Binghamton University

Thomas J. Watson School of Engineering

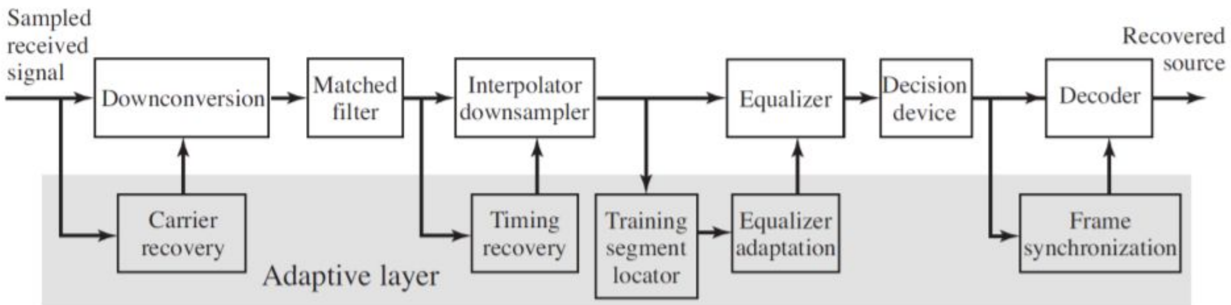
4QPSK Receiver Design Final Project

Software Defined Radio - EECE 580H

Professor Carl Betcher

By:

Jordan Veg & Francesco Summa

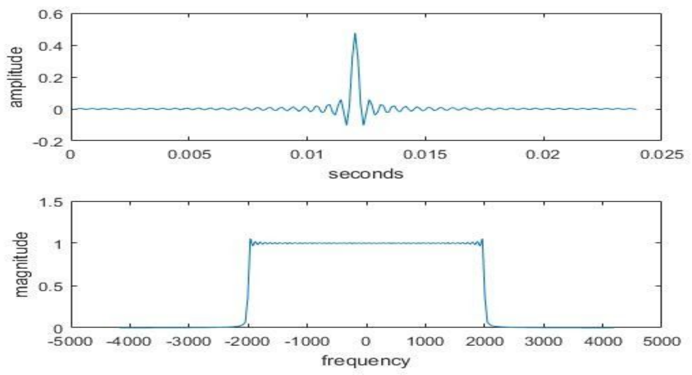


Final Report

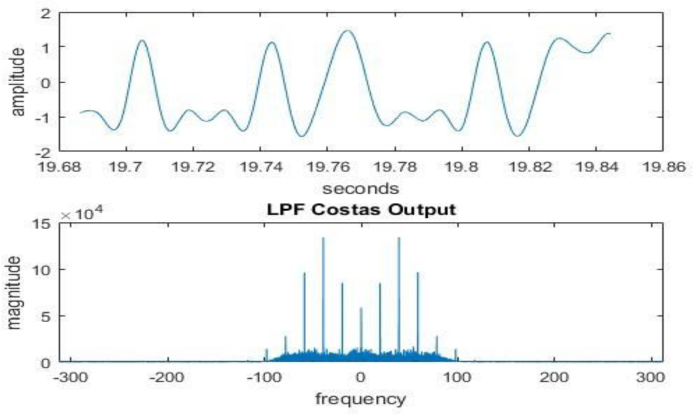
1. 4QAM Receiver Design

I. Carrier Recovery: Costa's Loop

Before the Costa's Loop algorithm is performed, a lowpass filter is applied to the received message signal, eliminating all adjacent frequencies to the message centered at 2000Hz and -2000Hz.



