DSP Implementation of Communication Systems

ECPE 4654

Experiment 5
Carrier Recovery Using a Second Order Costas Loop

April 1, 2002

Xuan Chi

Introduction

Phase-lock loops (PLLs) have been one of the basic building blocks in modern communication systems. They have been widely used in communications, multimedia and many other applications. The theory and mathematical models used to describe PLLs come in two types: linear and non-linear. Non-linear theory is often complicated and difficult to deal with in real-world designs.

There are many kinds of Phase Lock Loops; the Costas Loop, which is named by J. P. Costas, a pioneer in synchronous communications, is chosen for this experiment. The reason is that the implementation is quite simple and the structure is very powerful and useful in many situations.

Objective

The objective of experiment 5 is to learn about the Costas Loop and how it works. To do this, we will design and implement a Costas Loop to recover a carrier modulate AM and BPSK signal.

Theory

A. Hilbert Transform:

Theoretically, a Hilbert transform imparts a -π/2 phase shift of the input signal without modifying the magnitude of the input signal. A *Hilbert transformer* is a filter that implements a Hilbert transform. The transformer is defined by the frequency response given in (1) and illustrated in Figure 1.

$$H_H(w) = -j\,\mathrm{sgn}(w) = \begin{cases} -j \ldots w > 0 \\ j \ldots . w < 0 \end{cases}$$

$$|H_H(w)| = 1$$

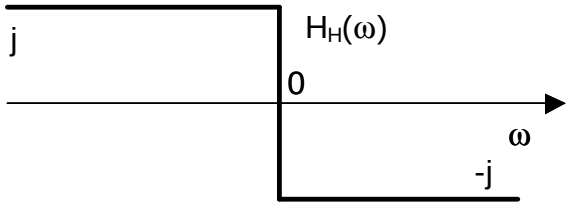


Figure 1. Hilbert Transform Frequency Response.

It is undefined at ω=0, although generally treated as $H_H(w=0)=0$.For notational purposes we will denote the Hilbert transform as H{} and the output of the Hilbert transform of the signal x(t), H{x(t)}, as $\hat{x}(t)$.

When a baseband message signal, m(t), is transmitted on a carrier of frequency w_c, the following relationship also holds

$$\mathrm{H}\{m(t)\cos(w_c t+q)\}=m(t)\cos(w_c t+q-p/2)$$

This can be quickly verified by utilizing the relationships $j=e^{j\frac{p}{2}}$ and $-j=e^{-j\frac{p}{2}}$. Thus in the frequency domain the output of a Hibert transform is given by

$$\begin{aligned}
&=-jM(w)\frac{1}{2}e^{j(w_c+q)}+jM(w)\frac{1}{2}e^{-j(w_c+q)}\\
&=M(w)\frac{1}{2}e^{j(w_c+q)}e^{-j\frac{p}{2}}+M(w)\frac{1}{2}e^{-j(w_c+q)}e^{j\frac{p}{2}}\\
&=M(w)\frac{1}{2}e^{j\left(w_c+q-\frac{p}{2}\right)}+M(w)\frac{1}{2}e^{-j\left(w_c+q-\frac{p}{2}\right)}
\end{aligned}$$

In order to implement a Hilbert transform, we need to express it in the time domain. The impulse response of an ideal Hilbert transform can be found as follows.

$$\begin{aligned}
h_H(n) &= \frac{1}{2p}\int_{-p}^{p}H_H(w)e^{jwn}\\
&=\frac{1}{2p}\left(\int_{-p}^{0}je^{jwn}-\int_{0}^{p}je^{jwn}\right)\\
&=\begin{cases}\dfrac{2\sin^2(pn/2)}{pn}, & n\neq 0\\ 0, & n=0\end{cases}
\end{aligned}$$

Thus an ideal Hilbert transform's impulse response is seen to be

- infinite in duration
- noncausal (as the output depends on future values of *n*)
- antisymmetric (equal in magnitude and opposite in sign)
- nonzero for only odd values of *n*

Because of the infinite duration, it is not possible to directly implement a Hilbert transform. However, it can be closely approximated by windowing the impulse response and implementing the filter using a linear phase FIR with an antisymmetric impulse response. Any of a number of different windowing methods are appropriate. Additionally, because the impulse response is antisymmetric, the frequency response must be 0 at ω=0 for even and odd order filters, and 0 at ω = π for an odd order filter, thus it will also not be possible to implement the Hilbert transform as an all pass filter. However, if the signal to be phase shifted is known -to be limited to a range of frequencies, then the filter can still be appropriately designed.

One method to implement a Hilbert Transformer is

$$h_H(n) = \begin{cases} \dfrac{2}{p(n-g)}\left[0.54 - 0.46\cos\left(\dfrac{2p\,n}{N-1}\right)\right], & n \in [0, N-1], n\ even \\ 0, & n \in [1, N-1], n\ odd \end{cases}$$

where $g = \left\lfloor \dfrac{N}{2} \right\rfloor$ and $\lfloor\ \rfloor$ indicates the floor function.

Another method for designing the transformer is to use the Remez algorithm with the Chebyshev approximation criterion can be employed.

B. Analytic Signal (Pre-envelope)

An analytic signal is defined as the original signal plus j times the original signal's Hilbert transform. Given a signal $x(t)$, its analytic signal, $x_+(t)$ can be constructed as

$$x_+(t) = x(t) + j\hat{x}(t)$$

In the frequency domain, the construction of an analytic signal has the effect of eliminating the original signal's negative frequency components and doubling the positive frequency components. This can also be thought of as folding the negative frequency components into the positive frequency components about ω = 0. This result can be formally expressed as

$$X_+(w) = 2X(w)u(w) = \begin{cases} 2X(w) & w > 0 \\ X(0) & w = 0 \\ 0 & w < 0 \end{cases}$$

C. Complex Envelope

The complex envelope is formed by multiplying the pre-envelope by the complex exponential, $e^{-jw_0 t}$. This has the effect of translating the pre-envelope -w_0 in the frequency domain. If w_0 is chosen as the center frequency of the pre-envelope, then the resulting complex envelope will be translated to baseband. Mathematically, the complex envelope, $\tilde{x}(t)$, of a signal, $x(t)$, is defined as

$$\tilde{x}(t) = x_+(t)e^{-jw_0 t}$$

In the frequency domain this is

$$\begin{aligned} \tilde{X}(w) &= X_+(w + w_0) \\ &= 2X(w + w_0) \end{aligned}$$

Thus with a properly chosen w_0, the formation of the complex envelope can be used to demodulate a signal modulated onto a carrier.

