EE 361 Lab 1 Report

Multiuser Communications and Jamming

Authors: Donovan Gratton, Brian Willis

Lab Conducted 09-April-2018, 16-April-2018, and 23-April-2018

Authorship: This lab was conducted by Donovan Gratton and Brian Willis.
Primary author of Abstract, Introduction, and Discussion: Donovan Gratton
Primary author of Methods, Results, and Conclusion: Brian Willis

Abstract

This lab exercise was an inquiry based design lab to build and implement an end to end digital radio that would send data from multiple users over a single waveform with the end goal of transmitting at least 10 bits per second. At a very high level, such a system encodes a bit sequence into an analog waveform to transmit across a medium. That signal is then sampled by the receiver where the message is processed and decoded. The method of sharing the spectrum that was chosen for this exercise was CDMA (Code Division Multiple Access). This is a technique used to reduce the throughput of a protocol where user data is encrypted with a pseudorandom "chip sequence" before being transmitted to "spread" the spectrum [1]. The receiver is able to decode the message if it has knowledge of the chip sequence.

Equipment and Parts

- 8 Ω speaker
- Single-jack microphone
- Computer with MATLAB

1 Introduction

The purpose of this laboratory experiment was to design and implement an end to end digital receiver in MATLAB that would allow multiple users to broadcast different messages over a common channel. We were to design and test the system with two other groups who were to also send and receive their signal over a single audio signal with the goal of transmitting at least 10 bits per second. Such a multi-user system is very important in practical, real-world applications because of the need to maximize the bandwidth of communication systems. With millions of people sending data wirelessly at any given moment, being able to combine multiple transmissions onto a single signal becomes paramount. Some of the most commonly used techniques of accomplishing this in practice are Time Division Multiple Access (TDMA) where each user's data is assigned to periodic time slots within the message, Frequency Division Multiple Access (FDMA) where different messages are sent over non overlapping frequency bands, and our choice of method, CDMA. CDMA is a form of PAM (Pulse Amplitude Modulation) where each user broadcasting on the channel is assigned a certain pseudorandom sequence of numbers called a "spreading code" or "chip sequence" [1]. After the users bit sequences are mapped into ±3 and ±1 PAM symbols, symbols are then multiplied by the chip sequence corresponding to that user, and the messages are summed together before being sent from the transmitter at a single carrier frequency. So long as the codes for all users on that channel are all orthogonal to one another, the each users can be recovered on the other side by a receiver that knows that user's code. The multiplication by the chip sequence spreads spectrum

of the messages out and makes the system more resilient to signal jamming, since the message is essentially encrypted with the chip sequence as the key [1]. A single cable transmission line carrying multiple phone is a practical example of how CDMA is used in practice.

The system for this lab consists of a transmitter module and a receiver module for each of the three sub groups, both of which we implemented in MATLAB. The transmitter is designed to be called from a main transmitter script that gives it a bit sequence. Our transmitter script then encodes those bits to a MATLAB representation of a single user's chip encoded audio signal at a carrier frequency that it returns to the main transmitter. That signal is then mixed with the chip sequence encoded message for the other groups before being played by as an audio tone and recorded. The receiver module then analyzes and decodes the analog waveform back into a bit sequence.

2 Methods

The designs of our digital transmitter and receiver are detailed in this section. The transmitter accepts a random sequence of bits as an input, then encodes the data into a single waveform. The waveforms of each of our three teams are combined into one, then transmitted via a speaker and recorded via three microphones on three separate computers [2]. The receiver performs DSP (Digital Signal Processing) on the recorded signal to determine the original message sent. The MATLAB codes for the transmitter and receiver are found in Appendices A and B, respectively.

The message transmission incorporates CDMA with four users, which means that four symbols are sent for every one symbol a user desires to send. We chose this approach as CDMA allows multiple user messages to be encoded into one signal relatively simply, and we already had experience programming a CDMA system in MATLAB prior to this laboratory exercise. Prior to transmitting, we encode the symbols to send with hamming pulses and modulate the signal with a carrier frequency of 7 kHz. We encode the symbols in this way as hamming pulses have adequately low frequency magnitudes outside their fundamental frequencies and we believed it would be straightforward to sample a received signal on the peak of every hamming pulse to recover the original symbols. An example of a hamming-encoded data sequence of 12 symbols is displayed in Figure 2.1. We chose the carrier frequency of 7 kHz as we knew our speakers had an acceptable frequency response at that frequency.

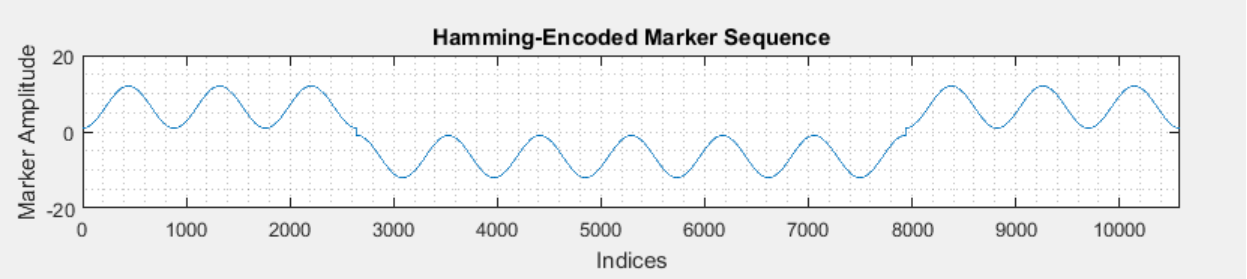

Figure 2.1 – Hamming-Encoded Data Sequence

A marker sequence, displayed in Figure 2.1, is used as a preamble to the data transmission so that the system can begin processing the data at the correct point in time. After the receiver records the incoming transmission, we cross-correlate the recording with a hamming-encoded and modulated marker to determine the location of the identically encoded and modulated marker in the recording (Figure 2.2). We used this approach as it could consistently find the location of the marker in the recording, especially after improving its "peakiness" by squaring the result.

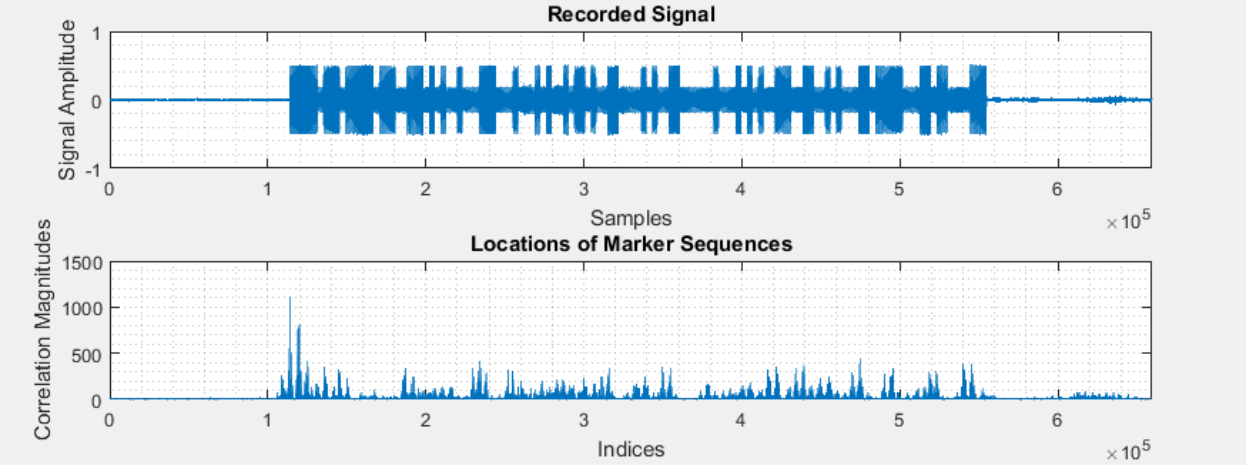

Figure 2.2 – Recorded Signal with Marker Location Acquired

After recording the signal and finding its marker, we demodulate and low-pass filter the recording to bring it back down to baseband such that it can be analyzed. We also trim the recording to remove redundant information. These operations are displayed in Figure 2.3. Figure 2.4 presents a zoomed version of the demodulated recording which highlights the marker sequence found in the very beginning of the signal.

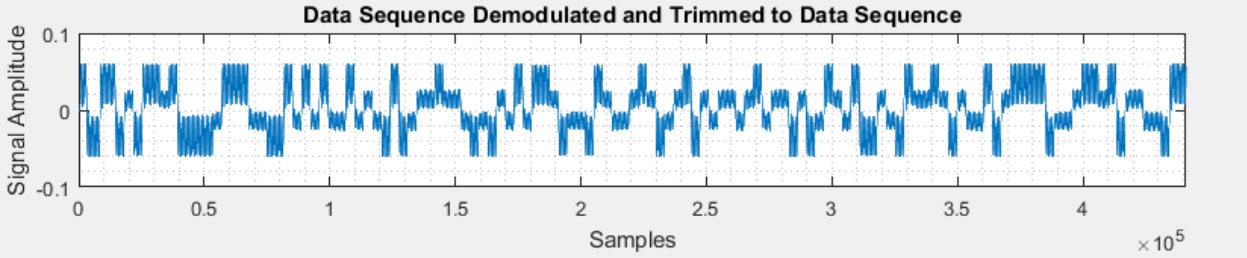

Figure 2.3 – Demodulated, Filtered, and Trimmed Recording

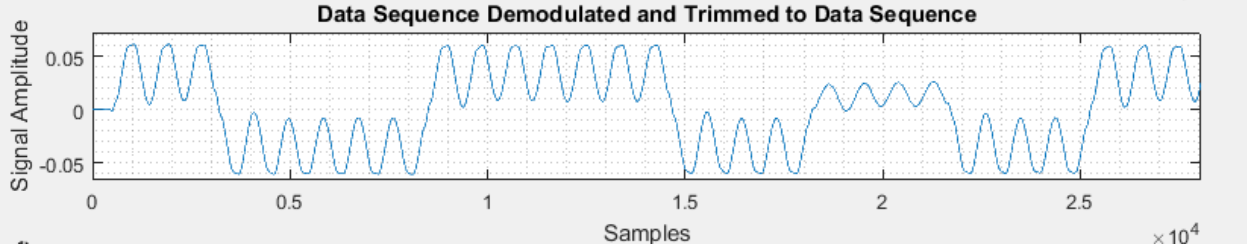

Figure 2.4 – Zoomed Demodulated, Filtered, and Trimmed Recording

To clean up any remaining noise present in the recording, we cross correlate the recording with a hamming pulse identical to the one used to encode the data originally. Figure 2.5 displays the result of this operation, where the signal is also trimmed to start at the first peak of the marker. Figure 2.6 is a zoomed version of Figure 2.5 and displays this effect clearly, especially when compared to Figure 2.4.

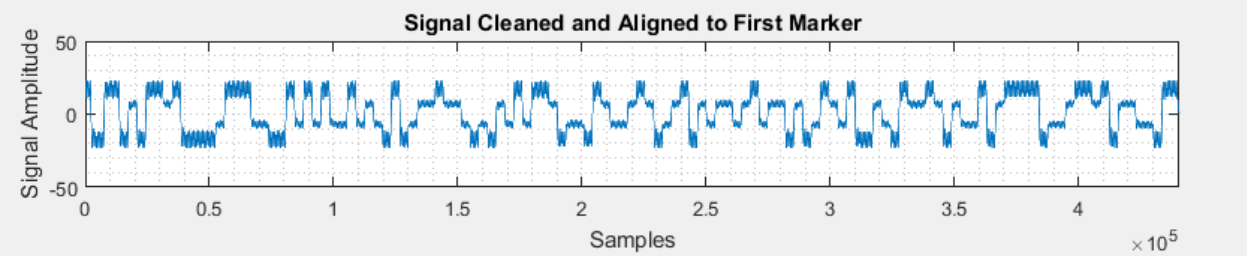

Figure 2.5 – Cleaned Recording Aligned to First Marker Peak

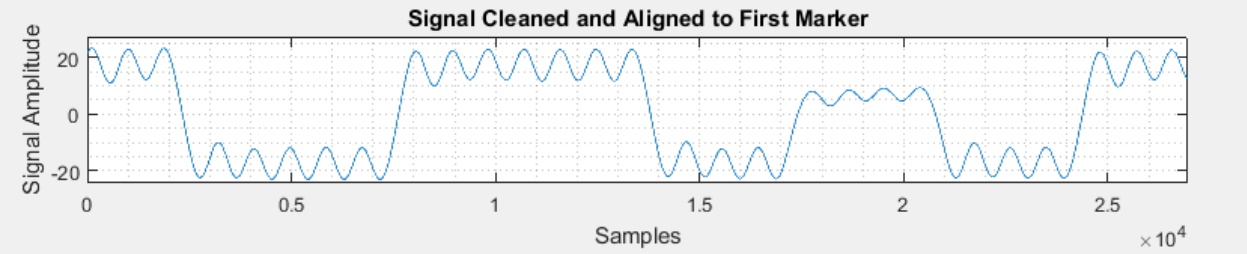

Figure 2.6 – Zoomed Cleaned Recording Aligned to First Marker Peak

Qualitatively, we noted that it was only possible to detect the symbols present in the processed recording relatively as the speaker/microphone system would alter the amplitude of the encoded symbols. To scale the signal back to its intended amplitudes, we scale the entire signal by the ratio of the detected marker amplitude versus its intended amplitude. Additionally, we trim the marker out of the signal such that the first point in the remaining signal is the peak corresponding to the first symbol of the original data sequence. Figure 2.7 displays the result of these operations.

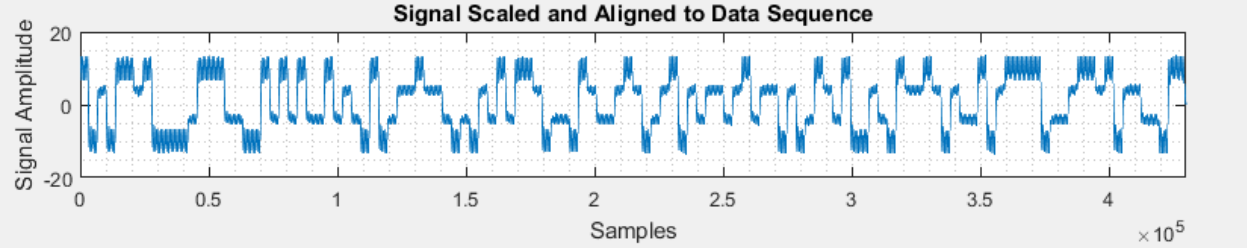

Figure 2.7 – Scaled Recording Aligned to First Data Peak

We then sample the signal at every hamming-pulse peak to begin to recover the original data sequence. Since the signal is trimmed to start at the first symbol's peak in the data sequence, we discard all but every "Mth" point in the signal, where "M" represents the number of indices in one hamming pulse. These results are presented in Figure 2.8.

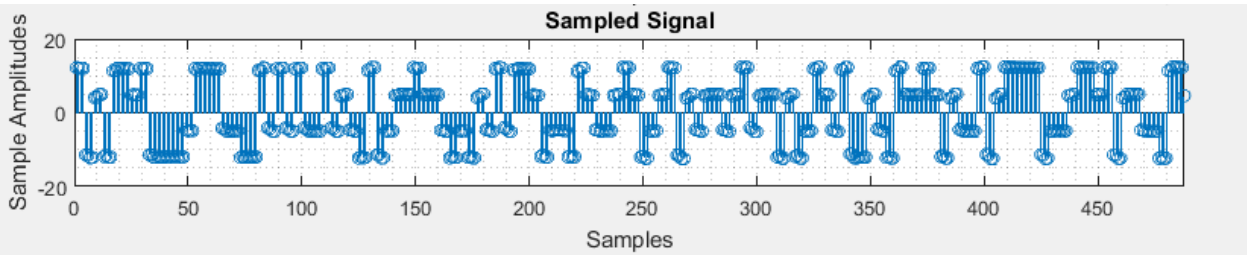

Figure 2.8 – Sampled Recording

We then quantize the signal to even numbers (Figure 2.9) and decode the detected symbols to a single arbitrary user using CDMA techniques. Lastly, we quantize the results once more to the expected PAM symbols (±3, ±1) and convert the symbols back to their corresponding bits. The final result of the receiver in symbol form is displayed in Figure 2.10.

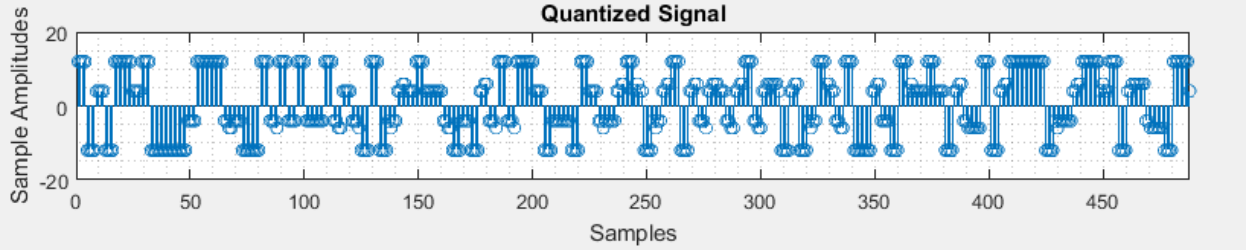

Figure 2.9 – Quantized Signal

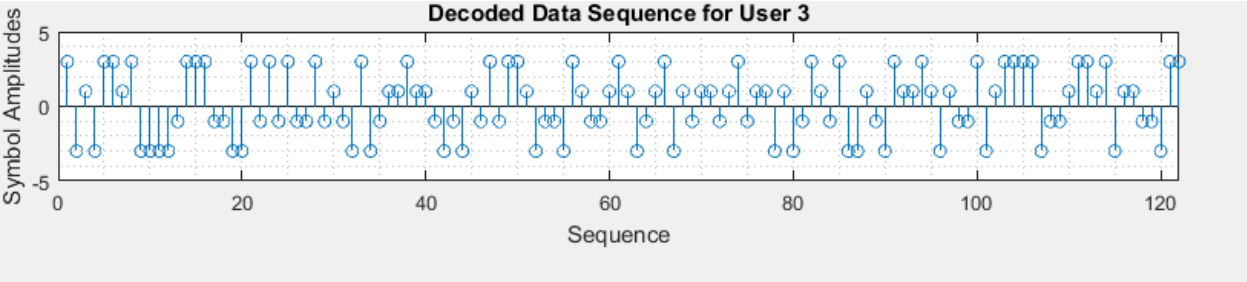

Figure 2.10 – Decoded and Re-Quantized Signal

3 Results

The results of our receiver design are detailed in this section. When we tested our transmitter and receiver on a single computer, we were able to perfectly decode a randomized test sequence of 122 symbols (244 bits). An example of a properly processed signal with decoded message in a single-computer test environment is displayed in Figures 2.1 to 2.10 in Section 2 of this report. When we mixed our signal with the signals from the other groups in our teams however, we ran into issues and neither us nor the rest of our larger group could decode our messages. Initially, our groups had an issue with transmitter similarity, and we were not outputting the same analog signal when given the same bit sequence. After this was fixed and verified, the receivers were still unable to decode the message error free. The closest we came was 130 out of 144 bits transmitted successfully. Most of the bit errors were seen towards the end and middle half of the message sequence. In examining wave forms during debugging, we noted that the sampled sequence at the receiver sinusoidally decreased and increased in amplitude. Figure 3.1 shows an example of this.

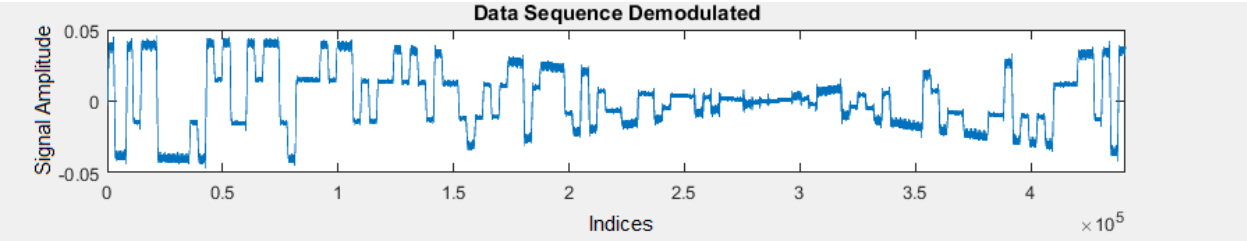

Figure 3.1 – Frequency Mismatch of Two Computers after Signal Demodulation and Filtering

4 Discussion

We believe the main culprit of the issues we ran into with this exercise was an oscillator frequency difference between the two machines. We verified that the designed transmitter-receiver scripts functioned while using the same computer, but when separating the transmitter and receiver to two computers we experienced a frequency mismatch on our recorded signal. This created a "fish-eye" effect which ruined the received symbol amplitudes, as illustrated in Figure 3.1. This is due to the imperfections in the clock rates of the two computers we used. Ideally, we desire to sample the received signal on every hamming-pulse peak. If the sampling rate is even slightly misaligned with the hamming pulse frequency however, the samples will slowly converge to the troughs of the pulses, then converge back to the peaks. The sinewave shape from the decreasing/increasing amplitudes of the signal in Figure 3.1 has a frequency representing the clock rate difference of the two computers. Splitting up the transmitted data into packets with additional marker sequences would help with this, as every marker sequence would allow the receiver to periodically resynchronize to the incoming data. If the marker sequences were close enough together, it would not allow the receivers sampling instants to drift too far away from the peaks of the hamming pulses. This would avoid the varying amplitude and "fish eye" phenomena.

5 Conclusion

Overall, we enjoyed the open-endedness of the laboratory exercise; it was nice to see the unique direction every team took with their designs. We learned of many techniques to increase the robustness of digital radio communication, involving properly encoding information, separating data from noise, cleaning noise, scaling signal amplitudes, and sampling. The exercise served as a great learning experience towards implementing an end to end radio system. There is much to improve with our design, such as eliminating the "fish-eye" effect on the receiver and increasing the bit rate of the transmitter and receiver. However, our most critical issue with this laboratory exercise was communication with our peers; we gained much experience in navigating the potential issues that may arise with moderately sized group projects such as this. We agreed on a general strategy but neglected to maintain a close enough parallel in our designs. We found it doesn't matter how impressive a design is if it is incompatible with the other designs it must perform with.

References

[1] O. B. Wojuola, S. H. Mneney and V. M. Srivastava, "CDMA in signal encryption and information security," *2016 Information Security for South Africa (ISSA)*, Johannesburg, 2016, pp. 56-61. doi: 10.1109/ISSA.2016.7802929

[2] A. Klein, "EE 460 Handout for Lab 1 - Multiuser Communication and Jamming," Western Washington University, Spring 2018

Appendices

Appendix A: Code Listing for tx1C_collaborative.m

```
function x = tx1C_collaborative(bits)
    MessageTimeLength = 10;        % Time seconds duration of transmitted message.
    Fs = 44100;                    % Set Sample frequency.
    TransmittedMessage = zeros(1,Fs*MessageTimeLength);   % Declare space for the
                                                           % message to transmit
    MessagePreamble = [3 3 3 -3 -3 -3 -3 -3 -3 3 3 3];    % Preamble for message.
    User1CDMACode = [1 1 -1 -1];                          % User CDMA code.
    PreambleLength = length(MessagePreamble);             % Number of bits in
                                                           % preamble
    NumPulses = (length(bits) * 2) + PreambleLength;    % # of pulses that will
                                                          % be transmitted after CDMA
    PulseWidth = floor(length(TransmittedMessage)/NumPulses ); % Width of each pulse
                                                  % to fit the required message length
    PulseShape = hamming(PulseWidth)';      % Define Pulse Shape for pulse shape
                                            % convolution
    pulse_value = [];
    OversampledPreamble = [] ;



    t = 1/Fs:1/Fs:MessageTimeLength;  % Define time vector for cosine modulation.
    fc = 7000;                            % Define carrier frequency.
    CarrierSignal = cos(2*pi*fc*t);   % Create carrier signal for transmitting the
                                       % message

    % Assemble Preamble
    for idx = 1:PreambleLength;
        pulse_value = MessagePreamble(idx) .* PulseShape;
        OversampledPreamble = horzcat(OversampledPreamble, pulse_value);
    end

    % Convert binary sequence to PAM symbols
    MessagePAMSymbols = [];
    MessagePAMSymbols = bin2PAM(bits);

    % Convert PAM symbols into 4 User CDMA code, add Preamble
    MessageCDMASymbols = pam2CDMA(MessagePAMSymbols, User1CDMACode);
    MessageCDMASymbols = horzcat(MessagePreamble, MessageCDMASymbols);

    % Create zero buffer for impulse train.
    ZeroBuffer = zeros(1, PulseWidth - 1);

    % Hold result of creating impulse train
    ImpulseTrain = [];

    % Create Impulse Train - Going to clean up later
    for i = 1:length(MessageCDMASymbols - 1) % Stop one short to nod zero pad the end
        temp = horzcat(MessageCDMASymbols(i), ZeroBuffer);
        ImpulseTrain = horzcat(ImpulseTrain, temp);
    end
```

```
    % Pulse shape impulse train with Hamming window to get CDMA pulse message
    ConvResult = conv(PulseShape,ImpulseTrain);
    plot(ConvResult);
    % Modulate baseband signal with carrier
    TransmittedMessage = ConvResult(1:441000) .* CarrierSignal;

    % subplot(2,1,1);
    % plot(TransmittedMessage);
    % subplot(2,1,2);
    % plot (ConvResult);

    plot(TransmittedMessage);
   % TransmittedMessage =TransmittedMessage;
    DonnyResult = TransmittedMessage;
    x = TransmittedMessage';
    return
end

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Converts binary sequence to PAM with the following mapping:
%
%                   BIN | PAM
%                  -----|------
%                   01  | (-3)
%                   00  | (-1)
%                   10  | (+1)
%                   11  | (+3)
%
% Param(): bin_seq -  The binary seqence to convert to a 4-PAM sequence
%
% Return : pam_seq  -  The 4 PAM sequence equivilant to bin_seq
%
% Donovan Gratton 4-16-2018
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
function pam_seq = bin2PAM(bin_seq)
    symbol_idx = 1;
    % symbols = zeros(1,length(bin_seq)/4);  % Create space for vector to convert to
    symbols = [];
        % Convert each byte into 4 PAM symbols
        for i = 1:2:length(bin_seq)
            % Map the first 2 digits to PAM
            if ((bin_seq(i) == 0) && (bin_seq(i+1)== 1))
                %symbols(symbol_idx) = -3;
                symbols = horzcat(symbols,-3);

            elseif((bin_seq(i) == 0) && (bin_seq(i+1) == 0))
                %symbols(symbol_idx) = -1;
                symbols = horzcat(symbols,-1);

            elseif((bin_seq(i) == 1) && (bin_seq(i+1) == 0))
                %symbols(symbol_idx) = 1;
                symbols = horzcat(symbols,1);
            else
```

```
                %symbols(symbol_idx) = 3;
                symbols = horzcat(symbols,3);
            end
            symbol_idx = symbol_idx + 1;  % Advance indexer for appending mapped PAM
symbol
        end
    pam_seq = symbols;
    return
end


%**************************************************************************
% Converts a PAM sequence to a CDMA weighted PAM sequence
%
% Param(): pam_symbols -  The binary seqence to convert to a 4-PAM sequence
%
% Donovan Gratton 4-16-2018
%**************************************************************************
function cdma_symbols = pam2CDMA(pam_symbols, cdma_code)

    cdma_symbols = [];  % Declare empty space for CDMA sequence

    % Loop over values in PAM symbols, multiply each element by the
    % cdma_sequence, append to running vector
    for i = 1:length(pam_symbols)
        temp_seq = pam_symbols(i) * cdma_code;
        cdma_symbols = horzcat(cdma_symbols, temp_seq);
    end
    return
end
```

Appendix B: Code Listing for rx1C_collaborative.m

```
function bits = rx1C_collaborative(signal, numBits)
    marker = [3, 3, 3, -3, -3, -3, -3, -3, -3, 3, 3, 3];
    %     marker = marker.*4;       %For testing single user case
    User3Code = [1 1 1 1];
    codedmarker = [];
    num_users = 4;
    fc = 7000;
    fs = 44100;
    %     numBits = 244;            %For testing

    %Create hamming pulse
    TransmittedMessage = zeros(1, fs*10);
    NumPulses = numBits*2 + length(marker);
    M = floor(length(TransmittedMessage)/NumPulses);
    p = hamming(M);

    %Encode marker with user code
    m = 1;
    while m <= length(marker)
        newcodedmarker = marker(m);
```

```
    codedmarker = [codedmarker newcodedmarker];
    m = m + 1;
end

%Hamming-encode marker
marker_padded = zeros(length(codedmarker)*M, 1);
marker_padded(1:M:end) = codedmarker;
y = conv(marker_padded, p);
y = nonzeros(y);                %Remove lingering zeros
figure(1);
subplot(511);
plot(y);
grid minor;
xlim([0, length(y)]);
xlabel('Indices');
ylabel('Marker Amplitude');
title('Hamming-Encoded Marker Sequence');

%Record signal for testing
%     rec = audiorecorder(fs, 16, 1); %Create recorder object
%     record(rec);                    %Begin recording
%     pause(15);                      %Record for 15 seconds
%     signal = getaudiodata(rec);     %Store recording

%Perform signal processing on input
signal = y;

%Plot recorded signal
subplot(512);
plot(signal);
grid minor;
xlim([0, length(signal)]);
xlabel('Samples');
ylabel('Signal Amplitude');
title('Recorded Signal');

%Cross correlate hamming-encoded data and marker
t = 0:1/M:(length(y)-1)/M;
y = y.*cos(2*pi*fc*t)';     %Modulate marker
z = xcorr(y, signal);
z = z.^2;                   %Square result to improve "peakiness"
z = flipud(z);              %Correct orientation
z = z(length(signal):length(z)); %Correct position
subplot(513);
plot(z);
grid minor;
xlim([0, length(signal)]);
xlabel('Indices');
ylabel('Correlation Magnitudes');
title('Locations of Marker Sequences');

%Location of largest correlation
[~, ind] = sort(z(1:length(z)), 'descend');
%Shift recorded signal to location of first marker
signal = signal(ind(1):length(signal));
```

```
%Truncate everything past 10 seconds (441000 samples)
signal = signal(1:fs*10);

%Demodulate signal
t = 0:1/fs:10-1/fs;
carrier = cos(2*pi*fc*t);
carrier = carrier(1:length(signal));
signal = signal.*carrier';
%Low pass filter the result of demodulation
fbe=[0 0.01 0.02 1];            %Frequency band edges as a fraction of Nyquist
damps=[1 1 0 0];                %Desired amplitudes at band edges
fl=1000;                        %Filter size
b=firpm(fl,fbe,damps);          %Designed impulse response
signal = filter(b, 1, signal);

%Plot trimmed demodulated signal
subplot(514);
plot(signal);
grid minor;
xlim([0, length(signal)]);
xlabel('Samples');
ylabel('Signal Amplitude');
title('Data Sequence Demodulated and Trimmed to Data Sequence');

%Plot frequency spectrum of demodulated, filtered signal
N=length(signal);                       %Length of the signal
ssf=(ceil(-N/2):ceil(N/2)-1)/(1/fs*N);  %Frequency vector
fx=fft(signal(1:N));
fxs=fftshift(fx);
subplot(515);
plot(ssf,abs(fxs));
grid minor;
xlabel('Frequency (Hz)');
ylabel('Frequency Magnitude');
title('Frequency Spectrum of Demodulated and Filtered Signal');

%Clean up received signal by correlating with hamming pulse
signalLength = length(signal);
signal = xcorr(p, signal);
signal = flipud(signal);                        %Correct orientation
signal = signal(signalLength:length(signal));   %Correct position
signal = signal(M/2:end);                       %Shift result up to first hamming
                                                 %pulse peak
figure(2);
subplot(511);
plot(signal);
grid minor;
xlim([0, length(signal)]);
xlabel('Samples');
ylabel('Signal Amplitude');
title('Signal Cleaned and Aligned to First Marker');

%Scale result to first element of marker
signal = (signal./signal(M)).*12;
```

```
%Jump ahead of marker to sample data
signal = signal(M*length(marker):end);
subplot(512);
plot(signal);
grid minor;
xlim([0, length(signal)]);
xlabel('Samples');
ylabel('Signal Amplitude');
title('Signal Scaled and Aligned to Data Sequence');

%Take every Mth element, set rest to 0
for i = 1:length(signal)
    if mod(i, M) && i ~= 1
        signal(i) = 0;
    end
end
signal = nonzeros(signal);  %Remove zeros for analysis

subplot(513);
stem(signal);
grid minor;
xlim([0, length(signal)]);
xlabel('Samples');
ylabel('Sample Amplitudes');
title('Sampled Signal');

sigQuant = -12:2:12;        %Quantize with resolution of 2
signal = quantalph(signal, sigQuant);
subplot(514);
stem(signal);
grid minor;
xlim([0, length(signal)]);
xlabel('Samples');
ylabel('Sample Amplitudes');
title('Quantized Signal');

%Decode received message
NewMsg = [];
Msg = [];
for j = 1:num_users:length(signal)
    %Take chunks of symbols num_users long
    NewMsg(1:num_users) = signal(j:j+num_users-1);
    %Decode with unique user code
    NewMsg = NewMsg.*User3Code;
    %Take sum of multiplied symbols, divide by num_users
    Msg = [Msg (sum(NewMsg)/num_users)];
    NewMsg = [];
end

%Quantize and plot decoded message
%     Msg = Msg./4;                    %For testing single user case
PAMQuant = [-3 -1 1 3];             %Quantize to PAM symbols
Msg = (quantalph(Msg, PAMQuant))';
subplot(515);
stem(Msg);
```

```
    grid minor;
    xlim([0, length(Msg)]);
    xlabel('Sequence');
    ylabel('Symbol Amplitudes');
    title('Decoded Data Sequence for User 3');

    %Convert symbols to bits
    q = 1;
    bits = [];
    while q < length(Msg)+1
        if isequal(Msg(q), 1)
            bits = [bits 0 0];
        elseif isequal(Msg(q), 3)
            bits = [bits 0 1];
        elseif isequal(Msg(q), -1)
            bits = [bits 1 0];
        elseif isequal(Msg(q), -3)
            bits = [bits 1 1];
        else
        end
    q=q+1;
    end
end
```