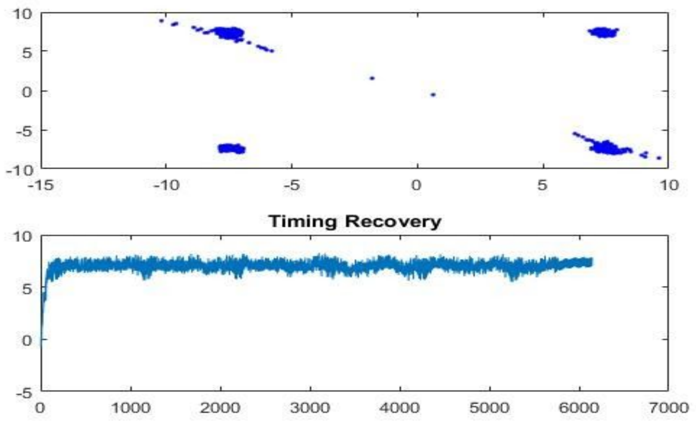
The Costa's Loop is a carrier recovery method to demodulate a signal containing two messages and return two theta values to compensate for any frequency and phase offsets. Our algorithm only handled the impairment of noise. When applying a frequency or phase offset, the system would break and not return a carrier able to demodulate the signal correctly. However, without impairments, the recovery method was able to correctly output a demodulation signal to return the received message signal back down to baseband shown in the figure below. The recovered signal is outputted as a complex-valued sinusoid containing both messages, the real components representing one message and the imaginary component representing the other message.



After the Costa's Loop algorithm demodulates the message to baseband, the same lowpass filter is applied to remove the upper frequencies.

