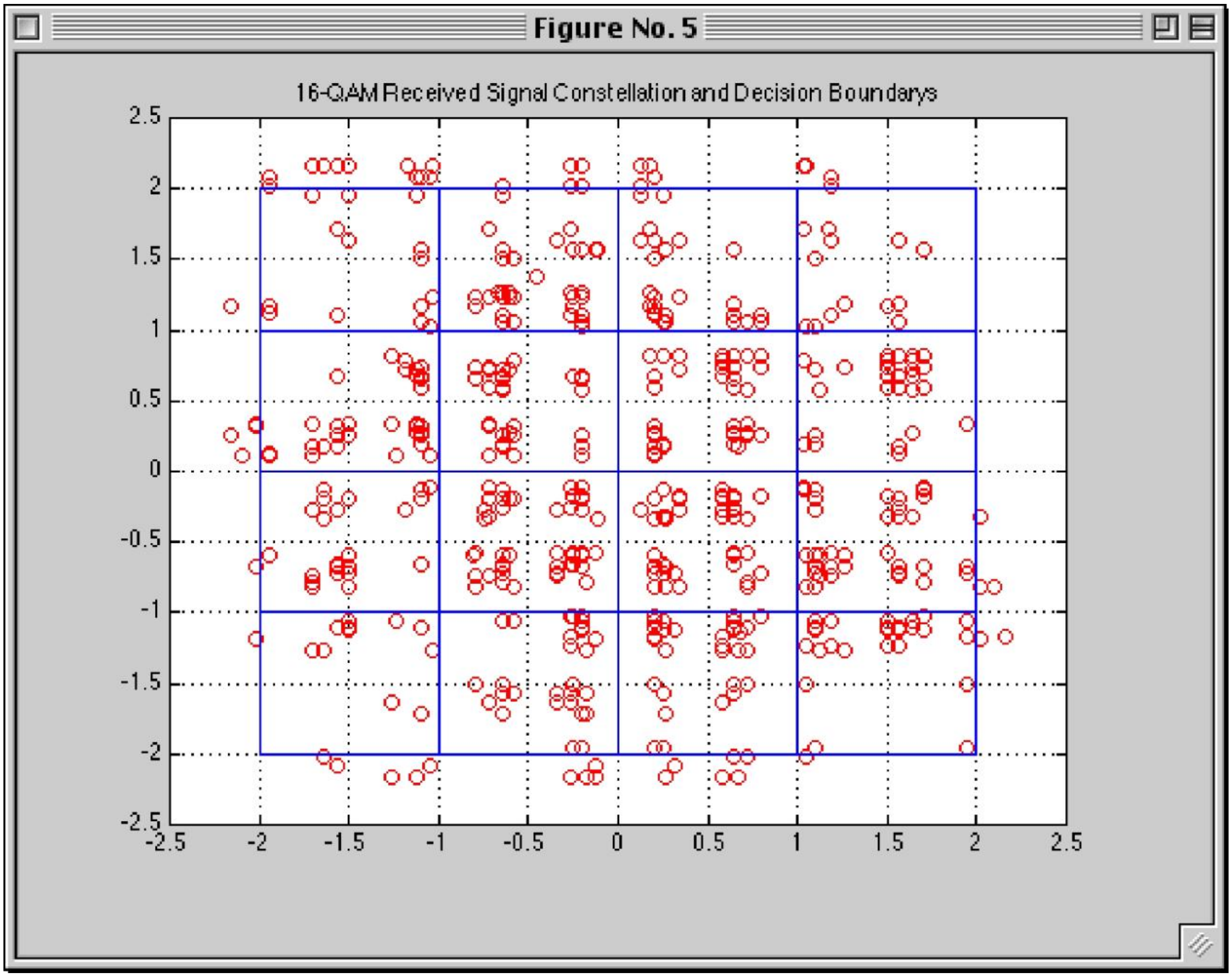


Figure 7: Received 16-QAM Signal Constellation

Depending on the input type chosen, appropriate output files are created. This enhances the numerical error analysis by showing how the errors degrade the data being transmitted. For one test, some test was transmitted. Figure 8 shows the results.

<table>
<tr><th>The Original Data</th><th>OFDM transmission
Bit Error Rate = 0.0699%
Binary Errors = 4</th><th>16-QAM transmission
Bit Error Rate = 23.0%
Binary Errors = 1,315</th></tr>
<tr><td>IN CONGRESS, July 4, 1776.

The unanimous Declaration of the thirteen united States of America,

When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness.</td><td>IN CONGRESS, July 4, 1776.

The unanimous Declaration of the thirteen united States of America,

When in the Course of human events, it becomes necessary for one people to dissolve the political baods which have connected thel with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

We hold these truÙ hs to be self-evident, that all meo are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness.</td><td>JO$iè__BÜSÉ\4õtl8$x<4tz*).___x
u$¥n!Æ)m/$rTïucú1±°¥yo_$ø_$¥
xu$¥xyq¥uen$¥n)tud4£§q¥urTø_$
í≠%q©cël___Xun$ πn$¥xu$ìü$q ≤ï
$ø_$∏tm!Æ$µuen4r\4 πt4≤•cü_%r
Tæ%cïr¢ë± ∏$∂o!dø_%_0µo
º5$¥ _¥yr¢ü_5e$¥xu$ ∞ø_9tycë¨4
≤°Æ$rT ∂Xycò4 ∏qµe$ ≥ü_.%c§ud
4¥xum_6Ytx4±Æ/$xuql4±Æ$4¥
_± ≤¢§m%$± ≠/_'_¥xu$∞ø&Uq≤Tø
_$¥xu$µa±¥x<4¥xu_2ïp±±°¥u$±
Æ$4µp¥a¨4 ≤§q¥yo_$¥ _ ∂Xycò4¥
xu$ù1 ∂bTø_$ü!¥tq•$±Æ$4ø___!¥t
q•+bTó__4µn4yt|5$¥xum,4±d¥ucï
n44±•r†µc§4¥ _¥xu$ø
πn)o_2Tø__-
!Æ+Yn$4±•p¥iq•rT¥xq¥4¥xux$ ≤ò
$l44¥ucú1±•$¥xu$ ≥ë¥rïrT ∂Xycò
4πm0µl_4xum$¥ _¥xu$ ≤ïp±±°¥y
o_.___U$∏ _44¥xurï$¥q¥txrT¥
_≤•$≤ïl6
%uidun4<4¥xq¥4±¨<4 Ω%n$±±•$ ≥
°•a¥ud_%p¥a¨<4¥xq¥4¥xux$±±•$
µn$ &Ud4 ≤∏$¥xuiqdì°•a¥ !d ∂Y
tx4≥ïq¥q©n$¥n!¨9en!¢¨5__©gXtr\
4¥xq¥4± ≠/_'_¥xurï$±±•$ù9fe,4ù9b
•q¥x$±Æ$4¥xu$ ∞¥q≤§it4ø_$ôq ∞
∞πn%r¢^_f</td></tr>
</table>

Figure 8: Text Example Comparing OFDM to 16-QAM

OFDM transmission had a very low bit error rate of 0.0699% so only four errors were caused by the multipath channel. 16-QAM incurred a 23.0% bit error rate. Since a character is represented by eight bits, every character had two bits in error on average. This resulted in unintelligible received text.

A second test using an audio file produced similar results. The difference is that users can see and hear the degradation caused by binary errors. Figure 9 shows plots of the audio files.

The Original Sound	*OFDM transmission* *Bit Error Rate = 0.01%* *Binary Errors = 17*	*16-QAM transmission* *Bit Error Rate = 21.2%* *Binary Errors = 35,957*

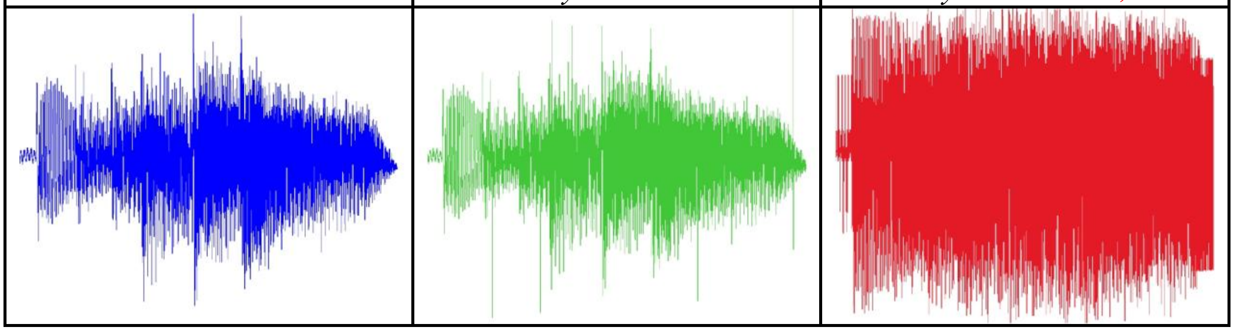


Figure 9: Audio Example Comparing OFDM to 16-QAM

In this case, the original sound is a guitar plucking a chord. The OFDM sound contains audible "clicks" due to bit errors and the waveform is similar to that of the original sound. The 16-QAM sound's waveform does not resemble the original and listening to the 16-QAM sound confirms this. The original guitar chord is barely discernable underneath loud static noise.

Conclusion

This MATLAB simulation proves that OFDM is better suited to a multipath channel than a single carrier transmission technique such as 16-QAM.

Future research may be based on this project. These extensions may include channel phase shift detection and correction, error correction by coding, adaptive transmission, peak to average power ratio considerations, and DSP implementation.

References

Bibliography

[1] Keller, Thomas, and Lajos Hanzo. "Adaptive Multicarrier Modulation: A Convenient Framework for Time-Frequency Processing in Wireless Communications." *IEEE Proceedings of the IEEE* 88 (May, 2000): 609-640

[2] Wang, Zhengdao, and Georgios B. Giannakis. "Wireless Multicarrier Communications." *IEEE Signal Processing Magazine* (May, 2000): 29-48

[3] Bingham, John A. C. "Multicarrier Modulation for Data Transmission: An Idea Whose Time Has Come." *IEEE Communications Magazine* (May, 1990): 5-14

[4] Van Nee, Richard, and Ramjee Prasad. *OFDM for Wireless Multimedia Communications*. Boston: Artech House, 2000.