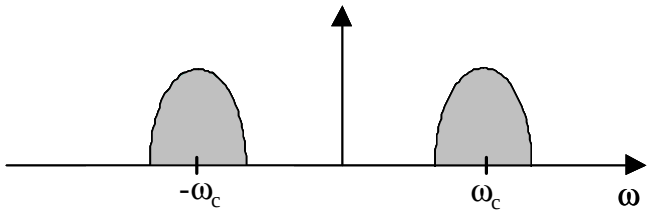
D. AM Example

Consider the reception of an AM signal, $x(t)$, with a message signal, $m(t)$, modulated onto a carrier with amplitude A_c and frequency ω_c, and a phase offset $q_1(t)$. This signal can be expressed as

$$x(t) = A_c\, m(t)\cos\left(w_c t + q_1(t)\right)$$

or

$$x(t) = A_c\, m(t)\left[\frac{1}{2}\left(e^{j\left(w_c t + q_1(t)\right)} + e^{-j\left(w_c t + q_1(t)\right)}\right)\right]$$

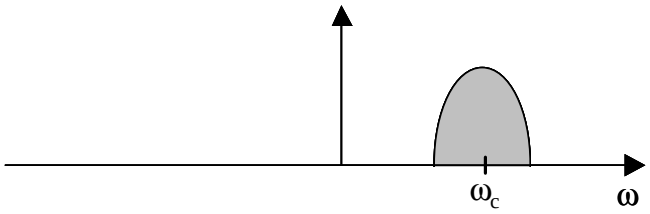


Its Hilbert transform is equal to

$$\hat{x}(t) = A_c\, m(t)\left[\frac{1}{2}\left(e^{j\left(w_c t + q_1 - p/2\right)} + e^{-j\left(w_c t + q_1 - p/2\right)}\right)\right]$$

Its analytic signal is then just

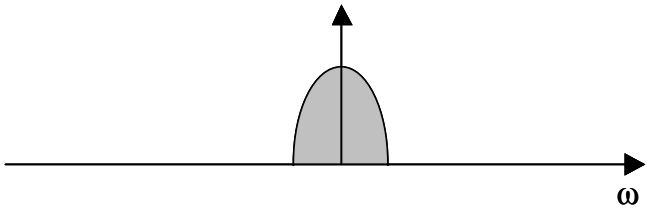
$$x_+(t) = A_c m(t) e^{j\left(w_c t + q_1\right)}$$



Suppose the analytic signal is multiplied by $e^{-j(w_c t+q_2)}$, where w_c is the receiver's estimate of the carrier frequency and q_2 is its estimate of the carrier phase. Presuming the carrier frequency is correctly estimated, the complex envelope is given by

$$\tilde{x}(t) = A_c m(t) e^{j(w_c t+q_1(t))} e^{-j(w_c t+q_2(t))}$$

$$\tilde{x}(t) = A_c m(t) e^{j(q_1(t)-q_2(t))}$$



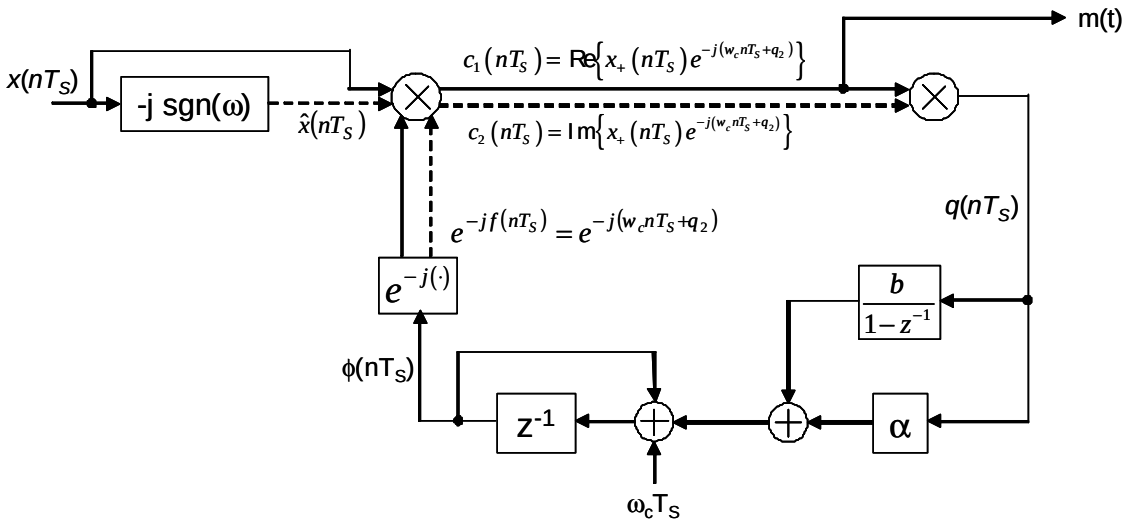
Thus the AM signal is translated to baseband. However, in this expression the message signal is multiplied by a complex exponential. Using the substitutions $\Delta q(t) = q_1(t) - q_2(t)$ and $e^{j\Delta q(t)} = \cos(\Delta q(t)) + j \sin(\Delta q(t))$, the complex envelope can be expressed as

$$\tilde{x}(t) = A_c m(t) \cos(\Delta q(t)) + jA_c m(t) \sin(\Delta q(t))$$

Thus if the receiver's phase estimate exactly matches the carrier's phase, then the real portion of the complex envelope will be the demodulated message signal multiplied by A_c. So presuming the receiver perfectly estimates the phase, the complex envelope can be used to demodulate a message signal transmitted on a carrier.