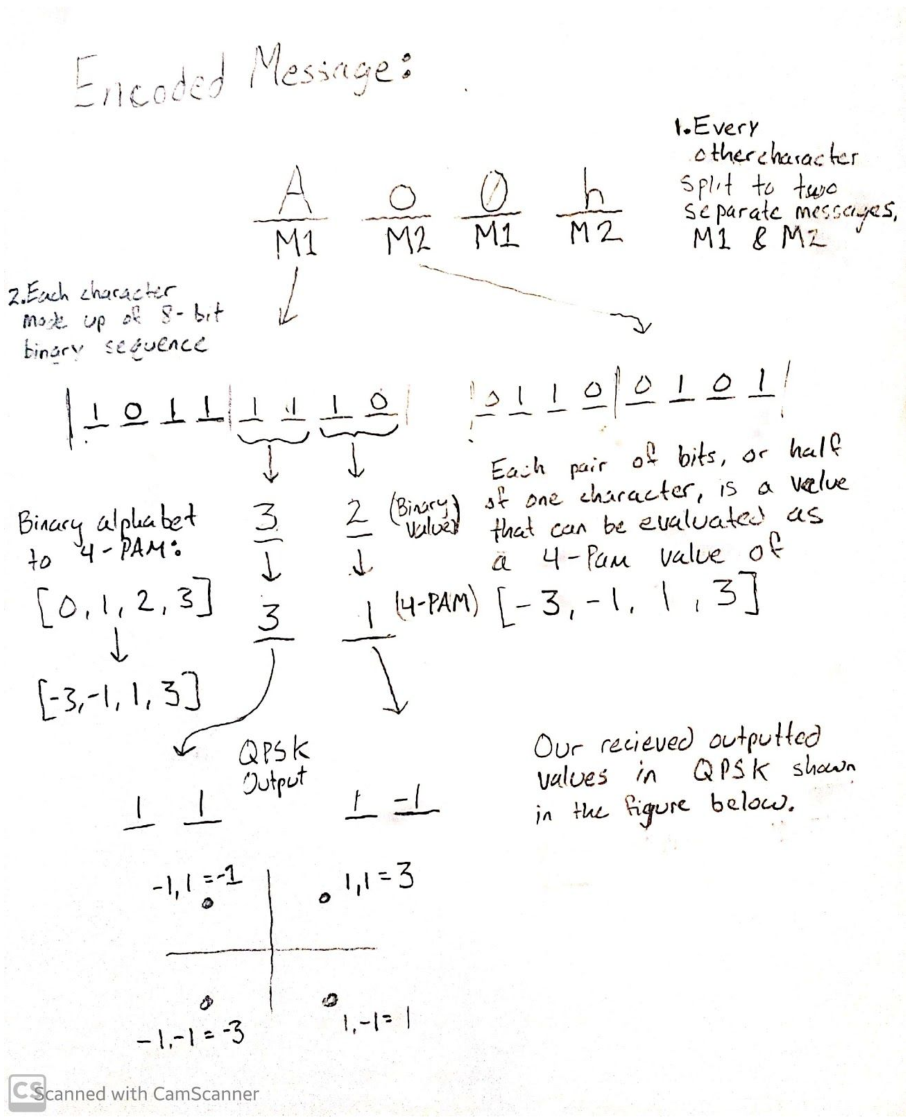
II. Timing Recovery

Given the demodulated signal, the message is downsampled using a time recovery method incorporating SRRC match filtering. Each iteration of the downsampling uses a delta value to surround the sample and pick out the desired value of the message. The algorithm uses a gradient technique to converge to a time-offset that all the points are held at from the assumed sample rate. The figure below represents the placement of each sampled value at their corresponding QPSK constellation value. This allows for the following decoding, frame synchronization, and quantization to occur. The bottom subplot shows the convergence of the sum of the two timing offsets, *tau* and *tau2*, each of which controls the timing recovery of either the real or imaginary portion of the message.

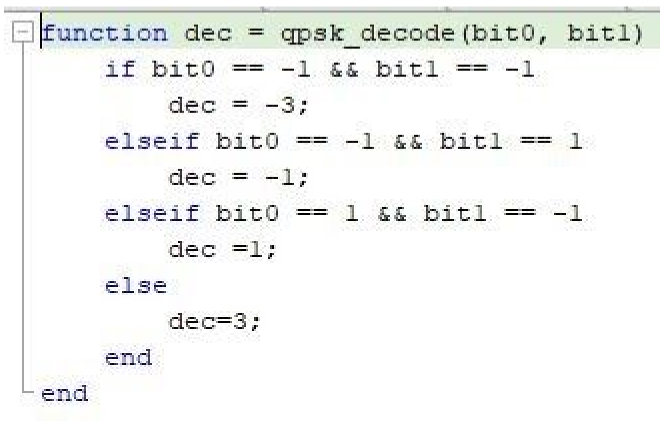


III. Decoding, Quantization, Frame Synchronization

The drawn figure below represents the encoding method used by the transceiver to create the two messages and how it relates to the receiver QPSK output values.

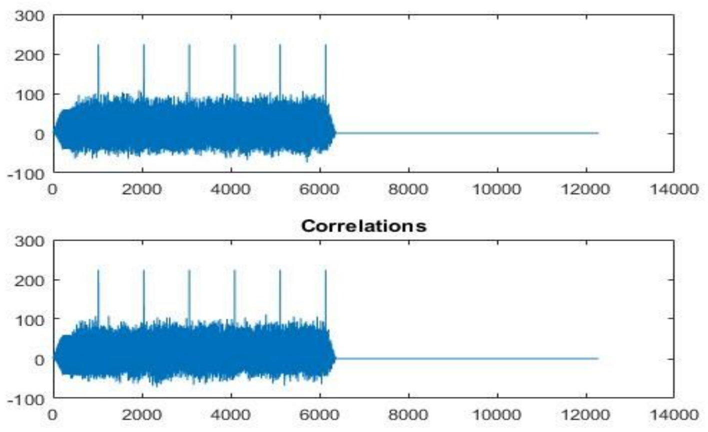

Encoded Message:
A o 0 h
M1 M2 M1 M2
1. Every other character split to two separate messages, M1 & M2
2. Each character made up of 8-bit binary sequence
1 0 1 1 1 1 1 0
0 1 1 0 0 1 0 1
3 2 (Binary Value)
3 1 (4-PAM)
Binary alphabet to 4-PAM:
[0, 1, 2, 3]
[-3,-1, 1, 3]
Each pair of bits, or half of one character, is a value that can be evaluated as a 4-Pam value of [-3, -1, 1, 3]
QPSK Output
1 1
1 -1
Our recieved outputted values in QPSK shown in the figure below.
-1,1 = -1
1,1 = 3
-1,-1 = -3
1,-1 = 1