[5] Naguib, Ayman F., Nambi Seshadri, and A. R. Calderbank. "Increasing Data Rate over Wireless Channels." *IEEE Signal Processing Magazine* (May, 2000): 76-92

[6] Mitra, Sanjit K. *Digital Signal Processing: A Computer-Based Approach*. New York: McGraw-Hill, 2001.

[7] O'Leary, Seamus. *Understanding Digital Terrestrial Broadcasting*. Massachusetts: Artech House, 2000.

[8] Bahai, Ahmad R. S., and Burton R. Saltzberg. *Multi-Carrier Digital Comunications:Theory and Applications of OFDM*. New York: Kluwer Academic/Plenum Publishers, 1999.

[9] Lawrey, Eric. OFDM Wireless Technology. 11 May 2000. 7 Nov. 2000. http://www.eng.jcu.edu.au/eric/thesis/Thesis.htm

References (cont.)

Patent History

Class/Subclass

370	Multiplex Communications
370/203	Generalized Orthogonal or Special Mathematical Techniques
370/208	Particular set of orthogonal functions (subset of 203)
708	Electrical Computers: Arithmetic Processing and Calculating
708/400	Transform (subset of 200)
708/403	Fourier (subset of 400)
708/404	Fast Fourier Transform (subset of 403)

Historical

3,488,4555 Orthogonal Frequency Division Multiplexing (Jan 6, 1970)

Current

<u>370/208</u>

6,125,124 Synchronization and sampling frequency in an apparatus receiving OFDM modulated transmissions (Sept 26, 2000)

6,021,110 OFDM timing and frequency recovery system (Feb 1, 2000)

<u>708/404</u>

6,115,728 Fast fourier transforming apparatus and method, variable bit reverse circuit, inverse fast fourier transforming apparatus and method, and OFDM receiver and transmitter (Sept 5, 2000)

Appendix

Complete MATLAB source code

Code Statistics: 1,969 lines; 6,635 words; 46,742 characters (omitting spaces)

% a_filter_design.m

```
% Design filter by specifying delay in units and
% looking at mag and phase response
                                % Good default values for fft_size = 128 and num_carriers = 32
delay_1      = 6;                                          %              6
attenuation_1 = 0.35;                              %              0.35
delay_2      = 10;                                         %              10
attenuation_2 = 0.30;                              %              0.30

num = [1, zeros(1, delay_1-1), attenuation_1, zeros(1, delay_2-delay_1-1), attenuation_2];
[H, W] = freqz(num, 1, 512);          % compute frequency response
mag = 20*log10(abs(H));                           % magnitude in dB
phase = angle(H) * 180/pi;                  % phase angle in degrees
figure(9), clf
subplot(211), plot(W/(2*pi),mag) title('Magnitude response of
multipath channel')
xlabel('Digital Frequency'), ylabel('Magnitude in dB') subplot(212),
plot(W/(2*pi),phase)
title('Phase response of multipath channel') xlabel('Digital Frequency'),
ylabel('Phase in Degrees')

break



% Design filter using MATLAB command 'fir2' nn = 40; %
order of filter
f = [0, 0.212, 0.253, 0.293, 0.5]; m =[1, 1, 0.5, 1,
1];
num = fir2(nn, 2*f, m); den = 1;


[H, W] = freqz(num, den, 256);                 % Compute freq response
mag = 20*log10(abs(H));               % Get mag in dB
phase = angle(H)*180/pi;              % Get phase in degrees

clf
subplot(211), plot(W/(2*pi),mag) subplot(212),
plot(W/(2*pi),phase)

break
% Design filter using MATLAB command 'fir1' % These coeffs
work well for OFDM vs. QAM!!!
% nn = 4; % order of filter
% wl = 0.134;          % low cutoff of     stopband
% wh = 0.378;          % high cutoff of stopband
% nn = 4; % order of filter
% wl = 0.195;          % low cutoff of     stopband
% wh = 0.309;          % high cutoff of stopband

nn =  8;    % order of filter
wl =  0.134;           % low cutoff of     stopband
wh = 0.378;            % high cutoff of stopband
num = fir1(nn, 2*[wl, wh], 'stop');
den = 1;

[H, W]  = freqz(num,    den, 256);            % Compute freq response
mag = 20*log10(abs(H));               % Get  mag in dB
phase = angle(H)*180/pi;              % Get  phase in degrees

clf
subplot(211), plot(W,mag), hold on, plot(wl*2*pi,0,'o'), plot(wh*2*pi,0,'o') subplot(212), plot(W,phase), hold on,
plot(wl*2*pi,0,'o'), plot(wh*2*pi,0,'o') hold off


break
% Design filter by specifying delay in units and looking at mag and phase response n = 512;


d1 =4;
a1 = 0.2;
```

```
d2 = 5;
a2 = 0.3;

num = [1, zeros(1, d1-1), a1, zeros(1, d2-d1-1), a2] den = [1];


[H, W] = freqz(num, den, n);
% F = 0:.1:pi;
% H = freqz(num, den, F*180/pi, 11025);

mag = 20*log10(abs(H));
% phase = angle(H * 180/pi); phase =
angle(H);

clf
subplot(211), plot(W,mag), hold on, plot(0.17*pi,0,'o'), plot(0.34*pi,0,'o') subplot(212), plot(W,phase), hold on,
plot(pi/2,0,'o')
hold off

break
% Design filter by specifying mag response at particular frequencies

n = 2;
f = [0, 0.25, 0.5];
mag = [1, .05, 1];

[num, den] = yulewalk(n,2*f,mag);

[H, W] = freqz(num, den);

mag = 20*log10(abs(H));
phase = angle(H * 180/pi);

clf
subplot(211), plot(W,mag)
subplot(212), plot(W,phase)
```