E. Costas Loop

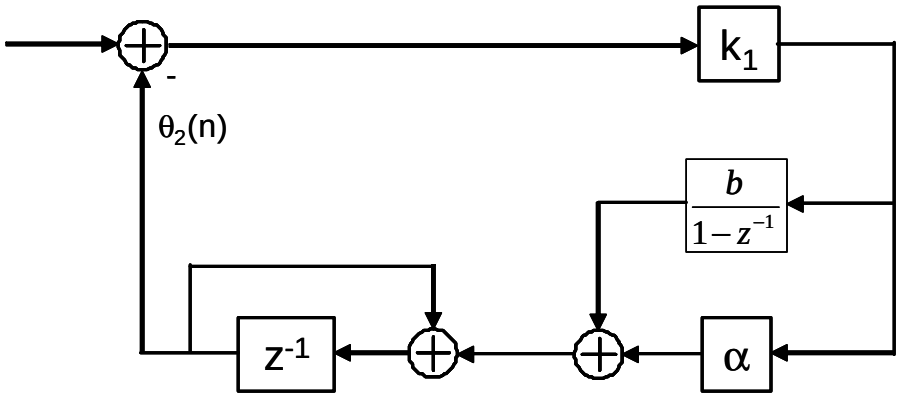
A Costas loop is a type of Phase Lock Loop (PLL) that attempts to transform a carrier modulated message signal into its complex envelope representation in order to recover the message signal. A Costas loop transforms the received signal into its pre-envelope representation. This pre-envelope signal is then multiplied by a complex exponential that using an estimate of the carrier signal's frequency and phase to form the complex envelope. If the frequency and phase of carrier were estimated accurately, the real components of the complex envelope represent the message signal. A block diagram of a second order Costas Loop is shown below in Figure 5.



$$a = 2x\left(\frac{2B_L T_S}{x+\frac{1}{4x}}\right)\ ;\ b = \left(\frac{2B_L T_S}{x+\frac{1}{4x}}\right)^2\ \ ;\ \ s_{q_2}^2 = \frac{B_L T_S}{E_S/N_0}[rad^2]\ ;\ \xi=0.707\ ;\ t_{acq} = \frac{1.2}{B_L T_S}[symbols]$$

The Costas Loop Filter

The Costas Loop filter is used to determine certain loop parameters. In order to determine these, we need a linearized model. The input of the loop is the Hilbert Transformed signal. This is implemented using a simple FIR filter. The figure below is model for the Linearized Loop Filter.



where $k1 = E\{A_c^2 m^2(nT_s)\}$ for AM demodulation scheme where $E\{\ \}$ denotes the expectation operator and $k_1 = A_c$ for the case of BPSK.

The transform function for the loop is

$$H(z)=\frac{k_1 b+k_1 a(1-\frac{a}{a+b}z^{-1})}{1-\{2-k_1(a+b)\}z^{-1}+(1-k_1 a)z^{-1}},$$

where B_L is the loop bandwidth and

$$ak_1=2x\left(\frac{2B_L T_S}{x+\frac{1}{4x}}\right),\ bk_1=\left(\frac{2B_L T_S}{x+\frac{1}{4x}}\right)^2$$

where ξ is the loop damping factor. It is set 0.707 for this experiment. (3dB).

Experimental Procedure, Design, and Results

1. Matlab Procedure

A. Hilbert transformer Design

We used the Hamming windows method and the remez method to design this part.

```
% order of the filter, have to be even for linear phase
order = 32;
```

For Hamming window to get an FIR filter approximating the Hilbert transform filter:

```
N = order + 1;
M = floor(N/2);
h = zeros(1, N);
for n = 0:N-1
       if mod(n-M, 2) == 1
        h(n+1) = 2/pi/(n - M)*(0.54 - 0.46*cos(2*pi*n/(N-1)));
    end
 end
```

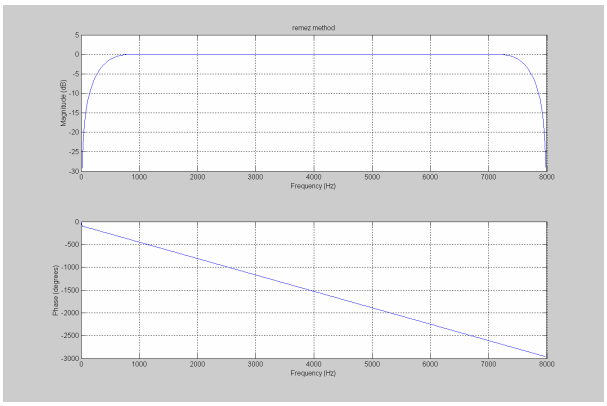
Remez method to get a linear phase FIR filter approximated the Hilbert transform filter:

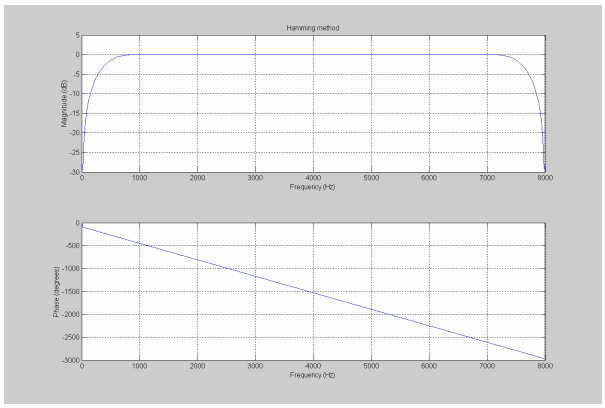
```
N=33;
f = 0.1:0.001:0.901;
a = ones(length(f));
b = remez(order, f, a, 'hilbert');
```

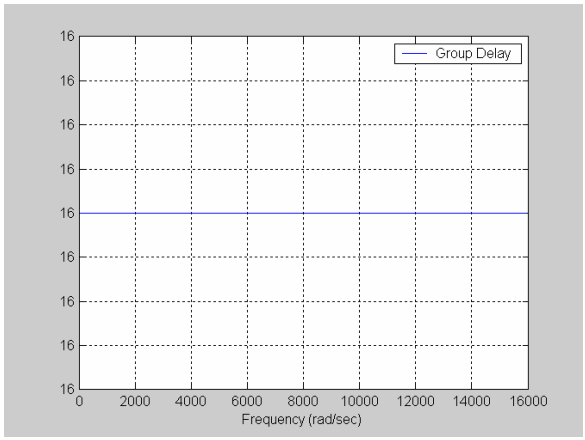
here for each part we calculated the group delay of the FIR filter:
code is :

```
[gd,f] = grpdelay(b,1,512,'whole',fs);
groupdelay=gd(10)
```

The results are:

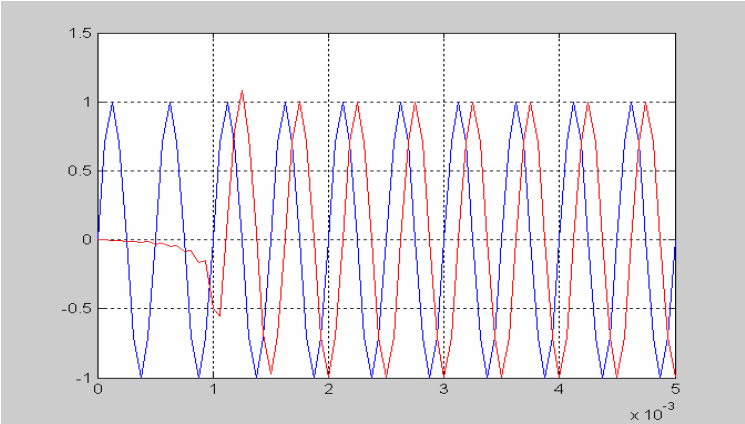







```
Groupdelay = 16

 H = 0.00032135137635  -0.00419833016827   0.00029473706789
-0.00928426633519   0.00044116303941
  -0.01883061296928        0.00052237631816     -0.03439039682080
0.00054705825564  -0.05954005287178
   0.00050104413726  -0.10302466436234   0.00038379079799
-0.19682220874362   0.00020832280911
  -0.63135112186835                  0   0.63135112186835
-0.00020832280911   0.19682220874362
  -0.00038379079799        0.10302466436234     -0.00050104413726
0.05954005287178  -0.00054705825564
   0.03439039682080  -0.00052237631816   0.01883061296928
-0.00044116303941   0.00928426633519
  -0.00029473706789   0.00419833016827  -0.00032135137635
```

To verify the Hilbert transform filter, input a sine wave of 2kHz. The result is a cosine wave which is imparted a 90 degree phase.



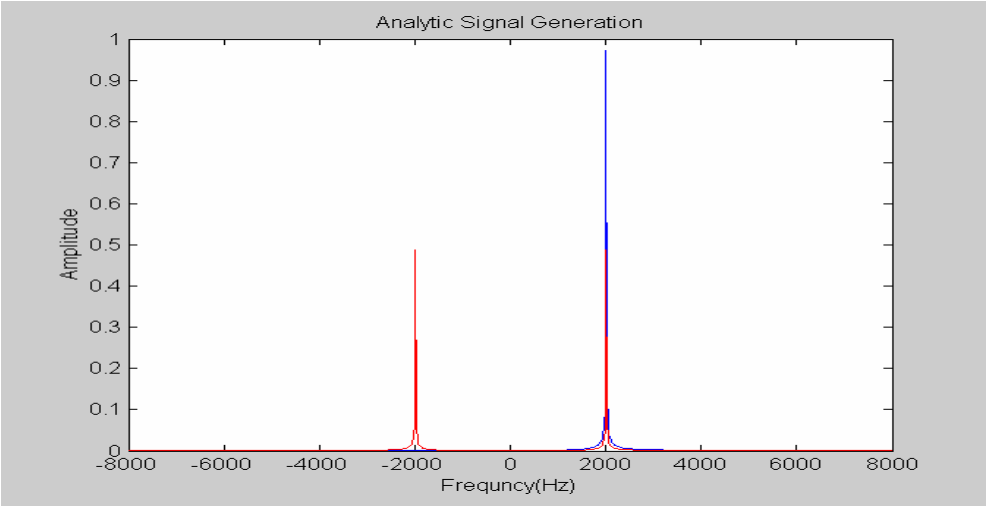
B. Analytic Signal Generation

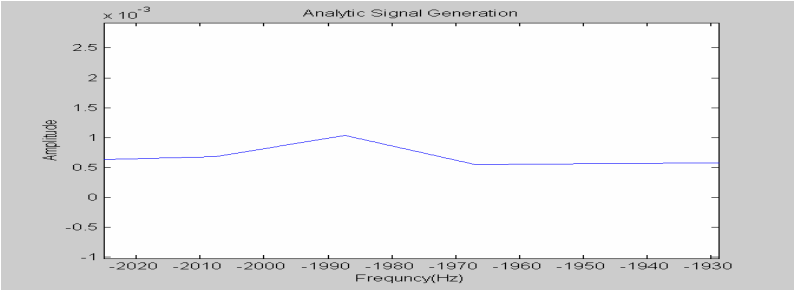
To construct an analytic signal, we need to pass the original signal through a Hilbert Transformer. We then multiply this signal by j and add it to the original signal. This creates the analytic signal. The analytic signal removes all negative parts of the original signal and increases the amplitude of the positive portion of the original signal.

The code is as follows:

```
hilbert_output = filter(b, 1, input);
%groupdelay should be accounted for generating a analytic
signal
analytic = input(1:len-groupdelay) +
j*hilbert_output(groupdelay+1:len);
```

The result is:





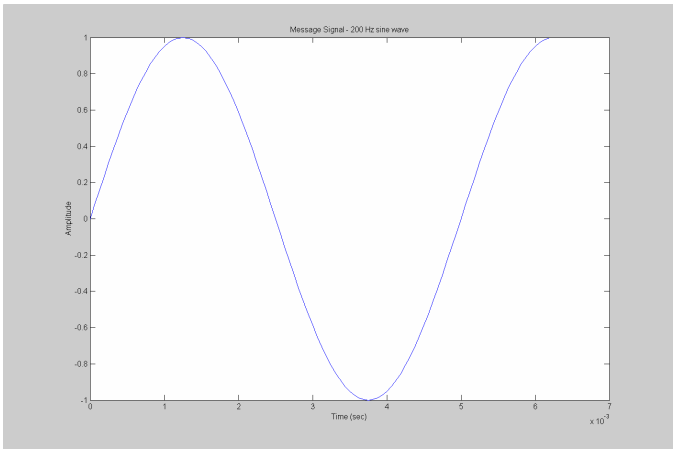
C. Costas Loop Filter Design for an AM signal:

The AM signal's frequency is 200 Hz and the sample frequency is 16000 Hz, amplitude is 1 V.

For the AM part:

```
Fs = 16000; fm = 200; t_span = 0.5;
t = 0:1/Fs:t_span;
Am = 1; m = Am*sin(2*pi*fm*t);
```

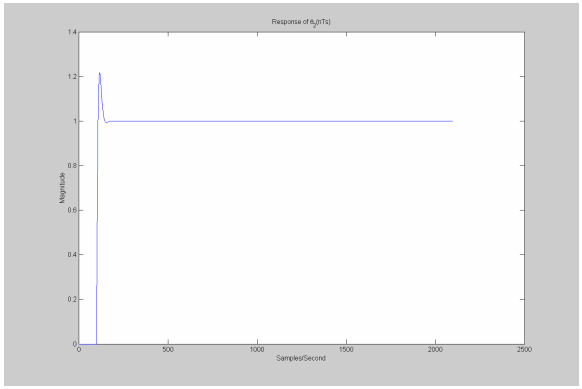
The waveform is:

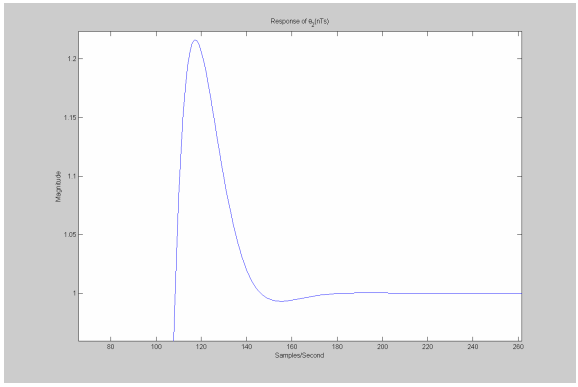


Next we calculate values for k1, beta and alpha and set the value of BL.