From the receiver side in decoding the message, we can start from the bottom of the drawn figure in which the values we need encode back into a sequence of characters are QPSK values of -1's and +1's. Every pair of QPSK values is mapped to a 4-Pam value with the alphabet of [-3,-1,1,1]. When that value is decided by our custom function QPSK_decode, the function pam2letters returns the 4-PAM values as the message characters.



```
function dec = qpsk_decode(bit0, bit1)
    if bit0 == -1 && bit1 == -1
        dec = -3;
    elseif bit0 == -1 && bit1 == 1
        dec = -1;
    elseif bit0 == 1 && bit1 == -1
        dec =1;
    else
        dec=3;
    end
end
```

What made this algorithm difficult to design was the fact that every other character in the message sequence belonged to different messages. Looking at the drawn figure from the bottom, each set of 4 QPSK outputs makes up one character of the message. Message 1 is made up of the imaginary component of the recovered message and message 2 is made up of the real component of the message. The preamble in both messages 1 and 2 is used as a header to synchronize the frames of the messages.



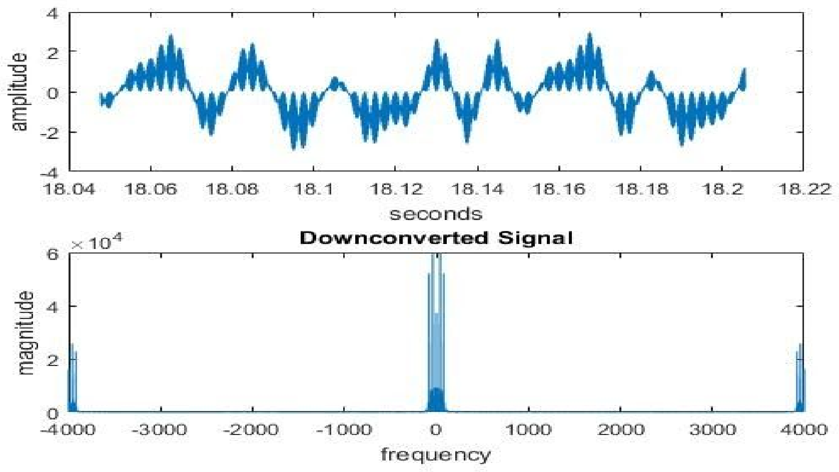
Once the frames are synced, You can decode letter by letter in sets of 4 QPSK values returning two message sequences made up every other message of the full message. Lastly, you can stitch together the complete message by inserting the imaginary, message 1 into all the odd spaces of the array and inserting the real, message 2, into all the even spaces.

2. 4PAM Receiver Design

Before implementing the aforementioned 4QAM using 4PSK design, we created a 4PAM receiver that works flawlessly without any channel impairments with the following design.

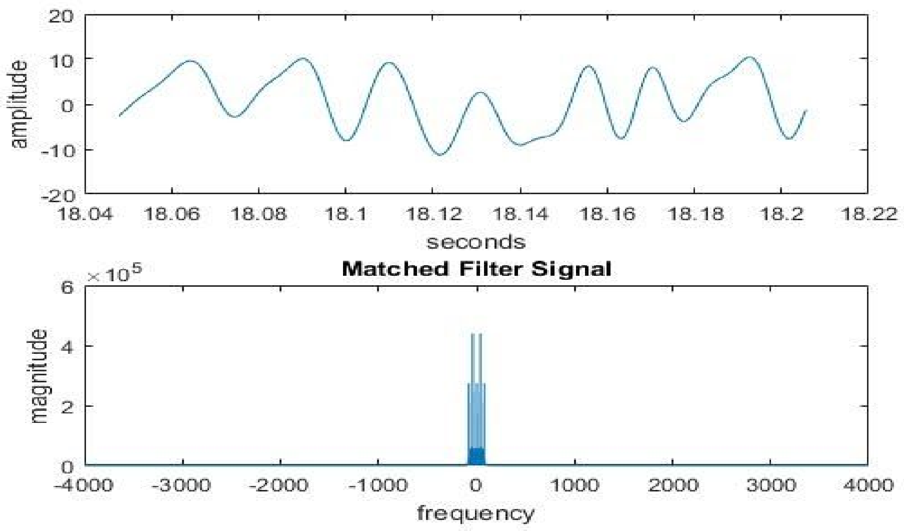
I. Downconversion

The first step in the 4PAM receiver design is to down convert the incoming signal to the baseband using the carrier. Without any impairments this is simple, as you know that the known carrier frequency is exactly what the channel uses. We simply built a carrier cosine signal using the known carrier frequency, and then demodulated with the incoming signal, as you can see this brings the frequency spectrum back down to 0.



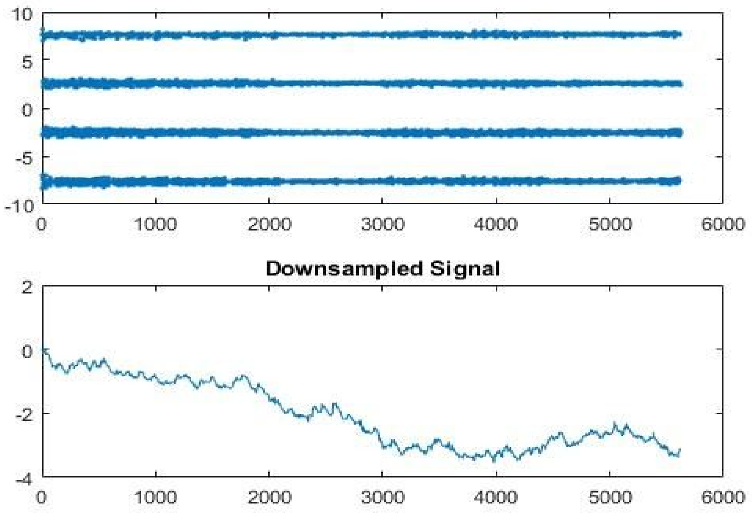
II. Matched Filter

Once the signal is at the baseband, a matched filter is used to remove any unwanted high frequency, as well as match the sent pulse shape. As we know that the transmitter uses an SRRC with length of 4, and a rolloff factor of 0.3. We can directly implement the same filter using the sampling rate as the final input. This filter is then convolved with the downconverted signal to result in the following graph.



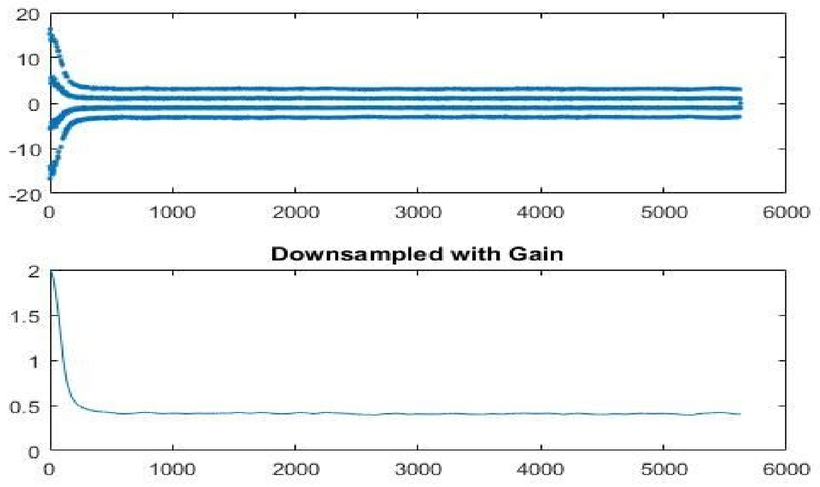
III. Interpolator Downsampler and Timing Recovery

After the signal is pulse shape matched and downconverted, the next step is to downsample and implement timing recovery. Our design implements the *ClockRecDD* method of timing recovery/downsampling. This method consists of taking interpolated samples every *m* points, where m is the estimated symbol rate. It then takes samples at *+delta* and *-delta*, then takes the numerical derivative between those two delta points. This numerical derivative is then used in the *tau* update, which tracks how far away the next sample should be taken. In an ideal transmitter/receiver, this algorithm will always sample every *m* points, which ours does. This results in a figure of a scaled version of the quantized message symbols.