% a_run_demo.m

```
setup

QAM

OFDM

analysis
```

% analysis.m

```
% Analysis

disp(' '), disp('------------------------------------------------------------')
disp('Preparing Analysis')

figure(1), clf
if (input_type == 1) & (test_input_type == 1)
        subplot(221), stem(data_in), title('OFDM Binary Input Data'); subplot(223), stem(output),
        title('OFDM Recovered Binary Data')
else
        subplot(221), plot(data_samples), title('OFDM Symbol Input Data'); subplot(223),
        plot(output_samples), title('OFDM Recovered Symbols');
end
subplot(222), plot(xmit), title('Transmitted OFDM'); subplot(224), plot(recv),
title('Received OFDM');


% dig_x_axis = (1:length(QAM_tx_data))/length(QAM_tx_data);
%       figure(4), clf, subplot(212)
%       freq_data = abs(fft(QAM_rx_data));
%       L = length(freq_data)/2;

dig_x_axis = (1:length(xmit))/length(xmit); figure(2), clf


if channel_on ==1
        num = [1, zeros(1, d1-1), a1, zeros(1, d2-d1-1), a2]; den = [1];

        [H, W] = freqz(num, den, 512);
```

```
        mag = 20*log10(abs(H)); phase =
        angle(H) * 180/pi;

        subplot(313)
        freq_data = abs(fft(recv)); L =
        length(freq_data)/2;
        plot(dig_x_axis(1:L), freq_data(1:L)) xlabel('FFT of
        Received OFDM') axis_temp = axis;


        subplot(311),
        freq_data  =  abs(fft(xmit));
        plot(dig_x_axis(1:L), freq_data(1:L)), axis(axis_temp) title('FFT of Transmitted
        OFDM')

        subplot(312)
        plot(W/(2*pi),mag),
        ylabel('Channel  Magnitude  Response')
else
        subplot(212)
        freq_data = abs(fft(recv)); L =
        length(freq_data)/2;
        plot(dig_x_axis(1:L), freq_data(1:L)) xlabel('FFT of
        Received OFDM') axis_temp = axis;


        subplot(211),
        freq_data  =  abs(fft(xmit));
        plot(dig_x_axis(1:L), freq_data(1:L)), axis(axis_temp) title('FFT of Transmitted
        OFDM')
end

% if  file_input_type  ==  4
%       figure(5)
%       subplot(211)
%       image(data_in);
%       colormap(map);
%       subplot(212)
%       image(output);
%       colormap(map);
% end

if  do_QAM  ==  1               %  analyze  if  QAM  was  done

        figure(3),  clf
        if  (input_type  ==  1)  &  (test_input_type  ==  1)
                subplot(221), stem(data_in), title('QAM Binary Input Data'); subplot(223), stem(QAM_data_out),
                title('QAM Recovered Binary Data')
        else
                subplot(221), plot(data_samples), title('QAM Symbol Input Data'); subplot(223),
                plot(QAM_output_samples), title('QAM Recovered Symbols');
        end
        subplot(222), plot(QAM_tx_data), title('Transmitted QAM'); subplot(224),
        plot(QAM_rx_data), title('Received QAM');

        dig_x_axis = (1:length(QAM_tx_data))/length(QAM_tx_data); figure(4), clf


        if channel_on ==1 subplot(313)

                freq_data = abs(fft(QAM_rx_data)); L =
                length(freq_data)/2;
                plot(dig_x_axis(1:L), freq_data(1:L)) xlabel('FFT of
                Received QAM') axis_temp = axis;


                subplot(311),
                freq_data = abs(fft(QAM_tx_data)); plot(dig_x_axis(1:L),freq_data(1:L)),
                axis(axis_temp) title('FFT of Transmitted QAM')


                subplot(312)
                plot(W/(2*pi),mag)
                ylabel('Channel  Magnitude  Response')
        else
                subplot(212)
                freq_data = abs(fft(QAM_rx_data)); L =
                length(freq_data)/2;
                plot(dig_x_axis(1:L), freq_data(1:L)) title('FFT of
                Received QAM')
```

```
                axis_temp = axis;

                subplot(211),
                freq_data = abs(fft(QAM_tx_data)); plot(dig_x_axis(1:L),freq_data(1:L)),
                axis(axis_temp) title('FFT of Transmitted QAM')

        end

        % Plots the QAM Received Signal Constellation
        figure(5), clf, plot(xxx,yyy,'ro'), grid on, axis([-2.5 2.5 -2.5 2.5]), hold on

%       % Overlay plot of transmitted constellation
%       x_const = [-1.5 -0.5 0.5 1.5 -1.5 -0.5 0.5 1.5 -1.5 -0.5 0.5 1.5 -1.5 -0.5 0.5 1.5];
%       y_const = [-1.5 -1.5 -1.5 -1.5 -0.5 -0.5 -0.5 -0.5 0.5 0.5 0.5 0.5 1.5 1.5 1.5 1.5];
%       plot(x_const, y_const, 'b*')

        % Overlay of constellation boundarys
        x1 = [-2 -2]; x2 = [-1 -1]; x3 = [0 0]; x4 = [1 1]; x5 = [2 2]; x6 = [-2 2]; y1 = [-2 -2]; y2 = [-1 -1]; y3 = [0 0]; y4 =
        [1 1]; y5 = [2 2]; y6 = [-2 2]; plot(x1,y6), plot(x2,y6), plot(x3,y6), plot(x4,y6), plot(x5,y6)

        plot(x6,y1), plot(x6,y2), plot(x6,y3), plot(x6,y4), plot(x6,y5)

        hold off
        title('16-QAM Received Signal Constellation and Decision Boundarys')

        binary_err_bits_QAM = 0; for i =
        1:length(data_in)
                err = abs(data_in(i)-QAM_data_out(i)); if err > 0

                        binary_err_bits_QAM = binary_err_bits_QAM + 1;
                end
        end
        BER_QAM = 100 * binary_err_bits_QAM/data_length;
end

figure(6), clf
if channel_on == 1
        subplot(211), plot(W/(2*pi),mag),title('Channel Magnitude Response') xlabel('Digital
        Frequency'),ylabel('Magnitude in dB')
        subplot(212), plot(W/(2*pi),phase),title('Channel Phase Response') xlabel('Digital
        Frequency'),ylabel('Phase in Degrees')
else
        title('Channel is turned off - No frequency response to plot')
end

% Compare output to input and count errors
binary_err_bits_OFDM = 0;
for i = 1:length(data_in)
        err = abs(data_in(i)-output(i)); if err > 0

                binary_err_bits_OFDM = binary_err_bits_OFDM +1;
        end
end
BER_OFDM = 100 * binary_err_bits_OFDM/data_length; disp(strcat('OFDM:
BER=', num2str(BER_OFDM,3), ' %'))
disp(strcat('              Number of error bits=', num2str(binary_err_bits_OFDM)))

if (do_QAM == 1)
        disp(strcat('QAM:   BER=', num2str(BER_QAM,3), ' %'))
        disp(strcat('              Number of error bits=', num2str(binary_err_bits_QAM)))
end

% Display text file before and after modulation if (input_type == 2) &
(file_input_type == 2)
        original_text_file = char(data_samples') if do_QAM ==1

                edit QAM_text_out.txt
        end
        edit OFDM_text_out.txt
end

% Listen to sounds
if (input_type == 2) & (file_input_type == 3) do_again = '1';

        while ( ~(isempty(do_again)) ) disp(' ')

                disp('Press any key to hear the original sound'), %pause
                sound(data_samples,11025)
                disp('Press any key to hear the sound after OFDM transmission'), %pause sound(output_samples,11025)
```

```
                    if do_QAM == 1
                              disp('Press any key to hear the sound after QAM transmission'), %pause
                              sound(QAM_output_samples,11025)
                    end
                    do_again = '';
                    do_again = input('Enter "1" to hear the sounds again or press "Return" to end   ', 's');
          end
end
```

% BasicGUI.m

```
function BasicGUI()
% This is the machine-generated representation of a MATLAB object
% and its children.  Note that handle values may change when these
% objects are re-created. This may cause problems with some callbacks.
% The command syntax may be supported in the future, but is currently
% incomplete and subject to change.
%
% To re-open this system, just type the name of the m-file at the MATLAB
% prompt. The M-file and its associtated MAT-file must be on your path.

load BasicGUI

a = figure('Color',[0.8 0.8 0.8], ...
          'Colormap',mat0, ...
          'CreateFcn','OFDMguiFn figure', ...
          'Position',[490 321 512 384], ...
          'Resize','off', ...
          'Tag','Fig1');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[1 1 1], ...
          'FontName','Monaco', ...
          'HorizontalAlignment','left', ...
          'Position',[8 5 340 94], ...
          'String','Basic OFDM Demo', ...
          'Style','text', ...
          'Tag','StaticTextFeedback');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0.3 0.3 0.3], ...
          'Position',[367 0 147 387], ...
          'Style','frame', ...
          'Tag','Frame1');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0.733333 0.733333 0.733333], ...
          'Callback','OFDMguiFn next', ...
          'FontSize',14, ...
          'Position',[379 340 102 32], ...
          'String','Next', ...
          'Tag','PushbuttonNext');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0.733333 0.733333 0.733333], ...
          'Callback','OFDMguiFn close', ...
          'FontSize',14, ...
          'Position',[379 11 102 32], ...
          'String','Close', ...
          'Tag','PushbuttonClose');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0.733333 0.733333 0.733333], ...
          'Position',[379 248 58 26], ...
          'String',mat1, ...
          'Style','popupmenu', ...
          'Tag','PopupMenu1', ...
          'Value',2, ...
          'Visible','off');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0.733333 0.733333 0.733333], ...
          'FontWeight','bold', ...
          'Position',[379 283 129 17], ...
          'String','Number of Carriers', ...
          'Style','text', ...
          'Tag','StaticText2', ...
          'Visible','off');
b = axes('Parent',a, ...
```

```
        'Units','points', ...
        'CameraUpVector',[0 1 0], ...
        'CameraUpVectorMode','manual', ...
        'Color',[1 1 1], ...
        'ColorOrder',mat2, ...
        'Position',[44 130 294 235], ...
        'Tag','AxesMain', ...
        'Visible','off', ...
        'XColor',[0 0 0], ...
        'YColor',[0 0 0], ...
        'ZColor',[0 0 0]);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[0.5 -0.0662393 0], ...
        'Tag','Text1', ...
        'VerticalAlignment','cap', ...
        'Visible','off');
set(get(c,'Parent'),'XLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[-0.0767918 0.502137 0], ...
        'Rotation',90, ...
        'Tag','Text2', ...
        'VerticalAlignment','baseline', ...
        'Visible','off');
set(get(c,'Parent'),'YLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','right', ...
        'Position',[-0.151877 1.08333 0], ...
        'Tag','Text3', ...
        'Visible','off');
set(get(c,'Parent'),'ZLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[0.5 1.0235 0], ...
        'Tag','Text4', ...
        'VerticalAlignment','bottom', ...
        'Visible','off');
set(get(c,'Parent'),'Title',c);
```

% bin2eight.m

```
function y = bin2eight(x)

% bin2eight
%
% Converts binary data to an eight bit form
% Accepts 1x8 array and returns the corresponding decimal

y = 0; k
= 0;
for i = 1:8
        y = y + x(8-k)*2^k; k =
        k+1;
end
```

% bin2pol.m

```
function y = bin2pol(x)

% bin2pol
% Converts binary numbers (0,1) to polar numbers (-1,1)
% Accepts a 1-D array of binary numbers

y = ones(1,length(x)); for i =
1:length(x)
        if x(i) == 0
                y(i) = -1;
        end
end
```

% ch.m

```
% ch

recv = xmit;            % channel is applied to recv, don't modify transmitted data

if channel_on == 1
        disp('Simulating Channel')

        norm_factor = max(abs(recv)); % Normalize all data before applying
        recv = (1/norm_factor) * recv;          % channel for a fair comparison

        ch_clipping

        ch_multipath

        ch_noise

        recv = norm_factor * recv;              % Restore data magnitude for proper decoding

end
```

% ch_clipping.m

```
% ch_clipping

for i = 1:length(recv)
        if recv(i) > clip_level
                recv(i) = clip_level;
        end
        if recv(i) < -clip_level
                recv(i) = -clip_level;
        end
end
```

% ch_multipath.m

```
% ch_multipath

copy1=zeros(size(recv)); for
i=1+d1:length(recv)
        copy1(i)=a1*recv(i-d1);
end

copy2=zeros(size(recv)); for
i=1+d2:length(recv)
        copy2(i)=a2*recv(i-d2);
end

recv=recv+copy1+copy2;
```

% ch_noise.m

```
% ch_noise (operate on recv)
% random noise defined by noise_level amplitude

if already_made_noise == 0 % only generate once and use for both QAM and OFDM noise = (rand(1,length(recv))-
        0.5)*2*noise_level;
        already_made_noise = 1;
end
recv = recv + noise;
```

% ComputeChannelGUI.m

```
% ComputeChannelGUI.m    plots the current channel

popupHnd1=findobj('Tag','PopupMenuMultipath');
    noChannel = 0;
if get(popupHnd1,'Value') == 3                              % Large
            d1   = 6;
            a1   = 0.4;
            d2   = 10;
            a2   = 0.3;
elseif get(popupHnd1,'Value') == 2                  % Small
            d1   = 6;
            a1   = 0.25;
            d2   = 10;
            a2   = 0.20;
else                                                                                              % None
            noChannel = 1;
            channel_on = 0;
            break
end
num = [1, zeros(1, d1-1), a1, zeros(1, d2-d1-1), a2];
den = [1];
[H, W] = freqz(num, den);
mag = 20*log10(abs(H));
phase = angle(H) * 180/pi;
% plot(W/(2*pi),mag)                     % comment me out normally
```