For k1, we can use the equation which is given before. For BL, we used several values to test which is best and we decided to use 1000 Hz.

The unit step input response is:

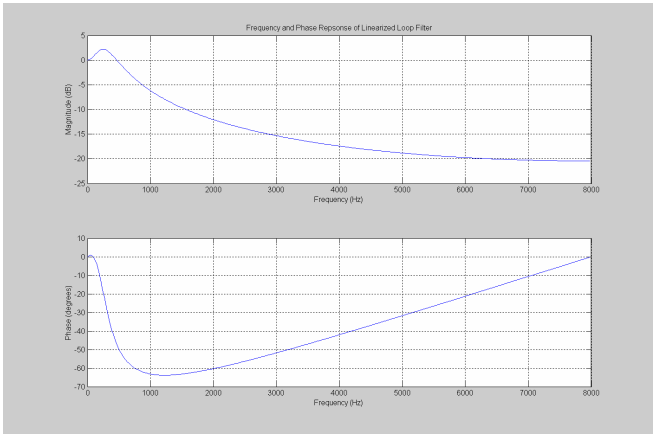




From these graphs, you can see that the waveform meets specs.

After beta and alpha are calculated, we can determine the response of the loop:

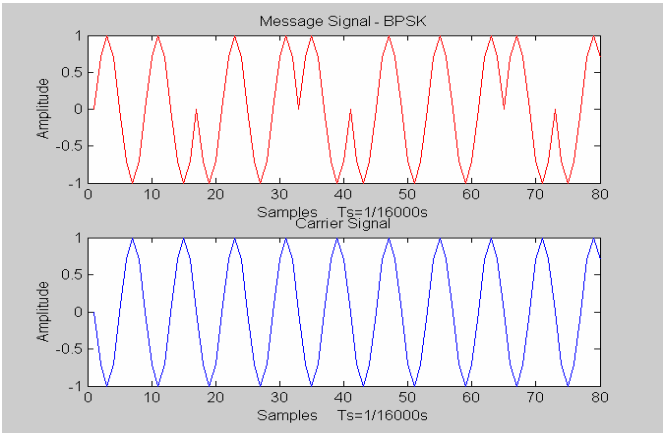
```
BL = 1000;          % Arbitrarily set
zeta = 0.707;       % Set in Specs
k1 = mean(abs((Ac*m).^2))
alpha = 2*zeta*2*BL/(zeta+1/4/zeta)/Fs/k1
beta = ((2*BL/(zeta+1/4/zeta)/Fs))^2/k1
ratio=alpha/beta
Hnum = [ k1*(beta+alpha)  -k1*alpha];
Hden = [1 -(2-k1*(alpha+beta)) (1-k1*alpha)];
LF = filter(Hnum, Hden, [zeros(1,100) ones(1,2000)]);
```



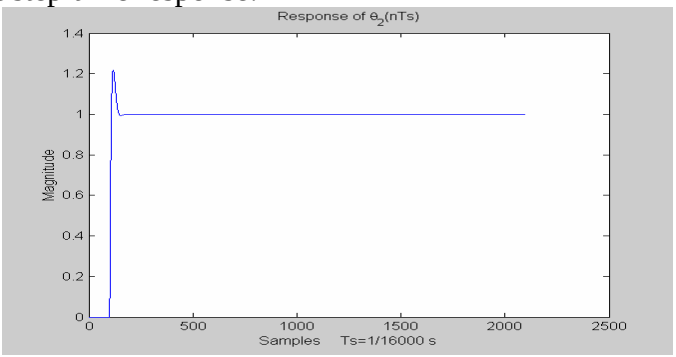
k1 = 0.4999
alpha = 0.3333
beta = 0.02778
alpha/beta = 12
BL = 1000 Hz

D. Costas Loop Design For BPSK:

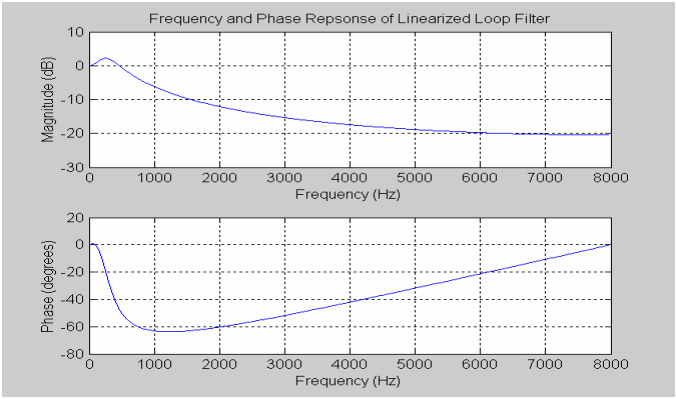
The BPSK signal is found by the same technique used in Experiment 2.
BPSK signal generation:



Costas loop unit step time response:



Costas loop frequency response:



k1 = 0.1
alpha = 0.16665
beta = 0.01389
alpha/beta = 12
BL = 1000 Hz

E. Costas Loop MATLAB Implementation for an AM Signal:

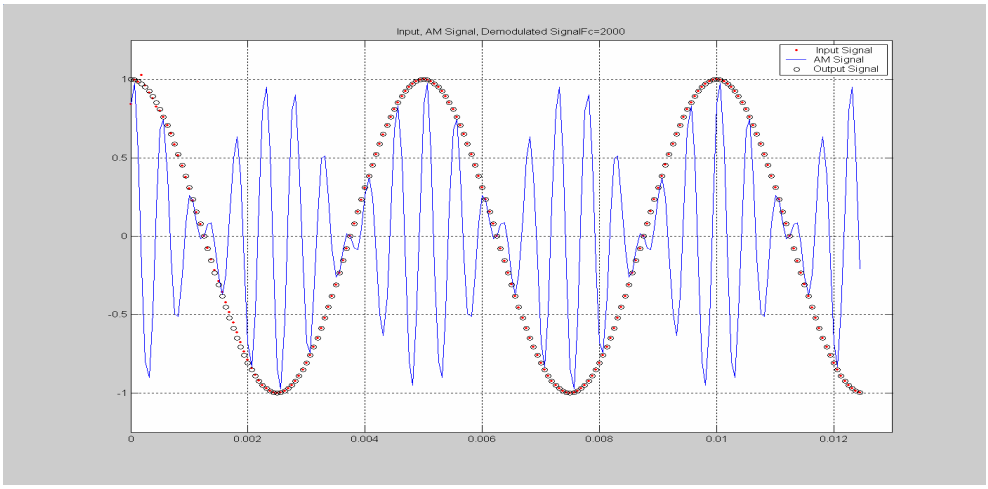
For this part, we input the am signal(f=200, fc=2000) into the Hilbert filter, make AM signal group delay then combine them, time the phase obtained from the loop. The real part is the demodulate output signal.

The detail implementation step is:

```
   m=Am*cos(2*pi*fm*t);---- AM signal generator
   theta=1;
   input = Ac*sin(2*pi*fc*t+theta).*m;  %Input is a modulated
signal

   hilbert_output = filter(b, 1, input);---after Hilbert filter

   %Initialization
   out = [];
   phi = [];
   phi(1) = 0;
   temp_out1=0;
   temp_pre_out1=0;
   temp_out2=0;
   temp_out3=0;

   %Simulation
   for I=1:len-groupdelay
       phi(I)= temp_out3;
       phase(I) = exp(-i*phi(I));
       c1(I) = real(analytic(I)*phase(I));
       c2(I) = imag(analytic(I)*phase(I));
       out(I)= sign(c1(I));
       q(I) = sign(c1(I))*c2(I);
       temp_out1=temp_pre_out1+q(I)*beta;
       temp_out2=alpha*q(I)+ temp_out1;
       temp_out3=2*pi*fc/fs+phi(I)+temp_out2;
       temp_pre_out1=temp_out1;

   end;
```

Theta is the carrier offset phase: if the offset is from 0 to pi, the loop tracks in the same phase with the modulated original signal; if the offset is from pi to 2pi, there will be 180 degree phase shift relative to the original signal.
Set the carrier frequency 1900Hz, 100Hz less than the carrier frequency fc, the loop can still keep track with the input.

The result is:

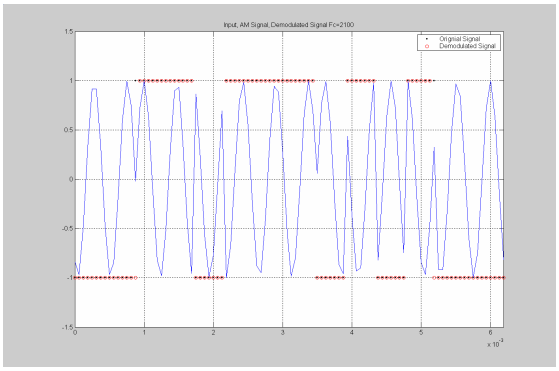


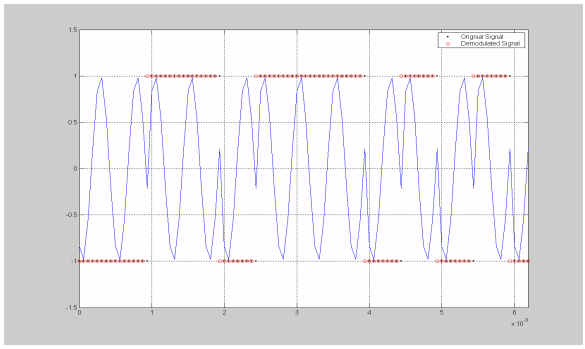
F. **Costas Loop MATLAB Implementation for a BPSK Signal:**

Change the MATLAB loop in the following places:

```
out(I)= sign(c1(I));
q(I) = sign(c1(I))*c2(I);
```

The results are below:





From this picture, we can see the BPSK's bit is [-1 -1 1 1 -1…], the carrier's frequency is 2000Hz, Fs=16000 (there are 8 *, at 1, so it is 2000*8=16000). The modulated signal is 1 sample bit delay than original signal; the reason is the loop time less than or equal to the sampling rate.

