EE 460 Lab 3 Report

Digital Communication Systems Final Project

Author: Brian Willis

Lab Conducted 14-May, 21-May, 28-May, and 4-June 2018

1 Introduction

For the Digital Communication Systems Final Project, I constructed and tested the DSP (Digital Signal Processing) portion of the software designed radio *Mix 'n' Match Mostly Marvelous Message Machine,* or M^6. While it does not perform in real time, it overcomes transmitter phase noise, transmitter-receiver frequency and pulse shaping variances, transmission channel noise, spectral noise, and intersymbol interference. It encodes messages with 4-PAM (Pulse Amplitude Modulation) square-root raised cosine pulses, and incorporates TDMA (Time-Division Multiple Access) to send multiple user messages with each transmission [2]. My Matlab-constructed receiver decodes three signal test files, each with varying amounts of signal impairments, with a 0% error rate via the provided Matlab receiver tester script [1].

2 Results

My receiver decodes all three provided signal test files with a 0% error rate. The output of my interpolator-decimator that downsamples the received signal after filtering is presented in Figure 2.1, which displays the hard test file. After being downsampled, the signal is immediately quantized to ±1, ±3 PAM symbols and decoded into individual user messages. The hard test file takes the receiver the longest number of iterations to adapt to 4-PAM symbols, but all three test files adapt in less than five frames.

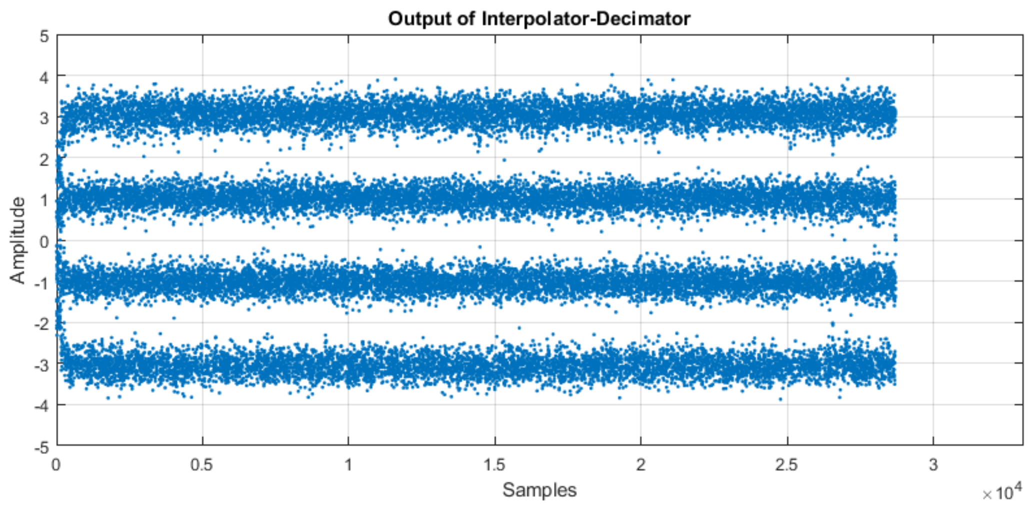

Figure 2.1: Output of Interpolator-Decimator for Hard Test File

3 Discussion

The first signal manipulation in my receiver is the demodulation of the received signal down to baseband. I incorporated a dual PLL (Phase-Locked Loop) to determine the carrier frequency and phase of the received signal as accurately as possible, then demodulated the signal with Equation 3.1:

$$Baseband\ Signal = 2\cos(2\pi f_0 t + \theta_1 + \theta_2) * Received\ Signal \tag{3.1}$$

Where f_0 is the intermediate frequency of the received signal (2 MHz), and θ_1 and θ_2 are the receiver's phases derived from a dual PLL. The dual PLL requires the received signal to be preprocessed in order to accurately determine its carrier frequency, where the signal is squared and BPF-ed (Band-Pass Filtered), shown in Equation 3.2.

$$rp = BPF\{(Received\ Signal)^2\} \tag{3.2}$$

Where rp represents the preprocessed signal. BPF-ing the signal also introduces a phase shift, so I used an FFT to determine the phase shift after filtering and incorporated it into the calculation of θ_1 and θ_2. Equations 3.3 and 3.4 display the operation for determining θ_1 and θ_2.

$$\theta_1[k+1] = \theta_1[k] - \mu_1 rp[k] \sin(4\pi f_0 t[k] + 2\theta_1[k] + \Psi) \tag{3.3}$$

$$\theta_2[k+1] = \theta_2[k] - \mu_2 rp[k] \sin(4\pi f_0 t[k] + 2\theta_1[k] + 2\theta_2[k] + \Psi) \tag{3.4}$$

Where μ_1 and μ_2 are the algorithm stepsizes (chosen to be 0.1 and 0.001 respectively), rp is the preprocessed received signal, and Ψ is the phase introduces by the BPF. The θ_1 and θ_2 convergences are displayed in Figure 3.1, and the frequency spectrums of the received, preprocessed, and demodulated signals are displayed in Figure 3.2.

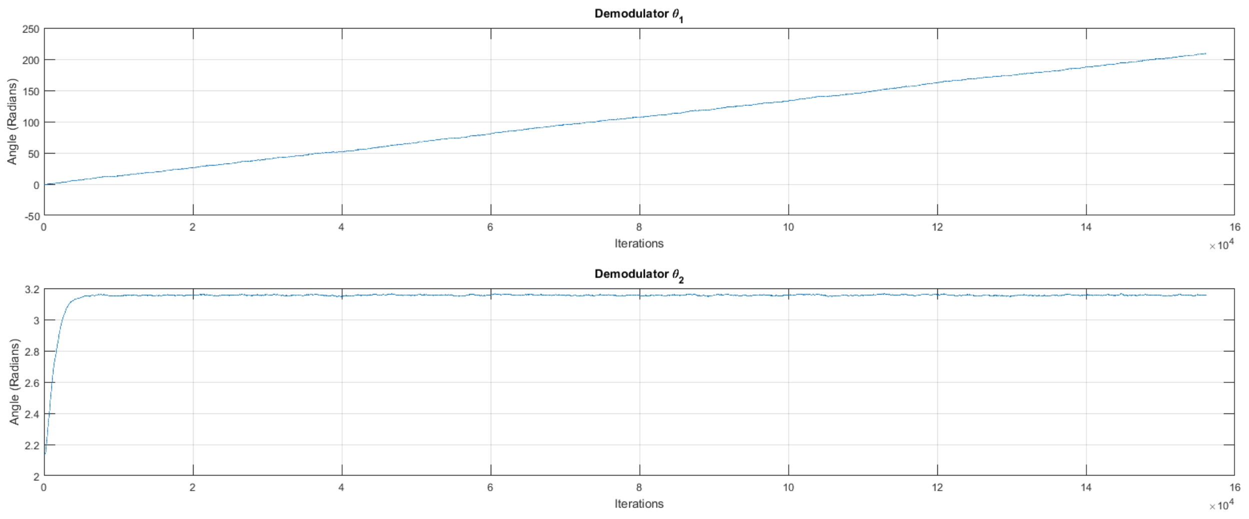

Figure 3.1: θ$_1$ and θ$_2$ Converging to Correct Values in Dual PLL Demodulator

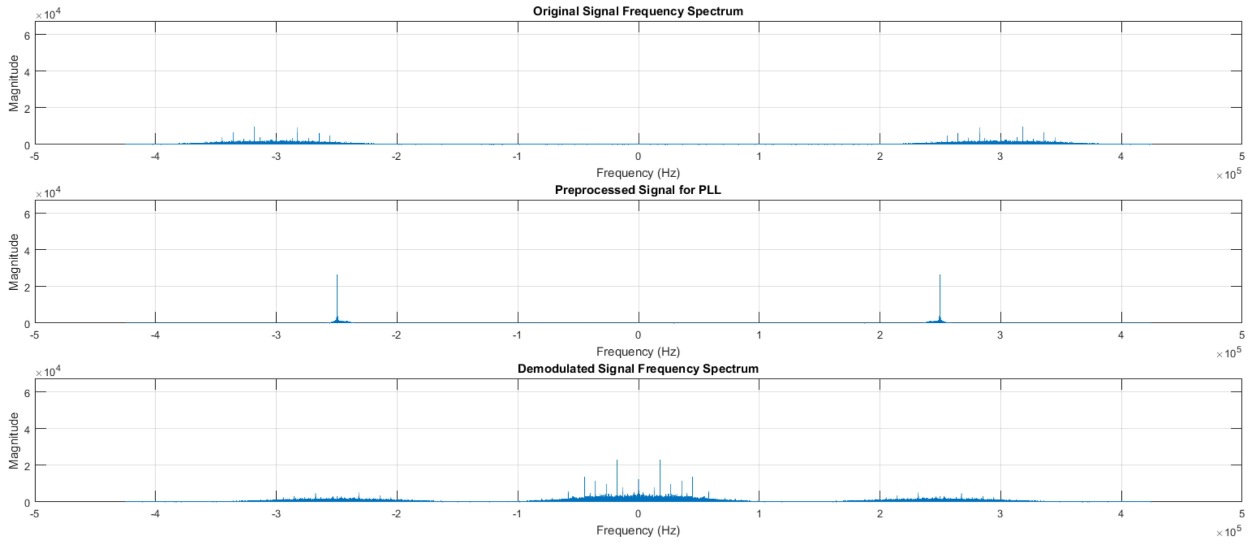

Figure 3.1: Received, Preprocessed, and Demodulated Signal Frequency Spectrums

I also found that a higher order BPF gives the signal a greater phase shift, which negatively impacted the signal's demodulation. While a high-order BPF provides excellent attenuation at undesired frequencies, it also provides too high of a phase shift for the dual PLL. The comparison between a suitable 320-order BPF and an unsuitable 2048-order BPF is illustrated in Figures 3.3-3.6.

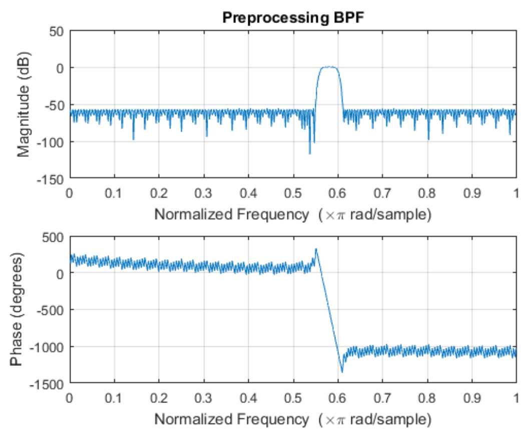

Figure 3.3: Suitable BPF Response

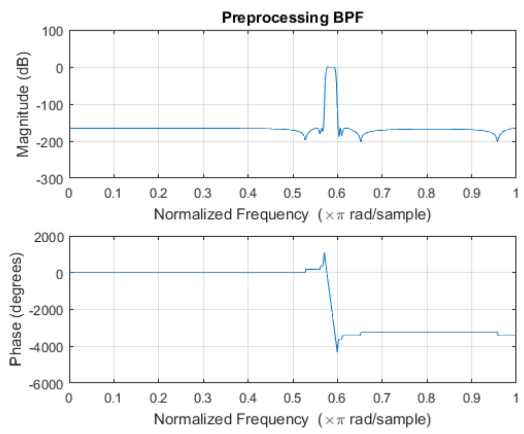

Figure 3.4: Unsuitable BPF Response

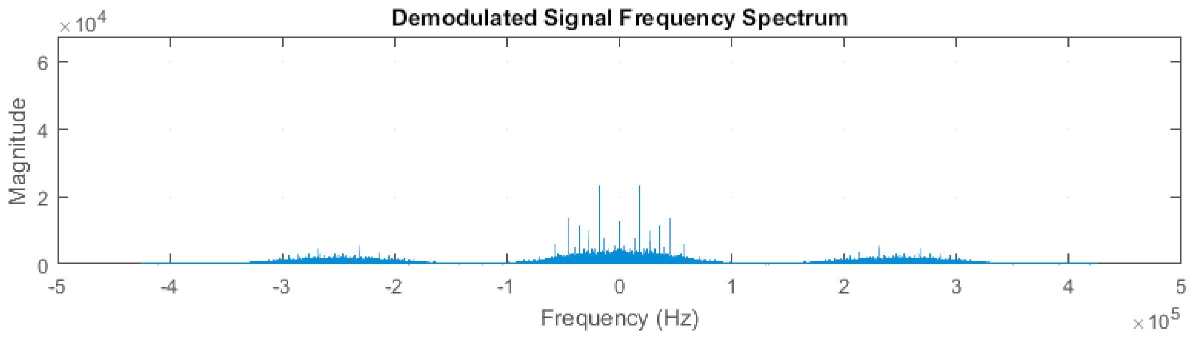

Figure 3.5: Correctly Demodulated Signal via Suitable BPF

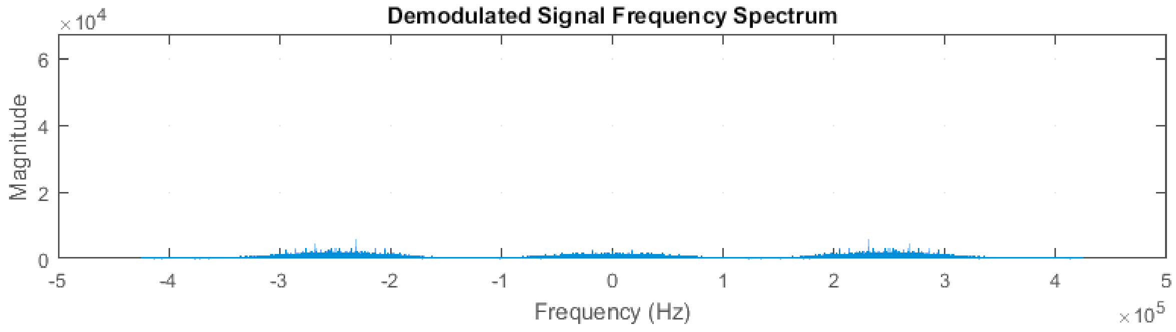

Figure 3.6: Incorrectly Demodulated Signal via Unsuitable BPF

After properly demodulating the signal, I LPF-ed (Low-Pass Filtered) it to isolate the baseband signal, then match-filtered it to improve its SNR (Signal-to-Noise Ratio). I found the LPF should be kept at a lower order similar to the BPF, but its order did not impact the signal nearly as much as the BPF's order did. The match filter was designed by creating a flipped version of the transmitted signal's pulse shape, a square-root raised cosine. Convolving the signal with this match filter significantly increased its amplitude in the time and frequency domains. The LPF response is illustrated in Figure 3.7, the match filter is displayed in Figure 3.8, and the frequency spectrum of the match-filtered signal is presented in Figure 3.9. A comparison of the signal in the time-domain with and without match-filtering is displayed in Figure 3.10.

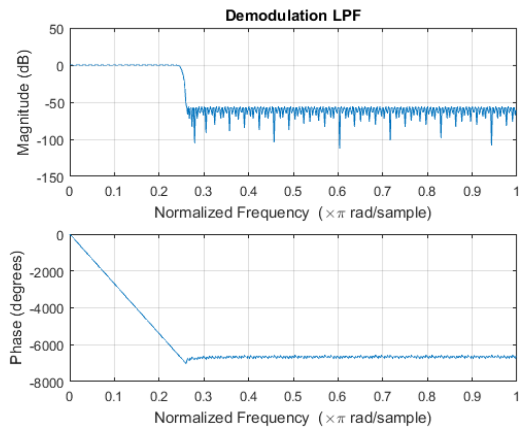

Figure 3.7: LPF Response

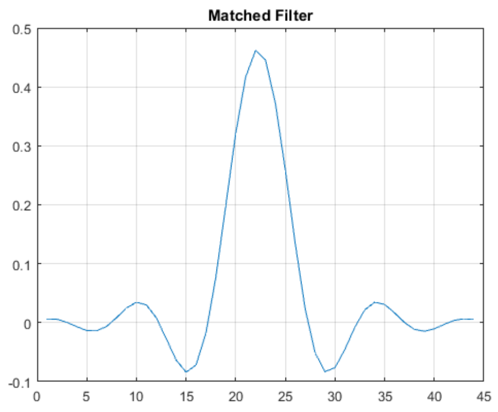

Figure 3.8: Matched Filter Pulse

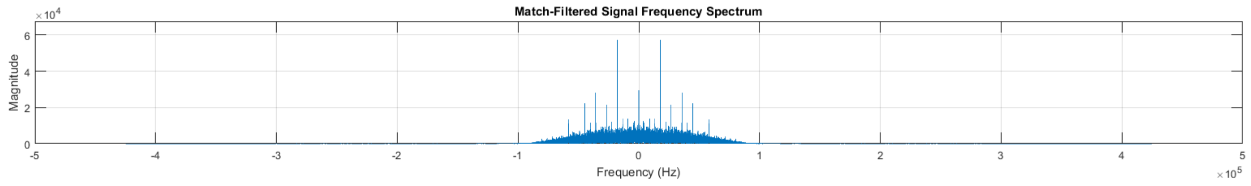

Figure 3.9: Match-Filtered Signal Frequency Spectrum

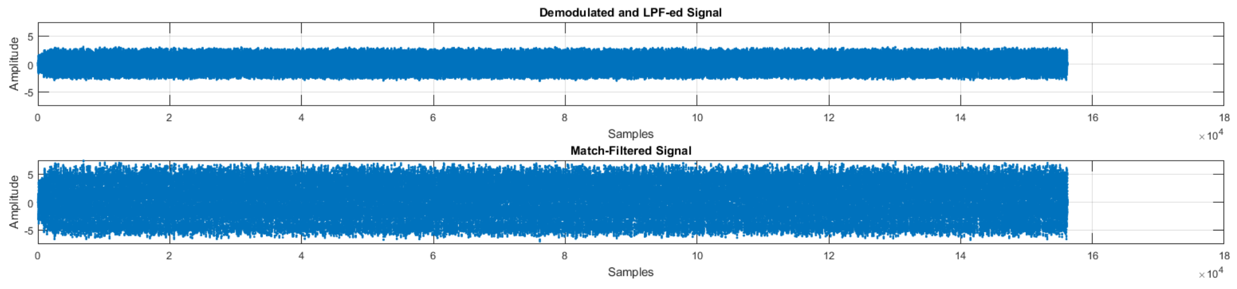

Figure 3.10: Demodulated Signal vs. Demodulated and Match-Filtered Signal

After demodulating and match-filtering the signal, I downsampled it by using an interpolator-decimator. I found the oversampling factor to be 5.44 via Equation 3.5, which means every 5.44th element must be sampled to decimate the signal properly. However, since taking the value at a fraction of an index is impossible in Matlab, I used a sinc-interpolator to estimate the value at each multiple of 5.44.

$$M = \frac{fs}{\frac{1}{Nominal\ Symbol\ Period}} = \frac{850\ kHz}{\frac{1}{6.4\ \mu s}} = 5.44 \tag{3.5}$$

Where fs is the receiver's sampling rate. The interpolator-decimator is an adaptive element as it attempts to maximize its power output, meaning it's sampling at the peaks of each square-root raised cosine pulse the data was originally encoded with. The resulting downsampled signal is then attenuated as to provide data points as close to ±1, ±3 PAM symbols as possible. The code for the interpolator-decimator is provided in Appendix A, and the result of the decimation is illustrated in Figure 2.1.

Lastly, the signal is quantized to 4-PAM symbols and decoded into individual user messages. To locate the preambles that separate each frame, I correlate the preamble's symbol string with the quantized signal, resulting in the preamble peaks displayed in Figure 3.11. If my dual PLL demodulator's θ_2 converged to a value off by π, the preamble peaks will be perfectly inverted. My receiver detects this inversion and re-demodulates the signal with a new θ_2 estimate, which un-inverts the signal without increasing the decoding error.

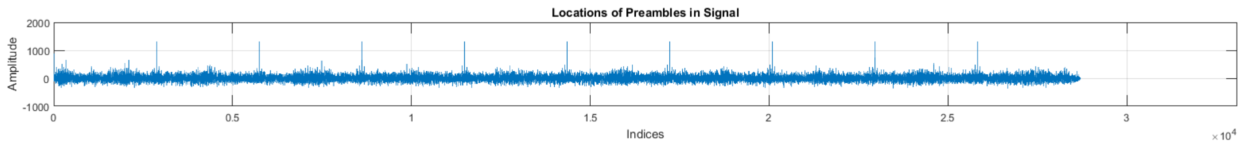

Figure 3.11: Quantized Signal Correlated with Preamble Symbols

These peaks allow my receiver to split the signal into the three user messages via TDMA, then it calls pam2letters2() on each user string to decode the symbols back into ASCII characters.

4 Conclusion

In this lab I was able to gain experience with correcting for signal impairments in a receiver and structuring a complex Matlab-based system. I improved my understanding of filters and adaptive elements, primarily how filter order can affect the phase of a signal and how stepsizes drastically alter adaptive elements' performances. If I were to do this exercise again I would take greater care in designing my filters, and also experiment more with the design of the adaptive portions of the system such as the interpolator-decimator.

References

[1] A. Klein, "EE 460 Handout for Lab 3– Digital Communication Systems Final Project," Western Washington University, Spring 2018

[2] C. R. Johnson, Jr., W.A. Sethares, and A.G. Klein, *Software Receiver Design*, Cambridge University Press, 2011.

Appendix

Appendix A. Matlab code listing for Downsample.m

```
%Interpolate-decimate match-filtered signal to downsample
tnow = srrc_width*M + 1;                                        %Starting sample point
tau = 0;                                                        %Adaptive element
mu = 0.05;                                                      %Algorithm stepsize
delta = 0.35;                                                   %Time for derivative
dnsampled = zeros(1, ceil(length(matched)/M-M));
tausave = zeros(1, ceil(length(matched)/M-M));
tausave(1) = tau;

i = 0;
while tnow < length(matched)-srrc_width*M
       i = i + 1;
       %Interpolated value at sample point
       dnsampled(i) = interpsinc(matched, tnow+tau, srrc_width);

       x_deltap = interpsinc(matched, tnow+tau+delta, srrc_width); %Get right value
       x_deltam = interpsinc(matched, tnow+tau-delta, srrc_width); %Get left value
       dx = x_deltap - x_deltam;                             %Calculate numerical derivative

       tau = tau + mu*dx*dnsampled(i);                       %Output Power algorithm update
       tnow = tnow + M;                                      %Update sample point
       tausave(i) = tau;                                     %Save for plotting
end

dnsampled = 0.72*dnsampled;
```