% eight2bin.m

```
function y = eight2bin(x)

% eight2bin
%
% Converts eight bit data (0-255 decimal) to a binary form for processing.

y = zeros(1,8); k = 0;

while x > 0
            y(8-k) = rem(x,2); k =
            k+1;
            x = floor(x/2);
end
```

% OFDM.m

```
% Run OFDM simulation

tic            % Start stopwatch to calculate how long QAM simulation takes
disp(' '),disp('------------------------------------------------------------')
disp('OFDM Simulation')

tx
ch
rx

% Stop stopwatch to calculate how long QAM simulation takes
OFDM_simulation_time = toc;
if OFDM_simulation_time > 60
            disp(strcat('Time for OFDM simulation=', num2str(OFDM_simulation_time/60), ' minutes.'));
else
            disp(strcat('Time for OFDM simulation=', num2str(OFDM_simulation_time), ' seconds.'));
end
```

% OFDMguiFn.m

```
function OFDMguiFn(action)
% Consolidates all of the GUI callbacks into one main function

stringArray = [...
        % Slide 1
        'Welcome to the basic OFDM (Orthogonal Frequency                '...
        'Division Multiplexing) demo. Please click the Next             '...
        'button to get started.                                         '...
        '                                                               '...
        '                                                               '...
        '                                                               '...
        '                                                               ';...
        % Slide 2
        'Assume that we want to transmit the following binary   '...
        'data using OFDM: [0 0 0 1 1 0 1 1].                            '...
        'The plot shows this binary data.                               '...
        '                                                               '...
        '                                                               '...
        '                                                               '...
        '                                                               ';...
        % Slide 3
        'In OFDM an IFFT (Inverse Fast Fourier Transform) is    '...
        'used to put the binary numbers onto many frequencies. '...
        'Due to the math involved in an IFFT, these                     '...
        'frequencies do not interfere with eath other (in               '...
        'communication terms, this is called "Orthogonality"). '...
        'The plot shows that each group of 2 blue data points   '...
        'under a red hump will be put onto one frequency.               ';...
        % Slide 4
        'The IFFT math is now complete. It has generated an             '...
        'OFDM signal that corresponds to the binary data.               '...
        'The plot shows the signal generated by the IFFT.               '...
        '                                                               '...
        '                                                               '...
        '                                                               '...
        '                                                               ';...
        % Slide 5 - same plot
        'Now, this OFDM signal can be transmitted through a             '...
        'media and then received. This media (or "Channel" in   '...
        'communication) could be wired or wireless. Once the    '...
        'signal is received, the reverse process is done to             '...
        'recover the original binary data.                              '...
        '                                                               '...
        '                                                               ';...
        % Slide 6
        'Finally, an FFT (Fast Fourier Transform) is used to    '...
        'recover the binary data as shown in the plot. Note             '...
        'that the FFT is the opposite of the IFFT used to               '...
        'generate the OFDM signal. As long as the Channel does '...
        'not distort the OFDM signal too much, the original             '...
        'binary data can be recovered.                                  '...
        '                                                               '];



switch(action)
        case 'next'                             %--------------------------------------
                textHnd1=findobj('Tag','StaticTextFeedback');
                nextHnd1=findobj('Tag','PushbuttonNext'); % handler for the Next button
                axisHnd1=findobj('Tag','Axes1');
                global COUNTER
                if isempty(COUNTER)
                        COUNTER = 0;            % initialize COUNTER if doesn't exist
                end
                COUNTER = COUNTER + 1;
                [r c]=size(stringArray);
                if COUNTER > r
                        COUNTER = 0;
                        close(gcf)
                        basicGUI   % set to file name in future!
                else
                        set(textHnd1,'String',stringArray(COUNTER,:))
                        switch(COUNTER)
                                case 1
                                        % disp('Slide 1')
                                case 2
                                        % disp('Slide 2')
```

```
                        setupGUI % sets up the GUI variables
                        set(axisHnd1,'Visible','on')
                        % Stem Plot the Binary Data
                        stem(data_in,'filled')
                case 3
                        % disp('Slide 3')
                        setupGUI % sets up the GUI variables % add
                        groupings around the stem plot
                        y=1.2*abs(sin(linspace(0,4*pi,80))).^(1/5);
                        x=linspace(0.5,8.5,80);
                        plot(x,y,'r'),hold on stem(data_in,'filled'),hold
                        off
                case 4
                        % disp('Slide      4')
                        setupGUI
                        % Perform the ifft and display the results
                        tx plot(xmit)

                case 5
                        % disp('Slide 5')
                        % same plot
                case 6
                        % disp('Slide 6')
                        setupGUI
                        tx, ch, rx
                        stem(output,'filled')

                        set(nextHnd1,'String','Start Over')              % repeat if desired
                otherwise
                        disp('error')
            end
        end


    case 'close'                %--------------------------------------
        clear global COUNTER
        close(gcbf)

    case 'figure'               %--------------------------------------
        % this is called whenever the figure is first created -or NOT???
                    textHnd1=findobj('Tag','StaticTextFeedback');
        axisHnd1=findobj('Tag','Axes1');
        set(textHnd1,'String','Basic OFDM Demo') % default text message
        set(axisHnd1,'Visible','off') % hide Axis to          begin
end
```

% OFDMguiFnSound.m

```
function OFDMguiFnSound(action)
% Consolidates all of the GUI callbacks into one main function
% Alan Brooks the man wrote this

stringArray = [...
    % Slide 1
    'Welcome to the Sound OFDM demo. This simulates QAM            '...
    'and OFDM using a sound file as input to demonstrate           '...
    'the advantages of using OFDM with a multipath                 '...
    'channel.                                                      '...
    'Choose the strength of multipath present in the               '...
    'channel and the plot will show the current channels           '...
    'frequency response.                                           ';...
    % Slide 2
    'Here is a frequency domain (FD) representation of the         '...
    'QAM data to be transmitted.                                   '...
    'Press any key to continue.                                    '...
    '                                                              '...
    '                                                              '...
    '                                                              '...
    '                                                              ';...
    % Slide 2b
    'For QAM (single-carrier) transmission, this plot              '...
    'shows the channel frequency response (black) and the          '...
    'received data (light blue) overlayed on the original          '...
    'data (blue). Note that the received data is slightly          '...
    'distorted due to the fading channel caused by                 '...
    'multipath.                                                    '...
    'Press any key to continue.                                    ';...
    % Slide 2c
    'Here is a frequency domain (FD) representation of the         '...
```

```
'OFDM data to be transmitted.                               '...
'Press any key to continue.                                 '...
'                                                           '...
'                                                           '...
'                                                           '...
'                                                           ';...
% Slide 2d
'For OFDM (multi-carrier) transmission, this plot           '...
'shows the channel (black) and received data (light         '...
'blue) overlayed on the original data (blue). Note          '...
'that the OFDM received data also exhibits multipath    '...
'distortion. Also, notice that the OFDM signal is           '...
'spread out over more bandwidth than QAM since OFDM         '...
'uses many carrier frequencies.                             ';...
% Slide 3
'Here are the final plots of the recovered sound files '...
'along with the Bit Error Rate (BER) for OFDM and QAM. '...
'Click any of the 3 buttons to hear these sounds.           '...
'Since OFDM handles multipath better, the sound is          '...
'less distorted.                                            '...
'The Long Sounds demonstrate longer examples that have '...
'already been processed offline.                            '];



switch(action)
	case 'next'			%--------------------------------------
		textHnd1=findobj('Tag','StaticTextFeedback');
		nextHnd1=findobj('Tag','PushbuttonNext'); % handler for the Next button
		% axis handlers
			axisHnd1=findobj('Tag','Axes1');			% main
			axisHnd2=findobj('Tag','AxesOriginal'); % original
			axisHnd3=findobj('Tag','AxesQAM');			% QAM
			axisHnd4=findobj('Tag','AxesOFDM');			% OFDM
		% multipath handlers
			textHnd2=findobj('Tag','StaticTextMultipath');
			popupHnd1=findobj('Tag','PopupMenuMultipath');
		% Generated Sounds handlers
			textHnd3=findobj('Tag','StaticTextGenSounds');
			OriginalHnd1=findobj('Tag','PushbuttonOriginal');
			QAMHnd1=findobj('Tag','PushbuttonQAM');
			OFDMHnd1=findobj('Tag','PushbuttonOFDM');
		% Long Sounds handlers
			textHnd4=findobj('Tag','StaticTextLongSounds');
			OriginalLongHnd1=findobj('Tag','PushbuttonOriginalLong');
			QAMLongHnd1=findobj('Tag','PushbuttonQAMLong');
			OFDMLongHnd1=findobj('Tag','PushbuttonOFDMLong');
		% BER handlers
			textHnd5=findobj('Tag','StaticTextBER1'); % label
			textHnd6=findobj('Tag','StaticTextBER2'); % label
			textHnd7=findobj('Tag','StaticTextBERQAM'); % OFDM BER field
			textHnd8=findobj('Tag','StaticTextBEROFDM'); % QAM BER field
		global COUNTER
		if isempty(COUNTER)
			COUNTER = 0;		% initialize COUNTER if doesn't exist
		end
		COUNTER = COUNTER + 1;
		[r c]=size(stringArray);
		if COUNTER > r
			COUNTER = 0;
			close(gcf)
			SoundGUI  % set to file name in future!
		else
			set(textHnd1,'String',stringArray(COUNTER,:))
			switch(COUNTER)
				case 1
					% disp('Slide 1')
					% Show/Hide the GUI
						set(nextHnd1,'String','Next')
						% show multipath controls
							set(textHnd2,'Visible','on')
							set(popupHnd1,'Visible','on')
						% enable multipath controls
							set(textHnd2,'Enable','on')
							set(popupHnd1,'Enable','on')
						% show main axis
							set(axisHnd1,'Visible','on'),axes(axisHnd1)
						% hide other axis's
							set(axisHnd2,'Visible','off')
```

```
                    set(axisHnd3,'Visible','off')
                    set(axisHnd4,'Visible','off')
          % hide generated sounds stuff set(textHnd3,'Visible','off')
                    set(OriginalHnd1,'Visible','off')
                    set(QAMHnd1,'Visible','off')
                    set(OFDMHnd1,'Visible','off')

          % hide long sounds stuff set(textHnd4,'Visible','off')
                    set(OriginalLongHnd1,'Visible','off')
                    set(QAMLongHnd1,'Visible','off')
                    set(OFDMLongHnd1,'Visible','off')

          % hide the BER displays set(textHnd5,'Visible','off')
                    set(textHnd6,'Visible','off')
                    set(textHnd7,'Visible','off')
                    set(textHnd8,'Visible','off')

set(popupHnd1,'Value',1) % no channel by default % default plot

plot(0:.05:.5,zeros(1,11)),axis([0 0.5 -12 6]),title('Channel Magnitude
Response')

xlabel('Digital Frequency'),ylabel('Magnitude (dB)')

case {2, 3, 4, 5}
          % disp('Slide 2')
          % disble multipath controls
                    set(textHnd2,'Enable','off')
                    set(popupHnd1,'Enable','off')
          setupSoundGUI % sets up the Sound GUI variables set(textHnd1,'String','QAM
          Simulation... Please Wait') QAM

          set(textHnd1,'String',stringArray(COUNTER,:)) fft_temp =
          abs(fft(QAM_tx_data));
          fft_temp = fft_temp(1:floor(0.5*length(fft_temp))); % truncate (+ spectrum) dig_x_axis = (1:length(fft_temp)) /
          (2*length(fft_temp));
          plot(dig_x_axis, fft_temp) title('FFT of
          Transmitted QAM')
          % calculate the BER and store for slide 6 global
                    BER_QAM_TEMP; binary_err_bits_QAM =
                    0;
                    for i = 1:length(data_in)
                              err = abs(data_in(i)-QAM_data_out(i)); if err > 0

                                        binary_err_bits_QAM = binary_err_bits_QAM + 1;
                              end
                    end
                    BER_QAM_TEMP = 100 * binary_err_bits_QAM/data_length; COUNTER =
          COUNTER + 1;
          pause

          % disp('Slide 2b') set(textHnd1,'String',stringArray(COUNTER,:))
          hold on

                    % QAM Plotting
                    fft_temp = abs(fft(QAM_rx_data));
                    fft_temp = fft_temp(1:floor(0.5*length(fft_temp))); % truncate plot(dig_x_axis,
                    fft_temp,'c'),title(' ')
                    % channel display if
                    channel_on == 1
                              ComputeChannelGUI
                              size_mag=max(mag)-min(mag); % for scaled channel plot
                              plot(W/(2*pi),(0.5*max(fft_temp)/size_mag)*(mag +
abs(min(mag))) + 0.5*max(fft_temp),'k')

                    end
          hold off
          COUNTER = COUNTER + 1;
          pause

          % disp('Slide 2c')
          set(textHnd1,'String','OFDM Simulation... Please Wait') OFDM

          set(textHnd1,'String',stringArray(COUNTER,:)) fft_temp =
          abs(fft(xmit));
          fft_temp = fft_temp(1:floor(0.5*length(fft_temp))); % truncate dig_x_axis =
          (1:length(fft_temp)) / (2*length(fft_temp)); plot(dig_x_axis, fft_temp)

          title('FFT of Transmitted OFDM')
          % calculate the BER and store for slide 6 global
                    BER_OFDM_TEMP; binary_err_bits_OFDM
                    = 0;
```

```
                    for  i  =  1:length(data_in)
                              err = abs(data_in(i)-output(i)); if err > 0

                                        binary_err_bits_OFDM  =  binary_err_bits_OFDM  +1;
                              end
                    end
                    BER_OFDM_TEMP = 100 * binary_err_bits_OFDM/data_length; COUNTER =
          COUNTER + 1;
          pause

          % disp('Slide 2d') set(textHnd1,'String',stringArray(COUNTER,:))
          hold on

                    %  OFDM  Plotting
                    fft_temp  =  abs(fft(recv));
                    fft_temp = fft_temp(1:floor(0.5*length(fft_temp))); % truncate plot(dig_x_axis,
                    fft_temp,'c'),title(' ')
                    % channel display if
                    channel_on == 1
                              plot(W/(2*pi),(0.5*max(fft_temp)/size_mag)*(mag  +
abs(min(mag)))  +  0.5*max(fft_temp),'k')

                    end
          hold  off

     case  6
          % disp('Slide              3')
          setupSoundGUI
          % hide  main  axis
                    plot(0) % clear the plot axis off

                    % set(axisHnd1,'Visible','off')
          % show  other  axis's
                    set(axisHnd2,'Visible','on')
                    set(axisHnd3,'Visible','on')
                    set(axisHnd4,'Visible','on')
          % hide multipath controls set(textHnd2,'Visible','off')
                    set(popupHnd1,'Visible','off')

          % show generated sound buttons set(textHnd3,'Visible','on')
                    set(OriginalHnd1,'Visible','on')
                    set(QAMHnd1,'Visible','on')
                    set(OFDMHnd1,'Visible','on')

          % show long sounds stuff set(textHnd4,'Visible','on')
                    set(OriginalLongHnd1,'Visible','on')
                    set(QAMLongHnd1,'Visible','on')
                    set(OFDMLongHnd1,'Visible','on')

          % show the BER displays set(textHnd5,'Visible','on')
                    set(textHnd6,'Visible','on')
                    set(textHnd7,'Visible','on') % QAM
                    set(textHnd8,'Visible','on') % OFDM

          % Display the BERs global
          BER_QAM_TEMP; global
          BER_OFDM_TEMP;
          set(textHnd7,'String',strcat(num2str(BER_QAM_TEMP,3),' %'))
          set(textHnd8,'String',strcat(num2str(BER_OFDM_TEMP,3),' %')) clear global
          BER_QAM_TEMP; % clean up the globals
          clear  global  BER_OFDM_TEMP;
          % Plot  the  Sounds
          %    Note: axes(handle) sets to plot on the handle axis axes(axisHnd2)
          plot(wavread(file_name)),title('Original sound') axes(axisHnd3)
          plot(wavread('QAM_out.wav')),title('QAM sound') axes(axisHnd4)
          plot(wavread('OFDM_out.wav')),title('OFDM sound')


          set(nextHnd1,'String','Start  Over')                    %  repeat  if  desired

     otherwise
          disp('error')
          COUNTER = 0;
  end
end


case  'mp_channel'     %-----------------------------------
     ComputeChannelGUI if
     noChannel ~= 1
```

```
                        % large or small case
                        plot(W/(2*pi),mag),axis([0 0.5 -12 6]),title('Channel Magnitude Response')
                        xlabel('Digital Frequency'),ylabel('Magnitude (dB)')
                else
                        % none case
                        plot(0:.05:.5,zeros(1,11)),axis([0 0.5 -12 6]),title('Channel Magnitude Response')
                        xlabel('Digital Frequency'),ylabel('Magnitude (dB)')
                end

        case 'close'            %-----------------------------------
                clear global COUNTER
                close(gcbf)

        case 'PlayOriginal'     %-----------------------------------
                sound(wavread('shortest.wav'),11025)

        case 'PlayQAM'          %-----------------------------------
                sound(wavread('QAM_out.wav'),11025)

        case 'PlayOFDM'         %-----------------------------------
                sound(wavread('OFDM_out.wav'),11025)

        case 'PlayOriginalLong'                 %-----------------------------------
                if strcmp('Student Edition',hostid)
                        sound(wavread('Long.wav',16384),11025)          % check for student array size limit
                else
                        sound(wavread('Long.wav'),11025)
                end

        case 'PlayQAMLong'      %-----------------------------------
                if strcmp('Student Edition',hostid)
                        sound(wavread('QAM_Long.wav',16384),11025)              % check for student array size limit
                else
                        sound(wavread('QAM_Long.wav'),11025)
                end

        case 'PlayOFDMLong'     %-----------------------------------
                if strcmp('Student Edition',hostid)
                        sound(wavread('OFDM_Long.wav',16384),11025)             % check for student array size limit
                else
                        sound(wavread('OFDM_Long.wav'),11025)
                end

        case 'figure'           %-----------------------------------
                % this is called whenever the figure is first created -or NOT???
%               textHnd1=findobj('Tag','StaticTextFeedback');
%               axisHnd1=findobj('Tag','Axes1');
%               set(textHnd1,'String','Sound OFDM Demo') % default text message
%               set(axisHnd1,'Visible','off') % hide Axis to begin
end
```

% pol2bin.m

```
function y = pol2bin(x)

% pol2bin
%
% Converts polar numbers (-1,1) to binary numbers (0,1)
% Accepts a 1-D array of polar numbers
% Removes trailing zeros, since they are not valid data

% % Remove zeros - not needed with intelligent decoding
% last_data=length(x);
% while x(last_data) == 0
%       last_data = last_data - 1;
% end

y = ones(1,length(x)); for i =
1:length(x)
        if x(i) == -1
                y(i) = 0;
        end
end
```

% QAM.m

```
% QAM.m compares OFDM (multicarrier) to multi-level QAM (single carrier)
% when they transmit the same # of bits in a given time period

read                    % read data for QAM - does not affect OFDM
data_in_pol = bin2pol(data_in);                     % Converts binary data to polar data

% check to see if num_carriers is a power of 2
is_pow_2 = num_carriers;
temp_do_QAM = 0;
if is_pow_2 ~= 2
        while temp_do_QAM == 0
                temp_do_QAM = rem(is_pow_2,2);
                is_pow_2 = is_pow_2/2;
                if is_pow_2 == 2
                        temp_do_QAM = -99;          % it is a power of 2 -> can do QAM
                end
        end
else
        temp_do_QAM = -99;   % 2 is a power of 2
end
if temp_do_QAM ~= -99
        do_QAM = 0;             % don't do it if it's not possible
        disp(' '),disp('ERROR: Cannot run QAM because num_carriers is not valid.') disp(' Please see "setup.m" for
        details.')
end


if do_QAM == 1
        tic             % Start stopwatch to calculate how long QAM simulation takes

        disp(' '), disp('------------------------------------------------------------')
        disp('QAM simulation'), disp('Transmitting')

        % Pad with zeros so data can be divided evenly
        data_length = length(data_in_pol);
        r = rem(data_length,num_carriers);
        if r ~= 0
                for i = 1:num_carriers-r
                        data_in_pol(data_length+i) = 0;         %pad input with zeros to complete last data set
                end                                                     %speed improve possible
        end
        data_length = length(data_in_pol);              %update after padding

        num_OFDM_symbols = ceil(data_length / (2*num_carriers));
        % num QAM symbols that represent equal amount of data to one OFDM symbol num_QAM_symbols =
        num_carriers / 2;
        % num samples per QAM symbol
        num_symbol_samples = fft_size / num_QAM_symbols;

        % convert polar data [-1, 1] to 4 level data [-3, -1, 1, 3] data_in_4 =
        zeros(1,data_length/2);
        for i = 1:2:data_length
                data_in_4(i - (i-1)/2) = data_in_pol(i)*2 + data_in_pol(i+1);
        end

        % define sample points between 0 and 2*pi
        ts = linspace(0, 2*pi*QAM_periods, num_symbol_samples+1);

        % Generate 16-QAM data
        % total length of 16-QAM transmission
        tx_length = num_OFDM_symbols * num_QAM_symbols * num_symbol_samples; QAM_tx_data =
        zeros(1,tx_length);
        for i = 1:2:data_length/2
                for k = 1:num_symbol_samples
                        QAM_tx_data(k+((i-1)/2)*num_symbol_samples) = data_in_4(i)*cos(ts(k)) + data_in_4(i+1)*sin(ts(k));
                end
        end

        % Do channel simulation on QAM data
        xmit = QAM_tx_data;             % ch uses 'xmit' data and returns 'recv'
        ch
        QAM_rx_data = recv;             % save QAM data after channel
        clear recv                              % remove 'recv' so it won't interfere with OFDM
        clear xmit                              % remove 'xmit' so it won't interfere with OFDM

        disp('Receiving')                       % Recover Binary data (Decode QAM)
        cos_temp = zeros(1,num_symbol_samples);         %
        sin_temp = cos_temp;                                    %
```

```
		xxx = zeros(1,data_length/4);				% Initialize to zeros for speed
		yyy = xxx;								%
		QAM_data_out_4 = zeros(1,data_length/2);		%

		for i = 1:2:data_length/2			% "cheating"
			for k = 1:num_symbol_samples
				% multiply by carriers to produce high frequency term and original data cos_temp(k) =
				QAM_rx_data(k+((i-1)/2)*num_symbol_samples) * cos(ts(k)); sin_temp(k) = QAM_rx_data(k+((i-
				1)/2)*num_symbol_samples) * sin(ts(k));
			end
			% LPF and decide - we will do very simple LPF by averaging xxx(1+(i-1)/2) =
			mean(cos_temp);
			yyy(1+(i-1)/2) = mean(sin_temp);
			% Reconstruct data in serial form
			QAM_data_out_4(i) = xxx(1+(i-1)/2);
			QAM_data_out_4(i+1) = yyy(1+(i-1)/2);
		end

		% Make decision between [-3, -1, 1, 3] for i =
		1:data_length/2
			if QAM_data_out_4(i) >= 1, QAM_data_out_4(i) = 3; elseif QAM_data_out_4(i)
			>= 0, QAM_data_out_4(i) = 1; elseif QAM_data_out_4(i) >= -1,
			QAM_data_out_4(i) = -1; else QAM_data_out_4(i) = -3;

			end
		end

		% Convert 4 level data [-3, -1, 1, 3] back to polar data [-1, 1] QAM_data_out_pol =
		zeros(1,data_length); % "cheating" for i = 1:2:data_length

			switch QAM_data_out_4(1 + (i-1)/2) case -3

					QAM_data_out_pol(i) = -1;
					QAM_data_out_pol(i+1) = -1;
				case -1
					QAM_data_out_pol(i) = -1;
					QAM_data_out_pol(i+1) = 1;
				case 1
					QAM_data_out_pol(i) = 1;
					QAM_data_out_pol(i+1) = -1;
				case 3
					QAM_data_out_pol(i) = 1;
					QAM_data_out_pol(i+1) = 1;
				otherwise
					disp('Error detected in switch statment - This should not be happening.');
			end
		end
		QAM_data_out = pol2bin(QAM_data_out_pol);			% convert back to binary

		% Stop stopwatch to calculate how long QAM simulation takes QAM_simulation_time
		= toc;
		if QAM_simulation_time > 60
			disp(strcat('Time for QAM simulation=', num2str(QAM_simulation_time/60), ' minutes.'));
		else
			disp(strcat('Time for QAM simulation=', num2str(QAM_simulation_time), ' seconds.'));
		end
end
```

% **read.m**

```
% read

% ******************FILE INPUT SETUP*********************************
if input_type == 2

	if file_input_type == 1 %binary file input

	end

	if file_input_type == 2 %text file input

		file = fopen(file_name,'rt'); data_samples =
		fread(file,'char'); fclose(file);

		data_in = zeros(1,8*length(data_samples)); for i =
		1:length(data_samples)
			data_in(1 + (i-1)*8:(i-1)*8 + 8) = eight2bin(data_samples(i));
		end
	end
```

```
        if file_input_type == 3 %sound file
                input
                data_samples=wavread(file_name);
                %needs to be normalized from -1:1 to 0:255 for 8 bit conversion data_samples_resized =
                round(128*data_samples +127);
                data_in = zeros(1,8*length(data_samples_resized)); for i =
                1:length(data_samples_resized)
                        data_in(1 + (i-1)*8:(i-1)*8 + 8) = eight2bin(data_samples_resized(i));
                end
        end

        if file_input_type == 4 %image file
                input
                [data_in,map]=imread(file_name);                % read image and corresponding color map for display

        end

end
```

% rx.m

```
% rx disp('Receiving')


rx_chunk

% perform fft to recover original data from time domain sets recv_spaced_chunks =
zeros(num_chunks,fft_size);
for i = 1:num_chunks
        recv_spaced_chunks(i,1:fft_size) = fft(recv_td_sets(i,1:fft_size));
        % Note: 'round()' gets rid of small numerical error in Matlab but a threshold will be needed for a practical system
        % 2001-4-17 -- Got rid of 'round()' to do decoding more intelligently
end

rx_dechunk

output = pol2bin(output);               % Converts polar to binary

write
```

% rx_chunk.m

```
% rx_chunk

% break received signal into parellel sets for demodulation
recv_td_sets = zeros(num_chunks,fft_size); for i =
1:num_chunks
        for k = 1:fft_size
                recv_td_sets(i,k) = recv(k + (i-1)*fft_size);
        end
end
```

% rx_dechunk.m

```
% rx_dechunk

% take out zeros_between from recv_spaced_chunks --> recv_padded_chunks
recv_padded_chunks = zeros(num_chunks, num_carriers+num_zeros); i = 1;

for k = zeros_between +1:zeros_between +1:fft_size/2 recv_padded_chunks(1:num_chunks,i) =
        recv_spaced_chunks(1:num_chunks,k); i = i+1;

end

% take out num_zeros from padded chunks --> recv_chunks recv_chunks =
zeros(num_chunks, num_carriers);
recv_chunks = recv_padded_chunks(1:num_chunks, num_zeros+1:num_carriers+num_zeros);

% Recover bit stream by placing reconstructed frequency domain data in series recv_dechunked = zeros(1,
num_chunks*num_carriers);
for i = 1:num_chunks
        for k = 1:num_carriers
                recv_dechunked(k + (i-1)*num_carriers*2) = real(recv_chunks(i,k)); recv_dechunked(k + (i-1)*num_carriers*2 +
                num_carriers) = imag(recv_chunks(i,k));
        end
end
```

```
% take out trailing zeros from output --                > output
output_analog = recv_dechunked(1:data_length);
output = sign(output_analog);
```

% setup.m

```
% setup
disp(' '), disp('----------------------------------------------------------------------------------------------------')
disp('Simulation Setup')

% OFDM Setup -----------------------------------------------------------
fft_size = 128                  % should be a power of 2 for fast computation
                                % more points = more time domain samples (smoother & more cycles)
num_carriers = 32          % should be <= fft_size/4
                                % number of carriers used for each data chunk
% new var - denotes even spacing or variations of carriers among fft points input_type = 2;

% 1 = test input
        test_input_type = 1;
        % 1 = bit specified (binary)
                binary_data = [0 1 0 1 0 1 0 1];
        % 2 = random data stream (samples in the range of 0-255)
                num_symbols = 9;
        % 3 = sinusoidal
                frequency = 2;
                num_samples = 50;
% 2 = external file input
        file_name = 'shortest.wav';            % Name of input file
        file_input_type = 3;
                % 1 = binary (not implemented)
                % 2 = text                                        % Demo file:  'text.txt'
                % 3 = sound                                  % Demo files: 'shortest.wav' & 'shorter.wav'
                % 4 = image (not implemented)


% QAM Setup ------------------------------------------------------------
do_QAM = 1;                     % (1=on, 0=off)
QAM_periods = 10;    % defines the number of periods per QAM Symbos (1=2*pi)


% Channel Simulation Parameters --------------------------------------------
channel_on = 1;                 % 1=on, 0=off
clip_level = 1.0;               % 0.0 - 1.0 (0-100%)
        % Max magnitude of the signal is 'clip_level' times the full magnitude of the signal
noise_level = 0.0;   % 0.0 - 1.0 (0-100%)
        % Magnitude of noise is 'noise_level' times the magnitude of the signal
% Multipath Channel Simulation
        % Good defaults when fft_size = 128 and num_carriers = 32:
        %       d1=6; a1=0.30; d2=10; a2=0.25
        d1  = 6;            % delay   in units
        a1  = 0.32;         % attenuation factor - multipath signal is x% of size or original signal
        d2  = 10;           % delay   for second multipath signal
        a2  = 0.28;         % attenuation factor for second multipath signal


% ****************** TEST INPUT SETUP - DO NOT MODIFY **************************

if input_type == 1
        if test_input_type == 1
                %specify BINARY input bit-by-bit data_in =
                binary_data;
        end
        if test_input_type == 2
                %random input defined by parameters
                num_levels = 255; %number of possible levels of a symbol %must be integer between 1-255

                data_samples = round(rand(1,num_symbols)*(num_levels-1)); data_in =
                zeros(1,8*length(data_samples));
                for i = 1:length(data_samples)
                        data_in(1 + (i-1)*8:(i-1)*8 + 8) = eight2bin(data_samples(i));
                end
        end
        if test_input_type == 3
                %data stream represents sine wave samples
                t = linspace(0,1,num_symbols); %evenly space number of samples %take 8-bit samples of sine wave

                data_samples = round(127.5*sin(frequency*2*pi*t) +127.5); data_in =
                zeros(1,8*length(data_samples));
```

```
                    for i = 1:length(data_samples)
                              data_in(1 + (i-1)*8:(i-1)*8 + 8) = eight2bin(data_samples(i));
                    end
          end
end

already_made_noise = 0;            % initialization (don't change)
```