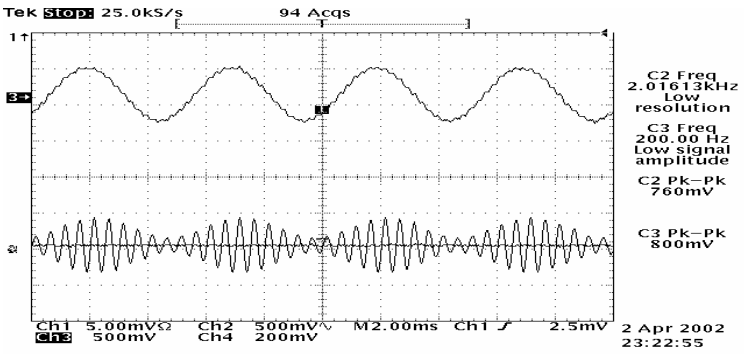
2. C Implementation Procedure

B. Costas Loop Demodulation of an externally Generated AM Signal

```
#define SAMPLING_RATE 8000
#define fc 2000
#define N 33
#define grpdelay 16

float h[N]={0.00032135137635, -0.00419833016827,
0.00029473706789, -0.00928426633519, 0.00044116303941,
-0.01883061296928,
              0.00052237631816, -0.03439039682080,
0.00054705825564, -0.05954005287178, 0.00050104413726,
-0.10302466436234,
             0.00038379079799, -0.19682220874362,
0.00020832280911, -0.63135112186835,                0,
0.63135112186835,
           -0.00020832280911,  0.19682220874362,-
0.00038379079799,  0.10302466436234, -0.00050104413726,
0.05954005287178,
           -0.00054705825564,  0.03439039682080,-
0.00052237631816,  0.01883061296928, -0.00044116303941,
0.00928426633519,
           -0.00029473706789,  0.00419833016827,-
0.00032135137635};
float x[N],buffer_x[grpdelay];
float beta = 0.0277840, alpha = 0.3333414;//Costas Loop filter
parameters
int oldest=0,oldest2=0;
int counter=0; //acount for the groupdelay for Hilbert transform
filter
float phi=0, temp_out1=0,temp_pre_out1=0;
float hilbert_output, c1, c2, q, y;
// Tell compiler where fir_conv function is
extern float conv_fcn(float *filter_coeffs, float
*data_array,int pointer,int order);
/*-------------------------------------------------------
   INTERRUPT SERVICE ROTINES
   ---------------------------------------------------------*/
interrupt void rcvISR(void)
{
    int new_sample;

    /* MCBSP0_DRR is the hardware register where received
samples are stored */
   new_sample = MCBSP0_DRR;

   // Updating data array with new sample
  // Scale the input
```

```
   x[oldest] = ((float)new_sample)*4.0e-9;

   // Increment oldest
   oldest++;
   if (oldest==N)
        oldest=0;

   hilbert_output = conv_fcn(h,x,oldest,N);

    if (counter >= grpdelay)
    {// analytic = buffer_x[oldest] + j*hilbert_output;
    //  phase = cos(phi) +j*(-sin(phi));
    //start processing
    c1 = buffer_x[oldest2]*cos(phi)+ hilbert_output*sin(phi);
    c2 = -buffer_x[oldest2]*sin(phi) + cos(phi)*hilbert_output;
    y = c1;
    q = c1 *c2;
    temp_out1 = temp_pre_out1 + q*beta;
    phi = 2*3.1415926*fc/SAMPLING_RATE + phi +alpha*q +
temp_out1;
    if (phi > 6.283185307)
        phi = phi-6.283185307;
    temp_pre_out1 = temp_out1;

    buffer_x[oldest2] = ((float)new_sample)*4.0e-9;

   // Increment oldest
   oldest2++;
   if (oldest2==grpdelay)
        oldest2=0;
    //end of processing
    }
   if (counter < grpdelay)
        buffer_x[counter++] = ((float)new_sample)*4.0e-9;
   // Scale the output
   MCBSP0_DXR = (int) (y*1.5e9);
}
```

The result is as follows:

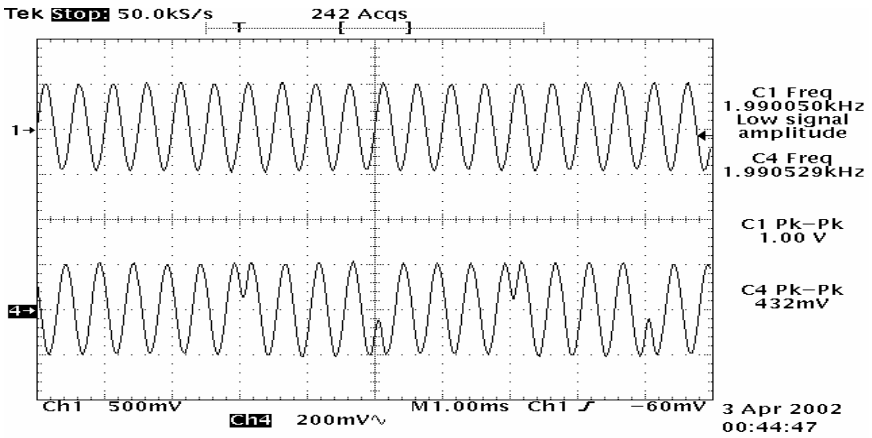


Pull in range: fc_mod = 2000Hz fc_max = 3300Hz fc_min = 400Hz
Fc_mod – fc_min = 1600Hz fc_max – fc_mod = 1300Hz

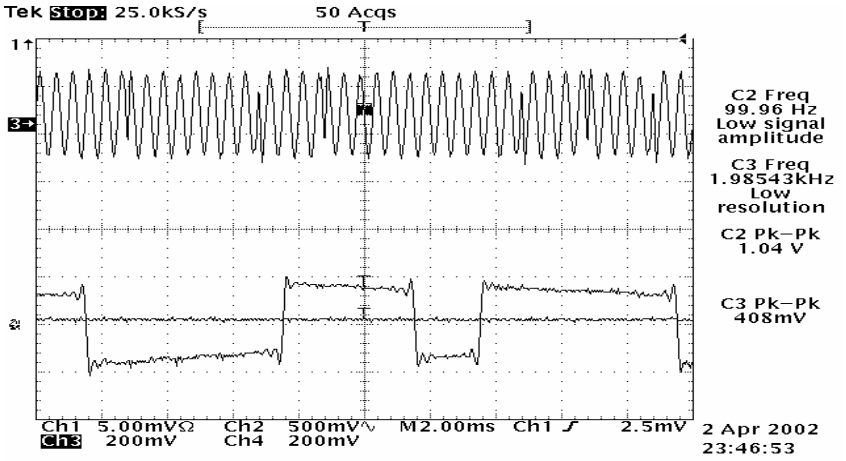
E. A. Costas Loop Demodulation of an externally Generated BPSK Signal
Modify the code you created in the BPSK experiment so that it outputs a BPSK signal on a 2000 Hz carrier sampled at a rate of 8000 Hz. One BPSK symbol should correspond to four complete carrier periods (16 samples per symbol). Implement this BPSK transmitter on one station, station A. Using available BNC cables, connect the output generated by station A to both an oscilloscope and another station, station B.
BPSK signal generation:

```
// Function lfsr
int lfsr(s)
{
   // Local Variables
   int x;                              // Output bit of pn code
   int h;                              // Connection Polynomial
   int  y;                             // Value of s AND h
   int  a;                             // Value of bit 3 of y
   int  b;                             // Value of bit 1 of y

   h = 37;                             // Define connection
polynomial

   // Bitwise AND s and h and set equal to y
   y = s & h;

   // Get values of bit 3 and bit 1 by masking out everything
but the bit you want
   a = (y & 4) >> 2;
   b = y & 1;

   // Bitwise XOR a and b
   x = a ^ b;

   // Return the value of x
   return x;
}

// Interrupt Service Routines
interrupt void rcvISR(void)
{
   int new_sample;
   float input,output;

   new_sample = MCBSP0_DRR;

   input = (float) new_sample;

   if (i< (counter-1))
```

```
    {
      i++;
      output = m_new*input;
    }
  else
    { i = 0;
      // Call lfsr function
       if (i % 45 == 0)
       {
            m = lfsr(s);
            if (m==0)
            {
                 m_new = -1;
            }
            else
            {
                 m_new = m;
            }
            //Create new x for new state vector
            x_new = m << 4;
            //Create new state vector
            s = s >> 1;
            s = s|x_new;
            output = m_new*input;
       }
  }

  MCBSP0_DXR = (int)output
}
```