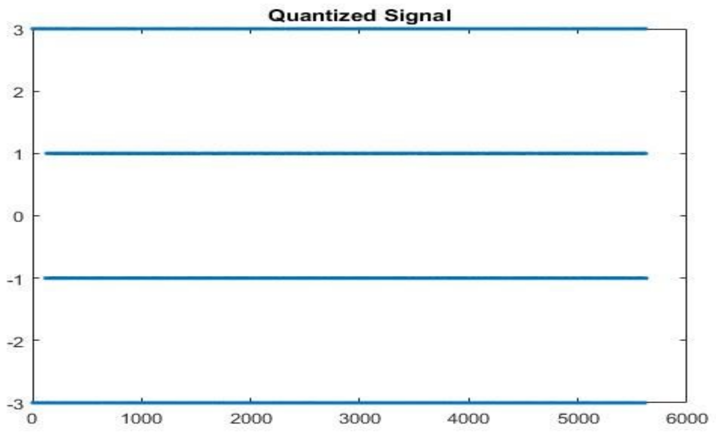
IV. Automatic Gain Control (AGC)

In order to reach the final step of decoding the downsampled symbols, we must first make sure that their amplitudes are within a "quantize-able" range. In the case of 4PAM, we know that we want to be roughly in the range of [-3, 3]. Thus we implemented an AGC to converge the amplitudes of the inputs to the desired range. We use a desired power of $\sqrt{20}$ to accomplish this. In the top half of the following figure you can see the signal converging into their new amplitude "bands", whereas the bottom half of the graph shows the gain value converging to roughly *0.5*. This gain is what allows the aforementioned signal convergence.



V. Decision Device and Decoding

The final portion of the receiver is to make the soft, followed by hard decisions. As we know that this is 4PAM, we know that each value should be quantized to the alphabet of [-3 -1 1 3]. The following figure shows the output of the *quantalph* function, where the quantization of the signal (which has just been put through the AGC) occurs.



Once the values are quantized to the appropriate levels, they can then be decoded via the *pam2letters* function. However, we found that it was first necessary to shift all of the symbols of the quantized signal back one. This allowed for the symbols to line up properly in order to get the correct message decoded out. The only part in which the decoder misses the appropriate message is before the AGC converges. This makes sense because, as the values are outside of the quantized range, they will automatically be pushed to *3 or -3* which results in incorrect symbols. However, once the AGC evens out, then the quantization is able to appropriately assign values to be properly decoded.

3. Conclusions

Once we finished implementing the perfect channel 4PAM receiver, we attempted to move onto implementing the adaptive elements for channel impairments. However after hours of work with nearly no consistent, observable progress we came to the realization that there was simply not enough time to implement this in it's entirety. We attempted multiple different methods for carrier recovery, which would either break the nonimpaired channel, or simply not work. We tried to implement the *Decision Directed Equalizer* for multipath, which not only did not fix the multipath impairment, but also diverged our nonimapired paths. For timing recovery, we would continuously try to edit the step size and delta values to try to catch the central peak values of the baud-error signal. However no matter how many values we tried, the algorithm remained unstable, not converging to any appropriate *tau* value.

That being said, now equipped with a better understanding of how receivers work in general, we were able to build the QPSK receiver. We used the knowledge gained from working on the PAM receiver to adjust elements of the QAM, for now we knew what to look for in terms of algorithm step sizes. The biggest take away from the 4PAM was learning how to decode bits into symbols, and we took this knowledge (plus some extra research) to build the decoding system described above. Overall, we were able to build a 4QAM receiver using the 4PSK method which is robust enough to handle a non-impaired channel as well as a noise impaired channel.