% SetupGUI.m

```
% SetupGUI.m sets up the basicGUI variables

% Initialize the appropriate setup.m variables
fft_size = 64;
num_carriers = 4;
input_type = 1; test_input_type = 1;
channel_on = 0;
do_QAM = 0;
data_samples = [0 0 0 1 1 0 1 1];             % data to be transmitted
data_in = data_samples;
```

% SetupSoundGUI.m

```
% SetupSoundGUI.m sets up the SoundGUI variables

% Initialize the appropriate setup.m variables
fft_size = 128;
num_carriers = 32;
input_type = 2; file_input_type = 3; file_name = 'shortest.wav';
channel_on = 1;
do_QAM = 1;
QAM_periods = 10;
clip_level = 1.0;                    % 0.0 - 1.0 (0-100%)
noise_level = 0.0;
already_made_noise = 0;
ComputeChannelGUI
```

% SoundGUI.m

```
function SoundGUI()
% This is the machine-generated representation of a MATLAB object
% and its children.  Note that handle values may change when these
% objects are re-created. This may cause problems with some callbacks.
% The command syntax may be supported in the future, but is currently
% incomplete and subject to change.
%
% To re-open this system, just type the name of the m-file at the MATLAB
% prompt. The M-file and its associtated MAT-file must be on your path.

load SoundGUI

a = figure('Color',[0.9 0.9 0.9], ...
          'Colormap',mat0, ...
          'CreateFcn','OFDMguiFn figure', ...
          'Position',[376 239 624 480], ...
          'Resize','off', ...
          'Tag','Fig1');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[1 1 1], ...
          'FontName','Monaco', ...
          'HorizontalAlignment','left', ...
          'Position',[59 2 340 94], ...
          'String','Sound OFDM Demo', ...
          'Style','text', ...
          'Tag','StaticTextFeedback');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0.3 0.3 0.3], ...
          'Position',[472 -1 152 481], ...
          'Style','frame', ...
          'Tag','Frame1');
b = uicontrol('Parent',a, ...
          'Units','points', ...
          'BackgroundColor',[0.733333 0.733333 0.733333], ...
          'Callback','OFDMguiFnSound next', ...
          'FontSize',14, ...
```

```
            'Position',[493 435 102 32], ...
            'String','Begin', ...
            'Tag','PushbuttonNext');
b = uicontrol('Parent',a, ...
            'Units','points', ...
            'BackgroundColor',[0.733333 0.733333 0.733333], ...
            'Callback','OFDMguiFnSound close', ...
            'FontSize',14, ...
            'Position',[493 10 102 32], ...
            'String','Close', ...
            'Tag','PushbuttonClose');
b = uicontrol('Parent',a, ...
            'Units','points', ...
            'BackgroundColor',[0.733333 0.733333 0.733333], ...
            'Callback','OFDMguiFnSound mp_channel', ...
            'Enable','off', ...
            'Position',[489 209 87 30], ...
            'String',mat1, ...
            'Style','popupmenu', ...
            'Tag','PopupMenuMultipath', ...
            'Value',2, ...
            'Visible','off');
b = uicontrol('Parent',a, ...
            'Units','points', ...
            'BackgroundColor',[0.733333 0.733333 0.733333], ...
            'Enable','off', ...
            'FontWeight','bold', ...
            'Position',[489 251 129 17], ...
            'String','Multipath Channel', ...
            'Style','text', ...
            'Tag','StaticTextMultipath', ...
            'Visible','off');
b = uicontrol('Parent',a, ...
            'Units','points', ...
            'BackgroundColor',[0.733333 0.733333 0.733333], ...
            'FontWeight','bold', ...
            'Position',[489 398 129 18], ...
            'String','Generated Sounds', ...
            'Style','text', ...
            'Tag','StaticTextGenSounds', ...
            'Visible','off');
b = uicontrol('Parent',a, ...
            'Units','points', ...
            'BackgroundColor',[0.733333 0.733333 0.733333], ...
            'Callback','OFDMguiFnSound PlayOriginal', ...
            'FontSize',14, ...
            'Position',[489 364 107 28], ...
            'String','Original', ...
            'Tag','PushbuttonOriginal', ...
            'Visible','off');
b = uicontrol('Parent',a, ...
            'Units','points', ...
            'BackgroundColor',[0.733333 0.733333 0.733333], ...
            'Callback','OFDMguiFnSound PlayQAM', ...
            'FontSize',14, ...
            'Position',[489 329 107 28], ...
            'String','QAM', ...
            'Tag','PushbuttonQAM', ...
            'Visible','off');
b = uicontrol('Parent',a, ...
            'Units','points', ...
            'BackgroundColor',[0.733333 0.733333 0.733333], ...
            'Callback','OFDMguiFnSound PlayOFDM', ...
            'FontSize',14, ...
            'Position',[489 293 107 28], ...
            'String','OFDM', ...
            'Tag','PushbuttonOFDM', ...
            'Visible','off');
b = axes('Parent',a, ...
            'Units','points', ...
            'Box','on', ...
            'CameraUpVector',[0 1 0], ...
            'CameraUpVectorMode','manual', ...
            'Color',[1 1 1], ...
            'ColorOrder',mat2, ...
            'Position',[51 363 361 84], ...
            'Tag','AxesOriginal', ...
            'XColor',[0 0 0], ...
            'YColor',[0 0 0], ...
            'ZColor',[0 0 0]);
```

```
c = line('Parent',b, ...
        'Color',[0 0 1], ...
        'Tag','Line1', ...
        'XData',1, ...
        'YData',0);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[0.997222 -1.37349 0], ...
        'Tag','Text13', ...
        'VerticalAlignment','cap');
set(get(c,'Parent'),'XLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[-0.141667 2.22045e-16 0], ...
        'Rotation',90, ...
        'Tag','Text14', ...
        'VerticalAlignment','baseline');
set(get(c,'Parent'),'YLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','right', ...
        'Position',[-0.286111 1.80723 0], ...
        'Tag','Text15', ...
        'Visible','off');
set(get(c,'Parent'),'ZLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[0.997222 1.13253 0], ...
        'Tag','Text16', ...
        'VerticalAlignment','bottom');
set(get(c,'Parent'),'Title',c);
b = axes('Parent',a, ...
        'Units','points', ...
        'Box','on', ...
        'CameraUpVector',[0 1 0], ...
        'CameraUpVectorMode','manual', ...
        'Color',[1 1 1], ...
        'ColorOrder',mat3, ...
        'Position',[51 249 363 82], ...
        'Tag','AxesQAM', ...
        'XColor',[0 0 0], ...
        'YColor',[0 0 0], ...
        'ZColor',[0 0 0]);
c = line('Parent',b, ...
        'Color',[0 0 1], ...
        'Tag','Line2', ...
        'XData',1, ...
        'YData',0);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[0.997238 -1.38272 0], ...
        'Tag','Text9', ...
        'VerticalAlignment','cap');
set(get(c,'Parent'),'XLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[-0.140884 0 0], ...
        'Rotation',90, ...
        'Tag','Text10', ...
        'VerticalAlignment','baseline');
set(get(c,'Parent'),'YLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','right', ...
        'Position',[-0.28453 4.69136 0], ...
        'Tag','Text11', ...
        'Visible','off');
set(get(c,'Parent'),'ZLabel',c);
```

```
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[0.997238 1.1358 0], ...
        'Tag','Text12', ...
        'VerticalAlignment','bottom');
set(get(c,'Parent'),'Title',c);
b = axes('Parent',a, ...
        'Units','points', ...
        'Box','on', ...
        'CameraUpVector',[0 1 0], ...
        'CameraUpVectorMode','manual', ...
        'Color',[1 1 1], ...
        'ColorOrder',mat4, ...
        'Position',[51 138 360 78], ...
        'Tag','AxesOFDM', ...
        'XColor',[0 0 0], ...
        'YColor',[0 0 0], ...
        'ZColor',[0 0 0]);
c = line('Parent',b, ...
        'Color',[0 0 1], ...
        'Tag','Line3', ...
        'XData',1, ...
        'YData',0);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[1 -1.4026 0], ...
        'Tag','Text5', ...
        'VerticalAlignment','cap');
set(get(c,'Parent'),'XLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[-0.091922 0 0], ...
        'Rotation',90, ...
        'Tag','Text6', ...
        'VerticalAlignment','baseline');
set(get(c,'Parent'),'YLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','right', ...
        'Position',[-0.286908 7.87013 0], ...
        'Tag','Text7', ...
        'Visible','off');
set(get(c,'Parent'),'ZLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[1 1.14286 0], ...
        'Tag','Text8', ...
        'VerticalAlignment','bottom');
set(get(c,'Parent'),'Title',c);
b = axes('Parent',a, ...
        'Units','points', ...
        'Box','on', ...
        'CameraUpVector',[0 1 0], ...
        'CameraUpVectorMode','manual', ...
        'Color',[1 1 1], ...
        'ColorOrder',mat5, ...
        'Position',[30 120 396 335], ...
        'Tag','Axes1', ...
        'XColor',[0 0 0], ...
        'YColor',[0 0 0], ...
        'ZColor',[0 0 0]);
c = line('Parent',b, ...
        'Color',[0 0 1], ...
        'Tag','Line4', ...
        'XData',1, ...
        'YData',0);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[1 -1.09281 0], ...
```

```
        'Tag','Text1', ...
        'VerticalAlignment','cap');
set(get(c,'Parent'),'XLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[-0.129114 0.00299401 0], ...
        'Rotation',90, ...
        'Tag','Text2', ...
        'VerticalAlignment','baseline');
set(get(c,'Parent'),'YLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','right', ...
        'Position',[-0.15443 1.15269 0], ...
        'Tag','Text3');
set(get(c,'Parent'),'ZLabel',c);
c = text('Parent',b, ...
        'Color',[0 0 0], ...
        'HandleVisibility','callback', ...
        'HorizontalAlignment','center', ...
        'Position',[1 1.03293 0], ...
        'Tag','Text4', ...
        'VerticalAlignment','bottom');
set(get(c,'Parent'),'Title',c);
b = uicontrol('Parent',a, ...
        'Units','points', ...
        'BackgroundColor',[0.733333 0.733333 0.733333], ...
        'FontWeight','bold', ...
        'Position',[489 162 129 18], ...
        'String','Longer Sounds', ...
        'Style','text', ...
        'Tag','StaticTextLongSounds', ...
        'Visible','off');
b = uicontrol('Parent',a, ...
        'Units','points', ...
        'BackgroundColor',[0.733333 0.733333 0.733333], ...
        'Callback','OFDMguiFnSound PlayQAMLong', ...
        'FontSize',14, ...
        'Position',[491 92 107 28], ...
        'String','QAM', ...
        'Tag','PushbuttonQAMLong', ...
        'Visible','off');
b = uicontrol('Parent',a, ...
        'Units','points', ...
        'BackgroundColor',[0.733333 0.733333 0.733333], ...
        'Callback','OFDMguiFnSound PlayOFDMLong', ...
        'FontSize',14, ...
        'Position',[491 58 107 28], ...
        'String','OFDM', ...
        'Tag','PushbuttonOFDMLong', ...
        'Visible','off');
b = uicontrol('Parent',a, ...
        'Units','points', ...
        'BackgroundColor',[0.733333 0.733333 0.733333], ...
        'Callback','OFDMguiFnSound PlayOriginalLong', ...
        'FontSize',14, ...
        'Position',[491 126 107 28], ...
        'String','Original', ...
        'Tag','PushbuttonOriginalLong', ...
        'Visible','off');
b = uicontrol('Parent',a, ...
        'Units','points', ...
        'BackgroundColor',[0.9 0.9 0.9], ...
        'Position',[414 284 36 15], ...
        'String','BER=', ...
        'Style','text', ...
        'Tag','StaticTextBER2', ...
        'Visible','off');
b = uicontrol('Parent',a, ...
        'Units','points', ...
        'BackgroundColor',[0.9 0.9 0.9], ...
        'Position',[414 176 36 15], ...
        'String','BER=', ...
        'Style','text', ...
        'Tag','StaticTextBER1', ...
        'Visible','off');
b = uicontrol('Parent',a, ...
```

```
        'Units','points', ...
        'BackgroundColor',[1 1 1], ...
        'HorizontalAlignment','left', ...
        'Position',[418 263 48 20], ...
        'Style','text', ...
        'Tag','StaticTextBERQAM', ...
        'Visible','off');
b = uicontrol('Parent',a, ...
        'Units','points', ...
        'BackgroundColor',[1 1 1], ...
        'HorizontalAlignment','left', ...
        'Position',[419 157 49 17], ...
        'Style','text', ...
        'Tag','StaticTextBEROFDM', ...
        'Visible','off');
```

% tx.m

```
% tx

disp('Transmitting')

read

data_in_pol = bin2pol(data_in);                 % Converts binary data to polar data

tx_chunk

% perform ifft to create time domain waveform representing data td_sets =
zeros(num_chunks,fft_size);
for i = 1:num_chunks
        td_sets(i,1:fft_size) = real(ifft(spaced_chunks(i,1:fft_size)));
end

tx_dechunk
```

% tx_chunk.m

```
% tx_chunk

data_length = length(data_in_pol);                        %number of symbols in original input
num_chunks = ceil(data_length/(2*num_carriers));          %2 data on each carrier (real and imaginary)
r = rem(data_length,2*num_carriers);

if r ~= 0
        for i = 1:num_carriers*2-r
                data_in_pol(data_length+i) = 0;           %pad input with zeros to complete last data set
        end                                               %speed improve possible
end

% break data into chunks
chunks = zeros(num_chunks,num_carriers);                  % for speed
for i = 1:num_chunks
        % *********************chunk done for k =
        1:num_carriers
                chunks(i,k) = data_in_pol(2*num_carriers*(i-1)+k) + data_in_pol(2*num_carriers*(i-1)+k+num_carriers)*j;
        end
end

% Padding chunks with zeros so num_carriers and fft_size are compatible
% Once compatible, further spacing is simplified
num_desired_carriers = num_carriers; num_zeros =
0;
thinking = 1;
while thinking == 1 % Continue if num_carriers and fft_size are not compatible if
        rem(fft_size/2,num_desired_carriers) == 0
                thinking = 0;
        else
                num_desired_carriers = num_desired_carriers + 1; num_zeros =
                num_zeros + 1;
        end
end

padded_chunks = zeros(num_chunks,num_carriers + num_zeros); % for speed
padded_chunks(1:num_chunks,num_zeros + 1:num_carriers + num_zeros) = chunks;


%compute zeros_between
```

```
zeros_between = ((fft_size/2) - (num_carriers + num_zeros))/(num_carriers + num_zeros);

spaced_chunks = zeros(num_chunks,fft_size); % for speed - extra room for folding later %add zeros_between

i = 1;
for k = zeros_between +1:zeros_between +1:fft_size/2 spaced_chunks(1:num_chunks,k) =
        padded_chunks(1:num_chunks,i); i = i+1;

end

% folding data to produce an odd function for ifft input for i = 1:num_chunks

        % Note: index = 1 is actually DC freq for ifft -> it does not get copied over y-axis spaced_chunks(i,fft_size:-1:fft_size/2+2) =
        conj(spaced_chunks(i,2:fft_size/2));
end
```

% tx_dechunk.m

```
% tx_dechunk

% Construct signal to transmit by placing time domain sets in series
xmit = zeros(1,num_chunks*fft_size); for i =
1:num_chunks
        for k = 1:fft_size
                xmit(k + (i-1)*fft_size) = td_sets(i,k);
        end
end
```

% write.m

```
% write

% ******************TEST OUTPUT*********************************
if input_type == 1

        if test_input_type == 1
                %already binary - do nothing
        end

        if (test_input_type == 2) | (test_input_type == 3)
                %random input        OR        sine wave samples
                output_samples = zeros(1,floor(length(output)/8));   %extra zeros are not original data
                for i = 1:length(output_samples)
                        output_samples(i) = bin2eight(output(1 + (i-1)*8:(i-1)*8 + 8));
                end
                if do_QAM == 1
                        QAM_output_samples = zeros(1,floor(length(QAM_data_out)/8));
                        for i = 1:length(QAM_output_samples)
                                QAM_output_samples(i) = bin2eight(QAM_data_out(1 + (i-1)*8:(i-1)*8 + 8));
                        end
                end
        end
end

% ******************FILE OUTPUT*********************************
if input_type == 2

        if file_input_type == 1
                %binary file output - not implemented
        end

        if file_input_type == 2
                %text file output
                output_samples = zeros(1,floor(length(output)/8));   %extra zeros are not original data
                for i = 1:length(output_samples)
                        output_samples(i) = bin2eight(output(1 + (i-1)*8:(i-1)*8 + 8));
                end
                file = fopen('OFDM_text_out.txt','wt+');
                fwrite(file,output_samples,'char');
                fclose(file);
                if do_QAM == 1
                        QAM_output_samples = zeros(1,floor(length(QAM_data_out)/8));   %extra zeros are not original data
                        for i = 1:length(QAM_output_samples)
                                QAM_output_samples(i) = bin2eight(QAM_data_out(1 + (i-1)*8:(i-1)*8 + 8));
                        end
                        file = fopen('QAM_text_out.txt','wt+');
                        fwrite(file,QAM_output_samples,'char');
                        fclose(file);
```

```
            end
    end

    if  file_input_type  ==  3
            output_samples_big = zeros(1,floor(length(output)/8)); %extra zeros are not original data for i = 1:length(output_samples_big)

                    output_samples_big(i)  =  bin2eight(output(1  +  (i-1)*8:(i-1)*8  +  8));
            end
            %convert dynamic range from 0:255 to -1:1 output_samples =
            (output_samples_big-127)/128; %sound file output

            wavwrite(output_samples, 11025, 8, 'OFDM_out.wav') if do_QAM == 1

                    QAM_data_out_big = zeros(1,floor(length(QAM_data_out)/8)); for i =
                    1:length(QAM_data_out_big)
                            QAM_data_out_big(i)  =  bin2eight(QAM_data_out(1  +  (i-1)*8:(i-1)*8  +  8));
                    end
                    %convert dynamic range from 0:255 to -1:1 QAM_output_samples =
                    (QAM_data_out_big-127)/128; %sound file output

                    wavwrite(QAM_output_samples,  11025,  8,  'QAM_out.wav')
            end
    end

    if  file_input_type  ==  4
            %image  file  output  -  not  implemented
    end

end
```