Demodulation Part:



The code is:

```
/*-------------------------------------------------------
   INTERRUPT SERVICE ROUTINES
-------------------------------------------------------*/
interrupt void rcvISR(void)
{
    int new_sample;

    /* MCBSP0_DRR is the hardware register where received
samples are stored */
   new_sample = MCBSP0_DRR;

   // Updating data array with new sample
   x[oldest] = ((float)new_sample)*4.0e-9;

   // Increment oldest
   oldest++;
   if (oldest==N)
        oldest=0;

   hilbert_output = conv_fcn(h,x,oldest,N);

    if (counter >= grpdelay)
    {// analytic = buffer_x[0] + j*hilbert_output;
    //  phase = cos(phi) +j*(-sin(phi));
    //start processing
    c1 = buffer_x[oldest2]*cos(phi)+ hilbert_output*sin(phi);
    c2 = -buffer_x[oldest2]*sin(phi) + cos(phi)*hilbert_output;
    y = c1;
    if (c1>=0)
       y=1;
    else
       y = -1;
    q = y *c2;
    temp_out1 = temp_pre_out1 + q*beta;
    phi = 2*3.1415926*fc/SAMPLING_RATE + phi +alpha*q +
temp_out1;
    if (phi > 6.283185307)
```

```
        phi = phi-6.283185307;
    temp_pre_out1 = temp_out1;

    buffer_x[oldest2] = ((float)new_sample)*4.0e-9;

   // Increment oldest
   oldest2++;
   if (oldest2==grpdelay)
        oldest2=0;
    //end of processing
    }
   if (counter < grpdelay)
        buffer_x[counter++] = ((float)new_sample)*4.0e-9;

   MCBSP0_DXR = (int) (y*6e8);

}
```

Questions

1. What is the advantage of storing a function into internal program memory? What types of functions make for attractive candidates for storing in internal program memory?

The advantage of storing a function in program memory is if you have code that is very time sensitive. By having the function on program memory, it will be able to be accessed faster and produce the results faster. Only the functions that are most frequently used should be placed in program memory.

2. Qualitatively, how are acquisition time and tracking influenced by the value of the loop bandwidth of a phase lock loop?

Since the loop bandwidth is inversely proportional to acquisition time, as the loop bandwidth increases, the acquisition time decreases but since the tracking (variance of phase estimate) is proportional to the loop bandwidth, a large loop bandwidth means large error. Hence there is a trade-off.

3. How does your measured pull-in range in Part B relate to your designed loop bandwidths, B_L?

Loop bandwidth is the frequency range over the span we could operate the Costas loop correctly. The pull-in range is the maximum change in frequency that the loop can tolerate once the phase is locked without having too much errors. The relationship is that the greater the loop bandwidth, the greater the pull-in range.

Conclusion

The Costas Loop is one type of Phase Lock Loop used for carrier recovery. The Costas Loop tracks the phase change of the incoming signal in order to demodulate the signal. For this experiment, we simulated the Costas Loop in MATLAB for the case of an AM signal and for a BPSK signal. We also implemented the Costas Loop in C on the C6701 for the AM and BPSK signals. Our results were consistent with the required specifications given to us for the experiment.

The performance is dependent on loop parameters Zeta, Alpha, Beta and the loop bandwidth.

References

Nezami, Mohamed K. "DSP-Based Carrier Acquisition andTracking for Burst TDMA Mobile Land and Satellite Receivers" Applied Microwave and Wireless, September 2001

Proakis, John G. **Digital Communications** McGraw Hill 4th edition, 2001.

Proakis. John G, and Dimitris G. Manolakis. **Digital Signal Processing** 3rd edition 1996.

Tretter, Steven A. **Communication System Design Using DSP Algorithms** Plenum Press, 1995.

VALIDATION SHEET

Show your code and your results for the following to the TA and have them initial in the appropriate position.

	Correct Operation	Maximum Frequency Deviation	Cycles
B. AM Demodulation			
C. AM Demodulation			
E. BPSK Demodulation			

Extra Credit

For the QPSK: we generate the QPSK (1.5 0.5 -0.5 1.5) signal by the Binary Bit. Use one binary bit plus one, plus another binary bit time 2, then use the result minus 1.5, then we get the Q bits(-1.5 -0.5 0.5 1.5)
The code for matlab is here:

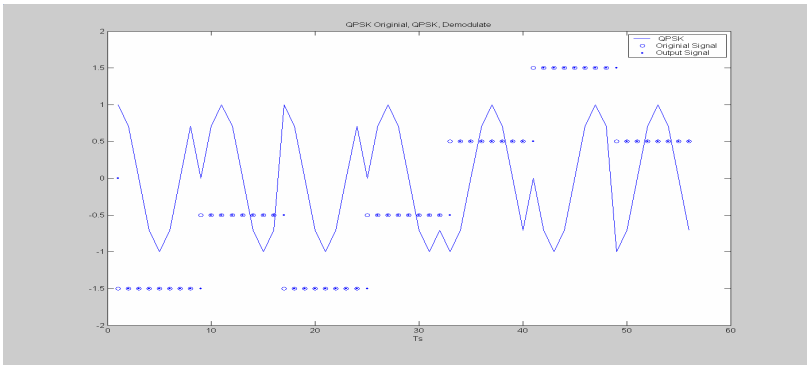
```
x=(ones(1,length(x))+x)/2;
data=[];
for I=1:length(x)
    qpsk_1=2*x(1,I);
    data=[data,qpsk_1];
    qpsk_2=2*x(1,I)+1;
    qpsk_3=qpsk_2-1.5
    data=[data,qpsk_2];
end;
```

for the loop , we use the rule mentioned by the exp5 which jody gave us.

The code is here:

```
for I=1:len-groupdelay
    phi(I)= temp_out3;
    phase(I) = exp(-i*phi(I));
if
    c1(I) = real(analytic(I)*phase(I));
    c2(I) = imag(analytic(I)*phase(I));
    out(I)= sign(c1(I));
    q(I) = sign(c1(I))*c2(I)-c1*sign(c2(I));
    temp_out1=temp_pre_out1+q(I)*beta;
    temp_out2=alpha*q(I)+ temp_out1;
    temp_out3=2*pi*fc/fs+phi(I)+temp_out2;
    temp_pre_out1=temp_out1;

end;
```

the result is here:



Maybe you will feel this QPSK looks ugly and it doesn't like QPSK, the key reason is the sample frequency is lower, I use the sample frequency 16k to replot this get the result is below: