Chapter 10

Communications Applications

Introduction

The great demand for Internet connectivity by consumers and the wireless revolution have pushed the demand for DSP based communication solutions to a very high level. What this chapter attempts to do is work through some of the basic issues in using real-time DSP to handle both analog and digital modulation schemes. The first step in this process is getting analog communication waveforms into and out of the discrete-time domain. Next follows a discussion of some basic signal processing functions, such as a DSP Costas-Loop for coherent carrier recovery.

The environment where the communication signal resides, the size weight and power requirements, and the market where the communication services reside, dictates how to choose a particular implementation. These is no simple one approach satisfies all solution. The dividing line between analog and DSP based portions of a particular implementation is always moving towards DSP, but where this line is placed depends on the above mentioned factors. Once you decide to use DSP, then you must decide which portions, if any are implemented using ASICs and which can be done on a general purpose DSP. By visiting a few vendor Web sites, you soon find that there are a vast array of

wireless communications oriented ASICs just waiting for you.

Summary of Factors Relating to Implementation Choices

How to structure the material presented here in this chapter is thus a challenge in itself. Factors which may be important in a DSP based communication system design include:

The Communications Channel

- Wired bandlimited channel, e.g., wireline/PSTN modems
- RF/Microwave line-of-sight, e.g., satellite, common carrier terrestrial
- Mobile radio
- Fiber optic
- others

Size, Weight, and Power Requirements

- Cellular base station
- Cellular portable
- PC PSTN modem; desktop and portable
- Satellite and satellite earth station
- Terrestrial point-to-point
- Wireless data networks; access node and remote
- others

Market Where Service Resides

- All consumer, e.g., cell phones
- Government services non-military
- Military
- others

DSP Technology

- ASIC
- General purpose DSP
- A combination of the two

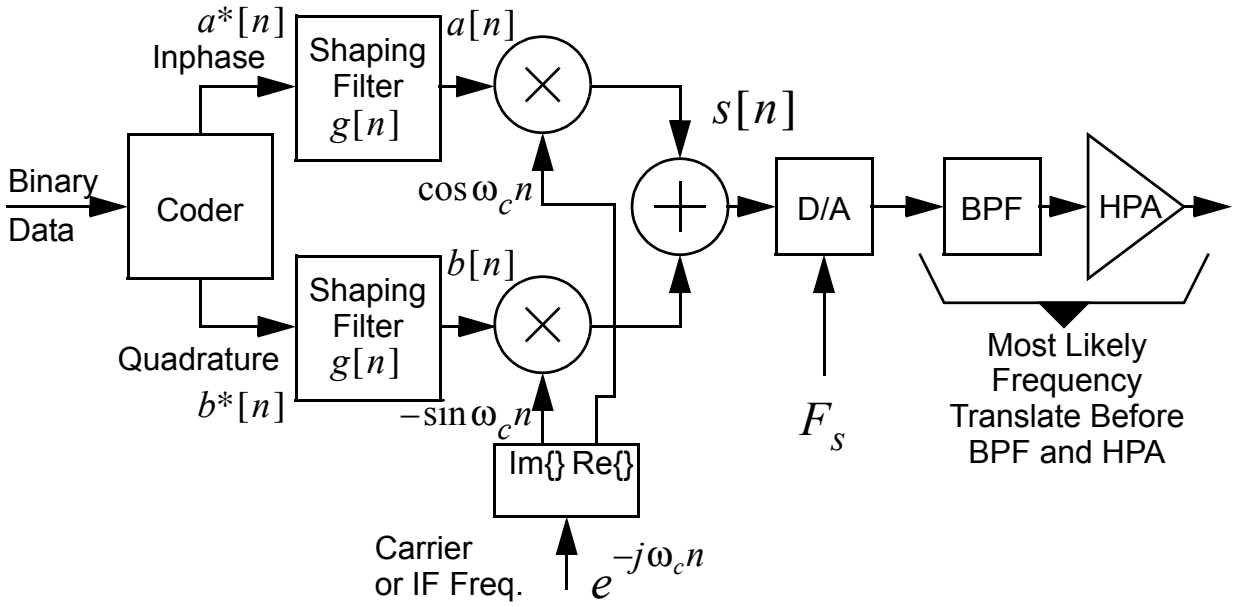
Transmitting Signals

In the book by Tretter[1] DSP techniques for communication system design is discussed, with emphasis on wireline modems. A good blend of theory and practice is presented. From the introductory section we know that with the great variety of communication system scenarios there are more considerations.

Baseband/IF Transmitter

- The most direct approach for using DSP at the transmitter is to simply modulate directly onto a discrete-time carrier and pass the composite signal out through the D/A

1. S. Tretter, *Communication System Design Using DSP Algorithms with Laboratory Experiments for the TMS320C6713™*, Kluwer Academic/Plenum Publishers, 2008.



- The coder output is likely to be an impulse train with the shaping filter, $g[n]$, used to control the baseband transmitted spectral occupancy
- The discrete-time signal prior to the D/A is of the form

$$s[n] = a[n]\cos(\omega_c n) - b[n]\sin(\omega_c n) \tag{10.1}$$

where $a[n] = a^*[n] * g[n]$, $b[n] = b^*[n] * g[n]$,

$$a^*[n] = \sum_{k=-\infty}^{\infty} a_k \delta[n - kN_s] \tag{10.2}$$

$$b^*[n] = \sum_{k=-\infty}^{\infty} b_k \delta[n - kN_s] \tag{10.3}$$

and N_s is the number of samples per symbol

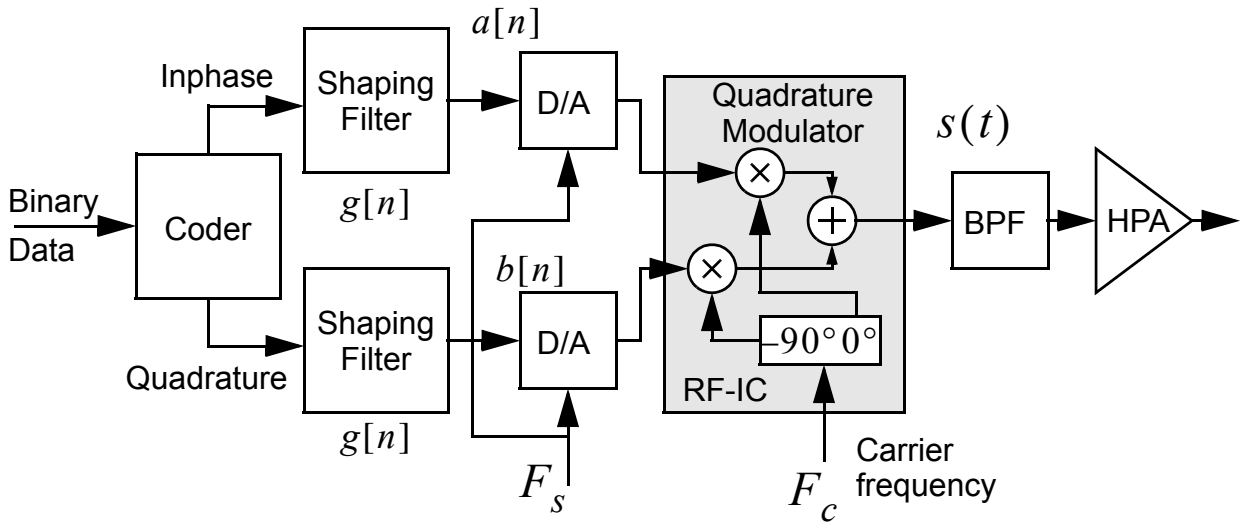
- The points in the 2-dimensional space defined by the point pairs (a_k, b_k), define the *signal constellation*
- In practice these point pairs are viewed in the complex plane as $c_k = a_k + jb_k$
- The coder block groups consecutive blocks of J-bits into J-bit binary words that define a 2^J element alphabet
  - The operation of J-bits being formed into a J-bit word is really nothing more than a serial-to-parallel converter
  - In other applications it might be that $K < J$ input bits form the J-bit word; in this case a rate K/J error correcting code may be involved (block or convolutional)
- The complex envelope or complex baseband signal corresponding to $s[n]$ is

$$\tilde{s}[n] = a[n] + jb[n] \tag{10.4}$$

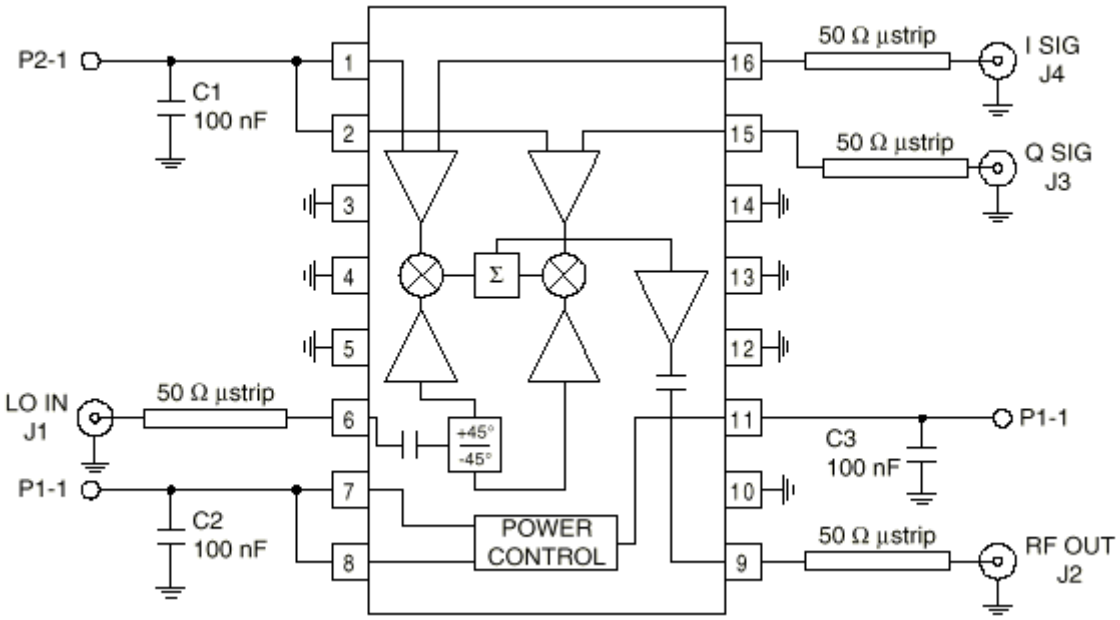
- For a wireline modem application the carrier frequencies are very low and may not need to be translated as depicted in the above figure
- For most other cases the D/A output would be considered at an IF frequency, and would need analog signal processing components to translate the signal to the proper center frequency, followed by high power amplification (HPA)

Complex Baseband Transmitter

- A more complex, yet also more flexible approach, is to have the DSP create the signal in complex baseband (complex envelope) format



- In this implementation the DSP is not responsible for carrier generation, but the AIC must have two D/A output channels
- Below 2.4 GHz the quadrature modulator is typically an RF integrated circuit which contains the mixers, power combiner, and phase splitting circuit for generating $\cos 2\pi f_c t$ and $\sin 2\pi f_c t$
  - As an example consider the *RF MicroDevices* part number RF2422



RF Microdevices RF2422
2.4 GHz Quadrature Modulator

- In this implementation the signals $a[n]$ and $b[n]$ are the impulse modulator outputs convolved with the pulse shaping filter

$$a[n] = \sum_{k=-\infty}^{\infty} a_k g[n - kN_s] \tag{10.5}$$

$$b[n] = \sum_{k=-\infty}^{\infty} b_k g[n - kN_s] \tag{10.6}$$

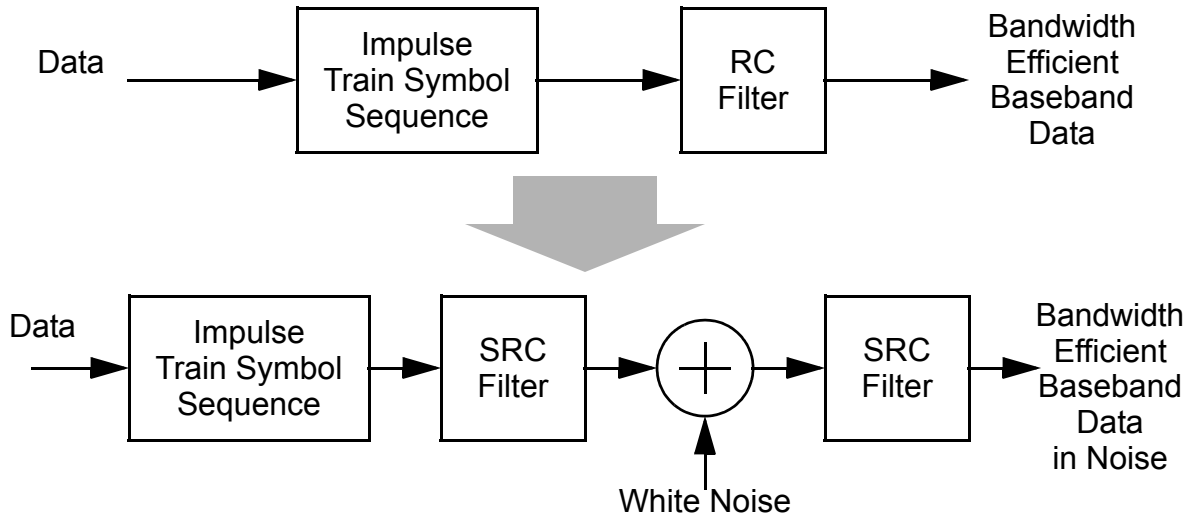
- Following the quadrature modulator we have the analog signal

$$s(t) = a(t)\cos(2\pi F_c t) - b(t)\sin(2\pi F_c t) \tag{10.7}$$

  – The signals $a(t)$ and $b(t)$ are $a[n]$ and $b[n]$ along with the influences of the D/A analog reconstruction filters

Practical Pulse Shaping

- A common form of pulse shaping is one that satisfies the Nyquist criterion for zero intersymbol interference (ISI)
- Zero ISI means that pulses corresponding to adjacent symbols do not interfere with each other at symbol spaced sampling instants
- A popular baseband shaping filter is the raised cosine, which has a parameter $\alpha \in [0, 1]$, known as the *excess bandwidth factor*
- When the channel frequency response is flat across the signal bandwidth and the noise is white (flat spectrum), it is best to equally split the raised cosine (RC) frequency response shape into the product of two square-root raised cosine (SRC) frequency responses



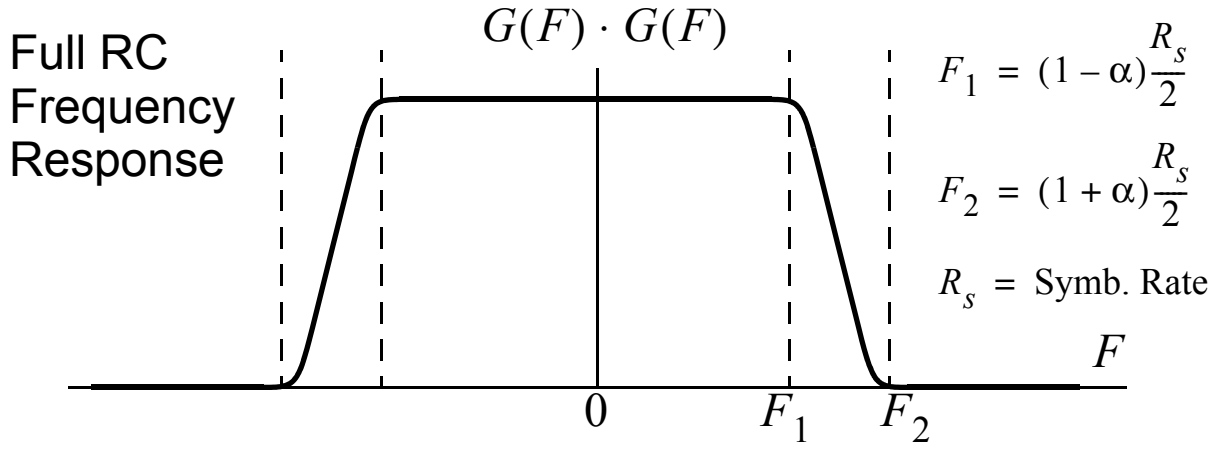
Example: MATLAB Implementation of Shaped BPSK Modulation using an FIR SRC filter

- The SRC impulse response can be implemented as an FIR filter by truncating the true SRC impulse response

$$g(t) = \frac{4\alpha\cos\left[(1+\alpha)\pi\frac{t}{T_s}\right] + \frac{\sin\left[(1-\alpha)\pi\frac{t}{T_s}\right]}{4\alpha t/T_s}}{\pi\sqrt{T_s}[1-16\alpha^2 t^2/T_s^2]} \qquad (10.8)$$

  where T_s is the symbol duration

- The frequency response of the full RC pulse (equivalent to $G(F)G(F)$) is shown in the following figure

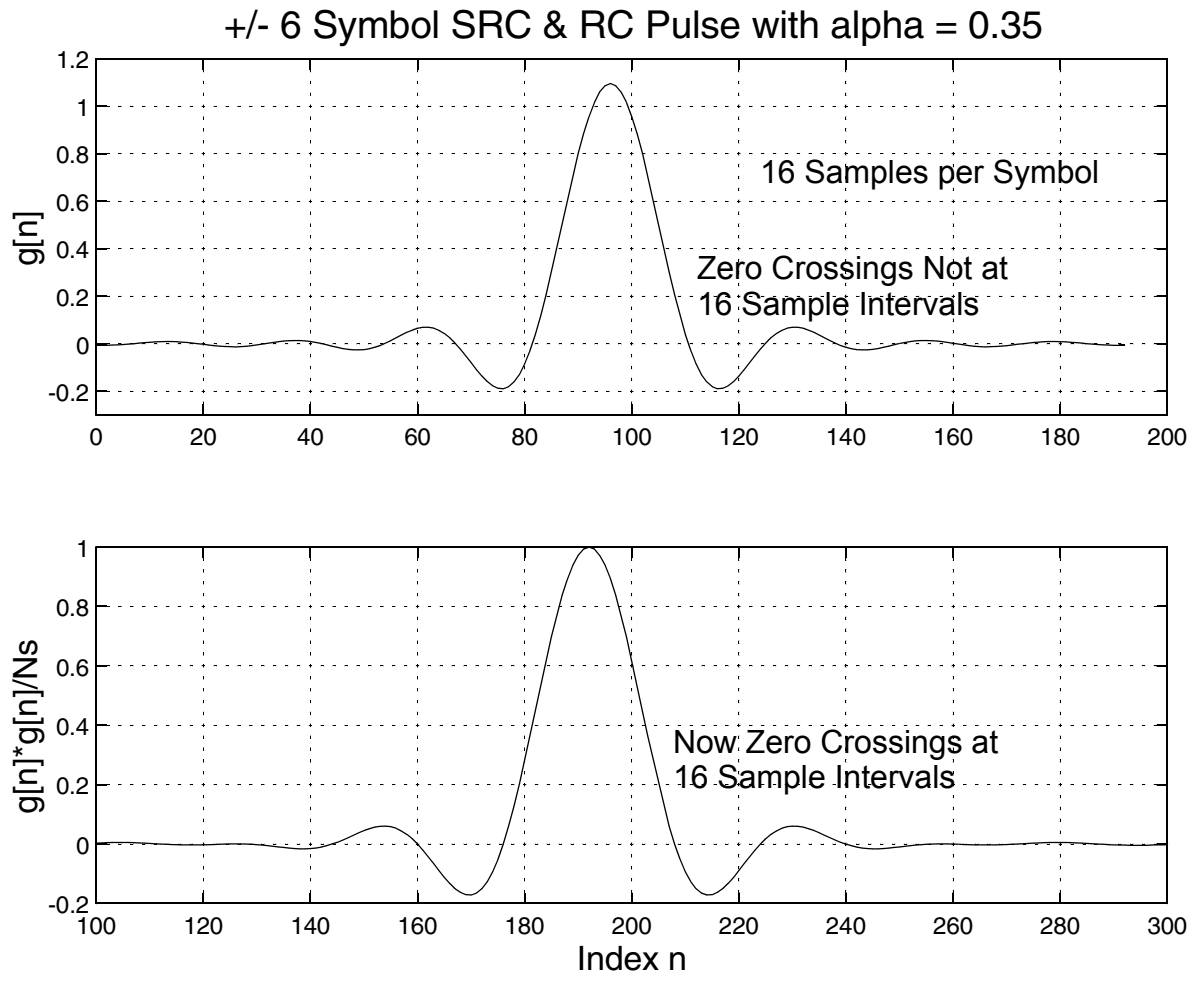


- A popular truncation interval for this impulse response is ±6 symbols, which implies a total impulse response duration of 12 symbol periods at N_s samples per symbol

- A MATLAB function for creating such a pulse shaping filter is the following

```
function b = sqrt_rc_imp(Ns,alpha,M)
%        b = sqrt_rc_imp(Ns,alpha,M)
%   Sqrt-Raised-Cosine Impulse Response Filter
%    Ns = number of samples per symbol
% alpha = excess bandwidth factor = 0.35 for IS 136
%     M = equals sqrt(RC) one-sided symbol truncation factor

% Design the filter
n = -M*Ns:M*Ns;
b = zeros(size(n));
a = alpha;
for i=1:length(n),
   if (1 - 16*a^2*(n(i)/Ns)^2) == 0
      b(i) = 1/2*((1+a)*sin((1+a)*pi/(4*a))- ...
          (1-a)*cos((1-a)*pi/(4*a))+(4*a)/pi*sin((1-a)*pi/(4*a)));
   else
      b(i) = 4*a./(pi*(1 - 16*a^2*(n(i)/Ns).^2));
      b(i) = b(i).*(cos((1+a)*pi*n(i)/Ns) + ...
          sinc((1-a)*n(i)/Ns)*(1-a)*pi/(4*a));
   end %end if statement
end %end for loop
```

- A plot of $g[n]$ at 16 samples per symbol (more typically this would be 4 or so), ± 6 symbol length, and $\alpha = 0.35$ is shown below along with the composite RC pulse $g[n] * g[n]$

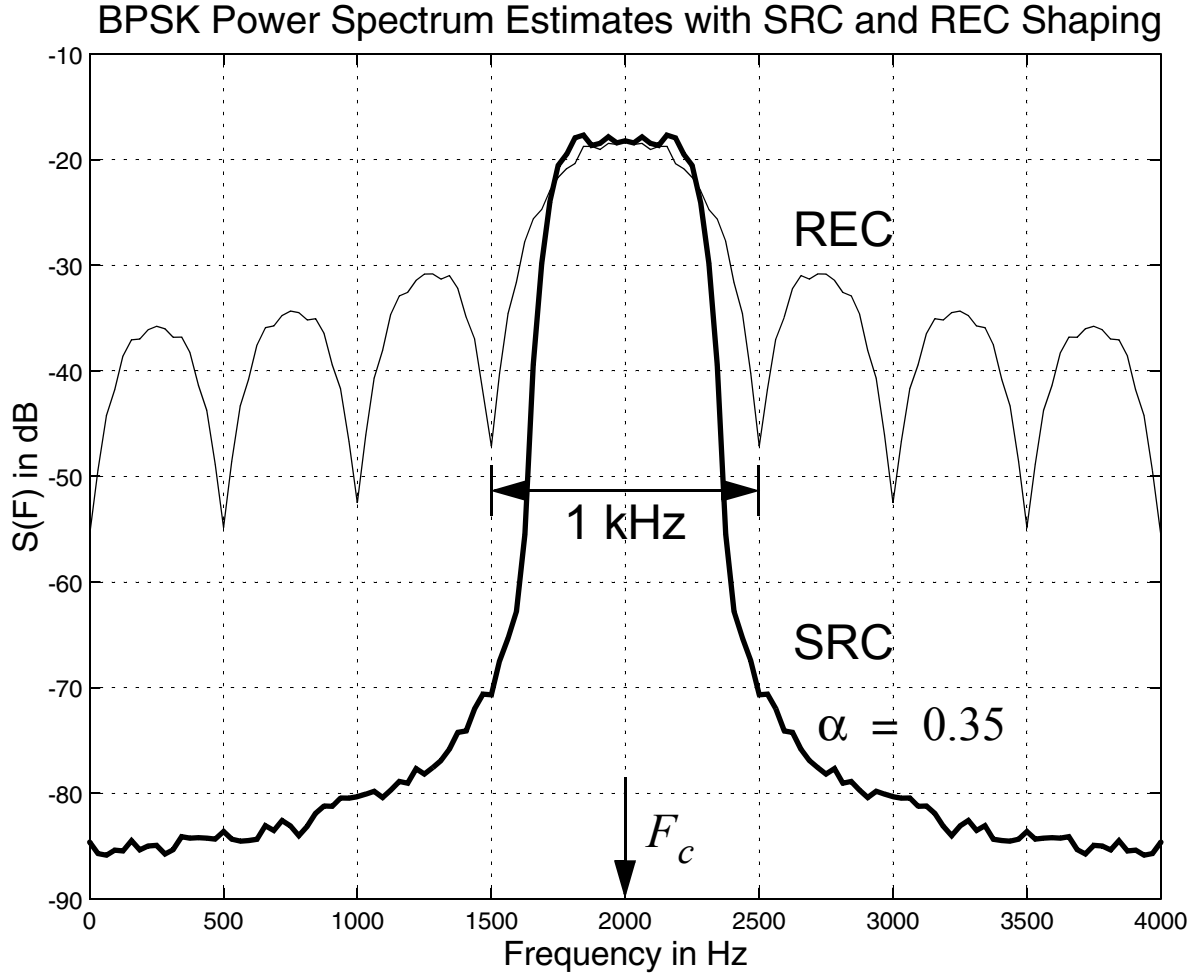


- To investigate further we will now implement a binary phase-shift keyed (BPSK) modulator ($a_k = \pm 1, b_k = 0$) using both rectangular and SRC pulse shaping

```
% A Square-Root Raised Cosine Pulse Shaping
% Demo that Creates a BPSK waveform. src_bpsk.m
%
% Mark Wickert 5/98
%
% Set Fs = 8 kHz
% Set Rb = 500 bits/sec
% Create a record of M = 1000 symbols using m_ary.m
% which creates symbols of amplitude 0 & 1
Fs = 8000;
Rb = 500;
M = 1000;
```

```
d = m_ary(2,M,1);
d = 2*d - 1; % translate to -1/+1 amplitudes
d = [d zeros(M,Fs/Rb-1)]; % Set up for impulse modulation
d = reshape(d',1,M*Fs/Rb);
n = 0:(M*Fs/Rb-1);
% Square-root raised cosine filter and then modulate
% on a cosine carrier of 2000 Hz
b_src = sqrt_rc_imp(Fs/Rb,0.35,6);
df = filter(b_src*Rb/Fs,1,d); %Filter with unity DC gain
Fc = 2000;
x = df.*cos(2*pi*Fc/Fs*n); %SRC Shaped BPSK
% Rectangle Pulse Shape filter and then modulate
% on a cosine carrier of 2000 Hz
b_rec = ones(1,Fs/Rb)/(Fs/Rb); %Make unity gain rec
dff = filter(b_rec,1,d);
y = dff.*cos(2*pi*Fc/Fs*n); %Rectangle Pulse BPSK

function [data,y] = m_ary(M, K, Ns)
% M_ary   [data,y] = m_ary(M, K, Ns): Create an M-level, e.g.,
M=2,4,8,16,
%        data sequence with Ns samples per symbol and containing K
%         total symbols; Ns * K total samples.
%        data is a vector of symbols taking on values from 0 to M-1
%         with Ns samples per symbol
%         y is a vector of symbol values at one sample per symbol

% create the data sequence
x = rand(K,1);
y = round((M*x)-0.5);

% create a zero padded (interpolated by Ns) symbol sequence
symb = [y zeros(K,Ns-1)]';
symb = reshape(symb,Ns*K,1);

% filter symb with a moving average filter of length Ns to fill-in
% the zero samples
data = filter(ones(1,Ns),1,symb);
```

- Compare the power spectral densities of the two pulse shaping schemes

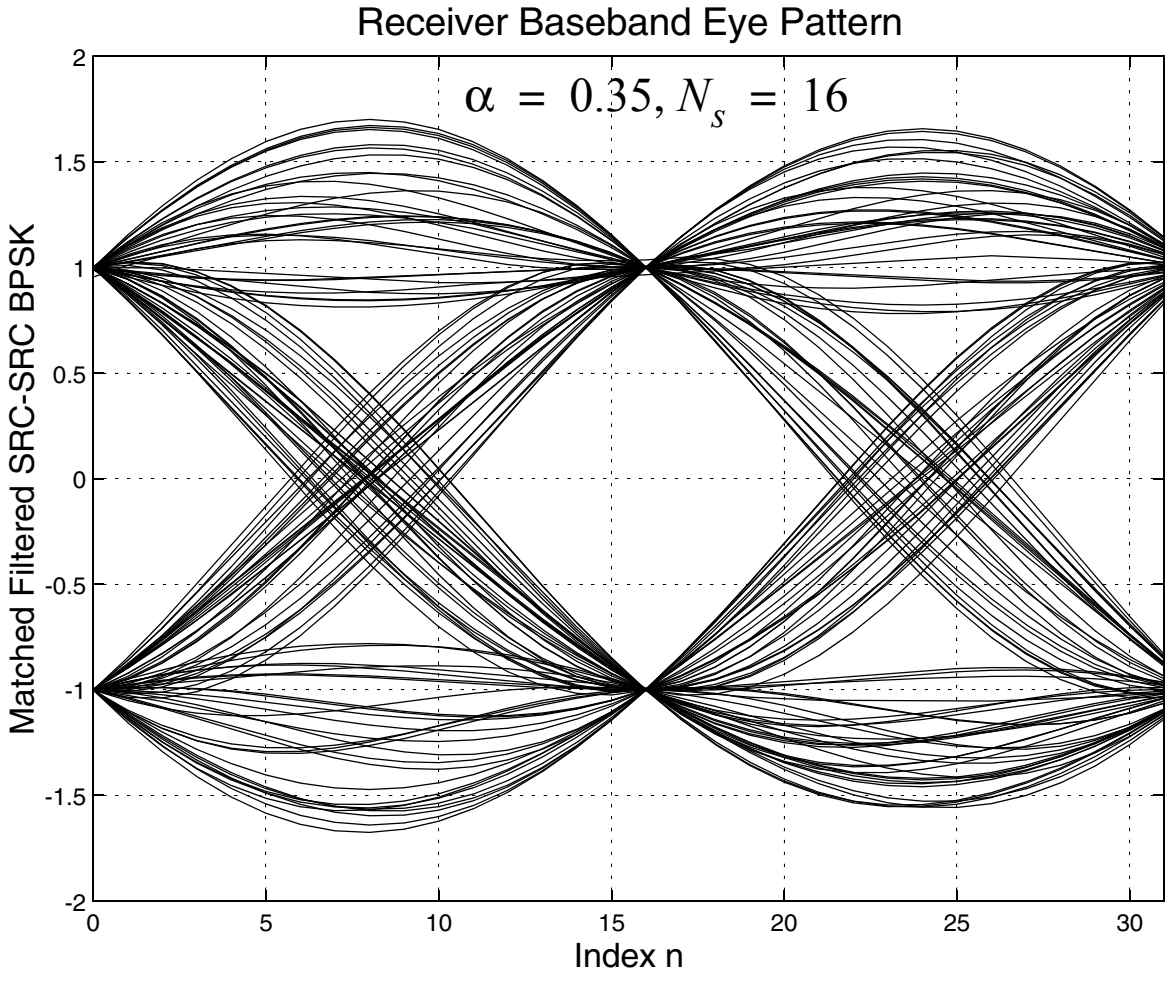


- The eye pattern of the baseband shaped impulse train modulation, following a second SRC or matched filter, is shown below as a result of using the MATLAB function `eyeplot()`

```
function eyeplot(data, W, OFS)
% eyeplot eyeplot(data, W, offset): Plot the eye pattern of a
% digital communication waveform using a window length of W
% and offset OFS

% define some variables
x = 0:W-1;
```

```
L = length(data);
% set axis limits
Limit = max([abs(min(data)) abs(max(data))]);
axis([0,W,-Limit,Limit]);
while L >= W+OFS,
        temp = data(1+OFS:W+OFS);
        plot(x,temp);
        hold on
        data = data(W+1:L);
        L = length(data);
end
hold off
```

```
» eyeplot(filter(b_src,1,df(1000:5000)),32,105)
```

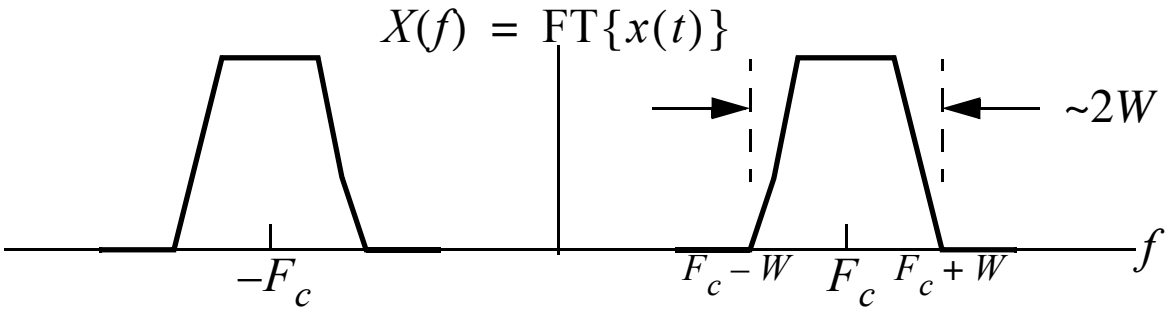


Receiving Signals

Signal processing at the receiver is typically more complex than at the transmitter. There are a lot of details involved. Only the main points will be considered here, or at least for now. In addition to the wireline modem design details given in Tretter, a very good source for more general information on DSP implementation of digital communication receivers is Meyr[1].

Complex Envelope Representation

- A communication signal such as $x(t)$, wireless or wired, is very often in the form of a bandpass signal



- The *complex envelope* representation of a bandpass signal can be written as

$$x(t) = \mathrm{Re}[\tilde{x}(t)e^{j\Omega_c t}] \tag{10.9}$$

where $\tilde{x}(t)$ is the complex envelope and $\Omega_c = 2\pi F_c$ is the carrier frequency in rad/s

1. H. Meyr, M. Moeneclaey, and S. Fetchel, *Digital Communication Receivers*, John Wiley, New York, 1998.

- The signal $\tilde{x}(t)$ is a complex baseband representation of $x(t)$
- We can easily expand (10.9) in rectangular form as

$$x(t) = x_I(t)\cos(\Omega_c t) - x_Q(t)\sin(\Omega_c t) \tag{10.10}$$

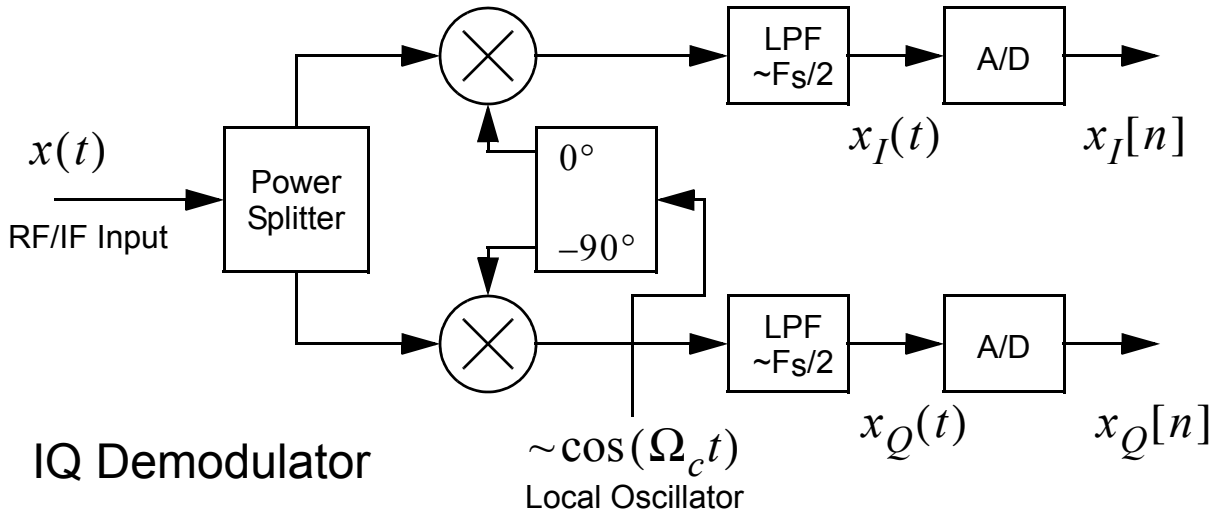
from which it follows that

$$\tilde{x}(t) = x_I(t) + jx_Q(t) \tag{10.11}$$

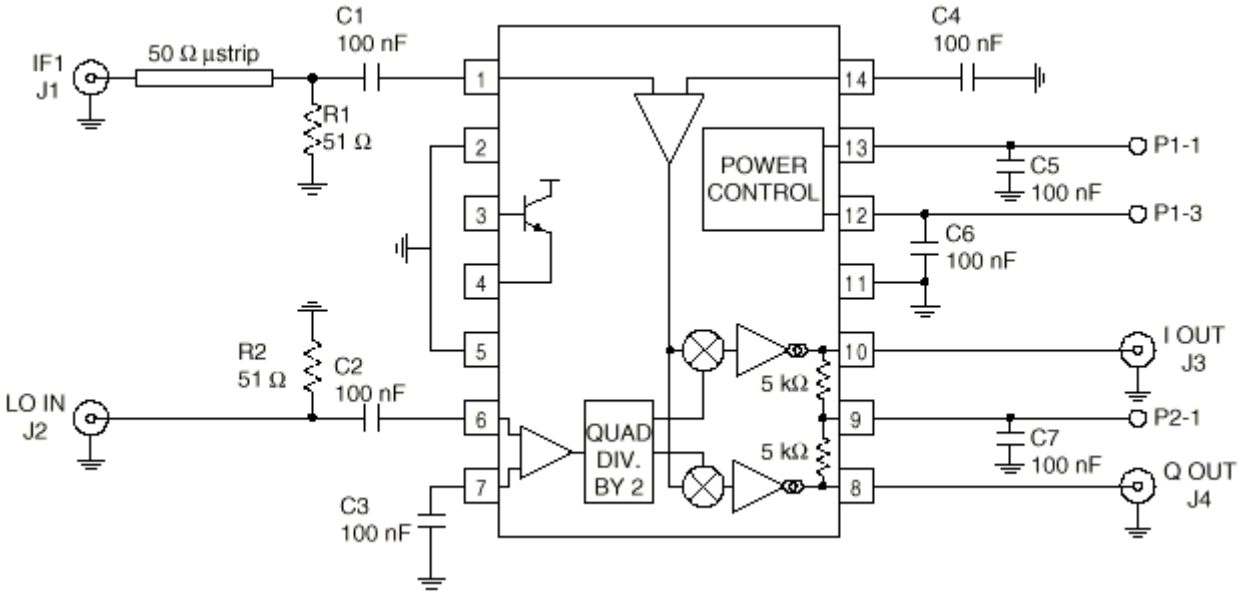
– Note: $x_I(t)$ and $x_Q(t)$ are referred to as the inphase and quadrature parts of the complex baseband signal, respectively

Standard I-Q Demodulation

- Given two channels of analog input on the DSP, a popular approach in receiver design is to convert the modulated RF or IF carrier to complex baseband form and then A/D convert $x_I(t)$ and $x_Q(t)$ into $x_I[n]$ and $x_Q[n]$ respectively



– As an example consider the RF MicroDevices part number RF2711



RF Microdevices RF2711 Quadrature Demodulator

- In the IQ demodulator the local oscillator does not have to remove the carrier completely
- Once in the discrete-time domain a carrier tracking loop can be used to remove a small frequency offset and track phase errors
- For wireline modems, with the carrier frequencies being very low, it is possible to A/D convert the entire modulated carrier signal and then remove the carrier in the discrete-time domain

Using the Hilbert Transform

- The complex envelope can also be obtained by using the Hilbert transform, which is defined by

$$\hat{x}(t) = x(t) * \frac{1}{\pi t} = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{x(\tau)}{t-\tau} d\tau \qquad (10.12)$$

- In theory the Hilbert transform (HT) of a signal is obtained by passing it through a filter with impulse response

$$h(t) = \frac{1}{\pi t} \tag{10.13}$$

and frequency response

$$H(\Omega) = -j\,\mathrm{sign}\Omega \tag{10.14}$$

where

$$\mathrm{sign}\Omega = \begin{cases} 1, & \Omega > 0 \\ 0, & \Omega = 0 \\ -1, & \Omega < 0 \end{cases} \tag{10.15}$$

- Simply put, an ideal Hilbert transforming filter is a $90°$ phase shifter

$$\tilde{X}(\Omega) = H(\Omega)X(\Omega) = (-j\,\mathrm{sign}\Omega)X(\Omega) \tag{10.16}$$

- A practical Hilbert transforming filter can be designed in the discrete-time domain as an FIR filter
  - MATLAB will be used to carry out this design shortly
- Simple HT relationships are:

$$\mathrm{HT}\{\cos\Omega_c t\} = \sin\Omega_c t \tag{10.17}$$

$$\mathrm{HT}\{\sin\Omega_c t\} = -\cos\Omega_c t \tag{10.18}$$

- A useful theorem for bandpass signals is that if $m(t)$ is low-pass with $M(\omega) \cong 0$ for $f > W_1$ and $c(t)$ is highpass with $c(t) \cong 0$ for $f < W_2$ and $W_1 < W_2$, then

$$\mathrm{HT}\{m(t)c(t)\} = m(t)\hat{c}(t) \tag{10.19}$$

– A practical example is

$$\mathrm{HT}\{m(t)\cos\Omega_c t\} = m(t)\sin\Omega_c t,\ F_c > W_1$$

- The signal

$$x_+(t) = x(t) + j\hat{x}(t) \tag{10.20}$$

is known as the *analytic signal* or *pre-envelope* associated with $x(t)$

- The Fourier transform of $x_+(t)$ is

$$X_+(\Omega) = 2X(\Omega)u(\Omega) \tag{10.21}$$

where $u(\Omega)$ is the usual step function, except now in the frequency domain

- Referring back to the complex envelope in (10.9), we see that

$$\tilde{x}(t) = x_+(t)e^{-j\Omega_c t} = [x(t) + j\hat{x}(t)]e^{-j\Omega_c t} \tag{10.22}$$

and

$$x_I(t) = x(t)\cos(\Omega_c t) + \hat{x}(t)\sin(\Omega_c t) \tag{10.23}$$

$$x_Q(t) = -j[x(t)\sin(\Omega_c t) - \hat{x}(t)\cos(\Omega_c t)] \tag{10.24}$$

Example: A Simple MATLAB Simulation

- In this first example we generate a simple amplitude modulated carrier

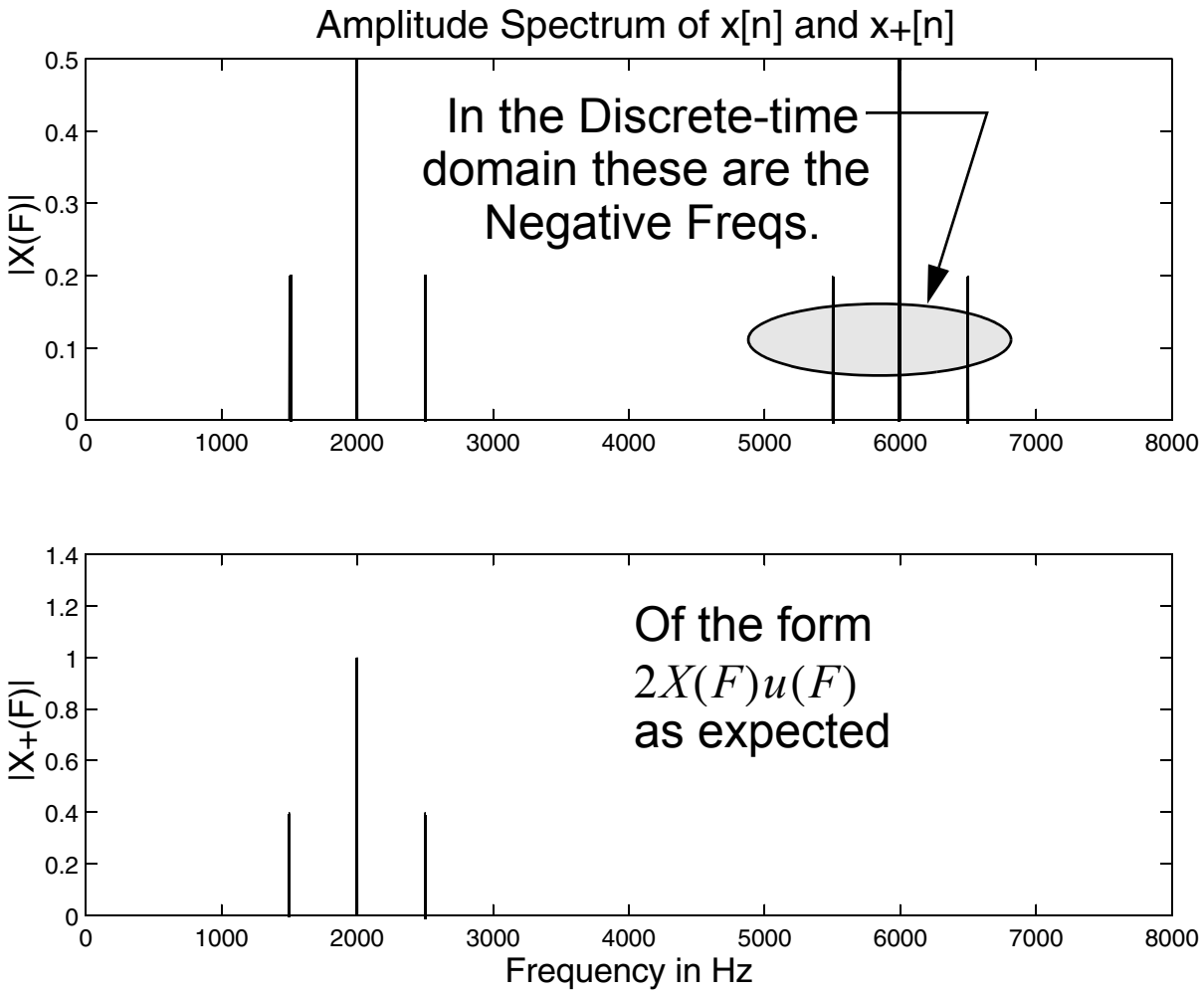
$$x[n] = \left[1 + 0.8\cos\left(2\pi\frac{500}{8000}n\right)\right]\cos\left(2\pi\frac{2000}{8000}n\right) \tag{10.25}$$

with modulation index 0.8, $F_m = 500\,\text{Hz}$, $F_c = 2000\,\text{Hz}$, and a sampling rate of $F_s = 8000\,\text{Hz}$

```
% A Simple Demo Using the Hilbert Transform
% on an AM waveform. hilbert_am.m
%
% Mark Wickert 5/98
%
% Set Fs = 8 kHz
% Set Fc = 2 kHz
% Set Fm = 500 Hz bits/sec
% Create a record of M = 2048 samples
Fs = 8000;
Fc = 2000;
Fm = 500;
M = 2048;
n = 0:M-1;
x = (1+ 0.8*cos(2*pi*Fm/Fs*n)).*cos(2*pi*Fc/Fs*n);
```

- Using the FFT we can plot the amplitude spectrum of $x[n]$ and the corresponding analytic signal $x_+[n]$
- MATLAB makes it very easy to obtain the analytic signal using the function `hilbert()`, which returns an analytic sequence $x_+[n] = x[n] + j\hat{x}[n]$

```
» subplot(211);
» plot(n*Fs/M,abs(fft(x))/M)
» subplot(212);
» plot(n*Fs/M,abs(fft(hilbert(x)))/M)
```



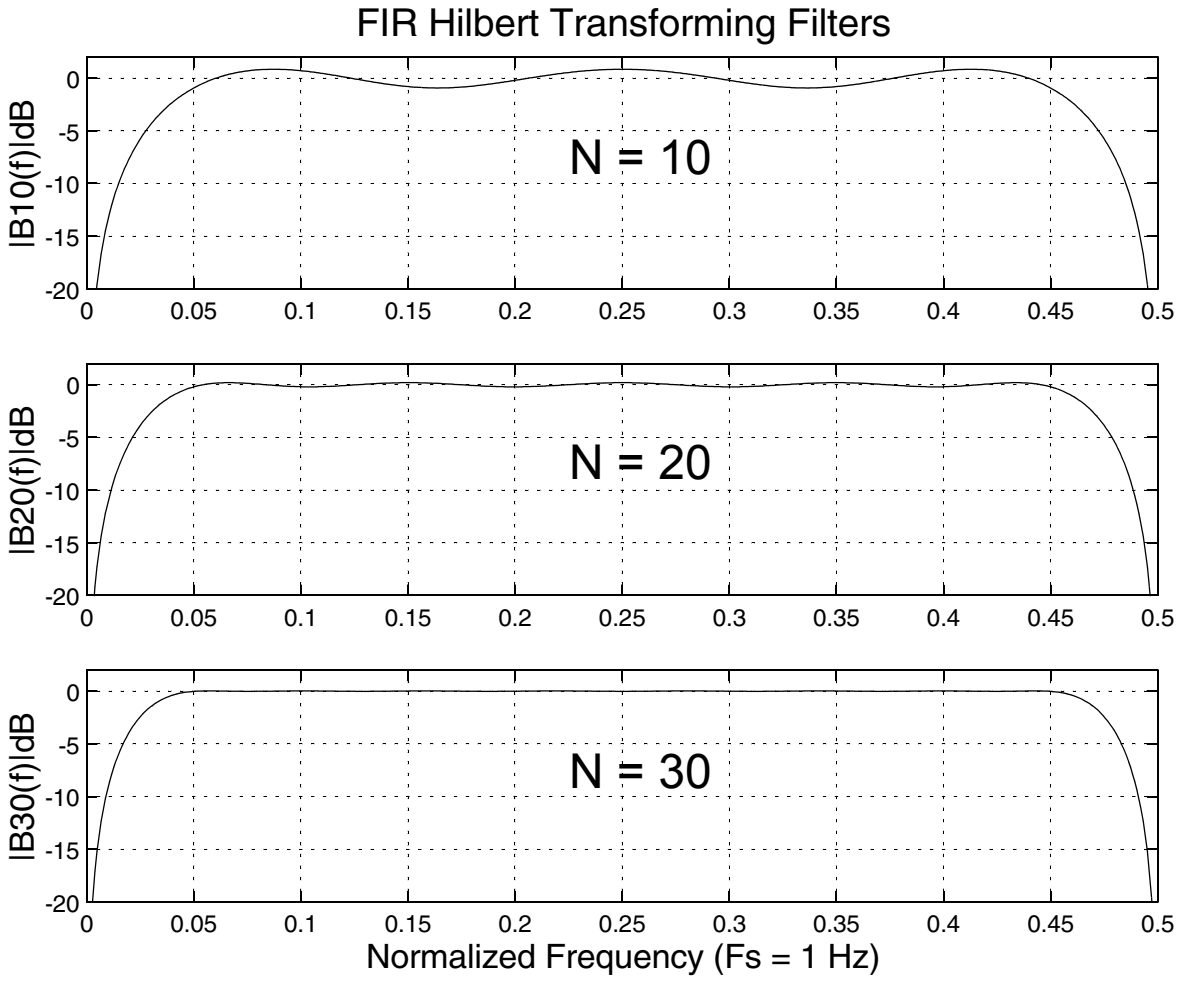
- To obtain an FIR based HT filter we can use the MATLAB function `remez()`, e.g.,

```
» b=remez(N,[.1 .9],[1 1],'Hilbert')
```

  which designs an order N filter ($N + 1$ taps) that has approximately unity gain for $0.1F_s/2 \le F \le 0.9F_s/2$

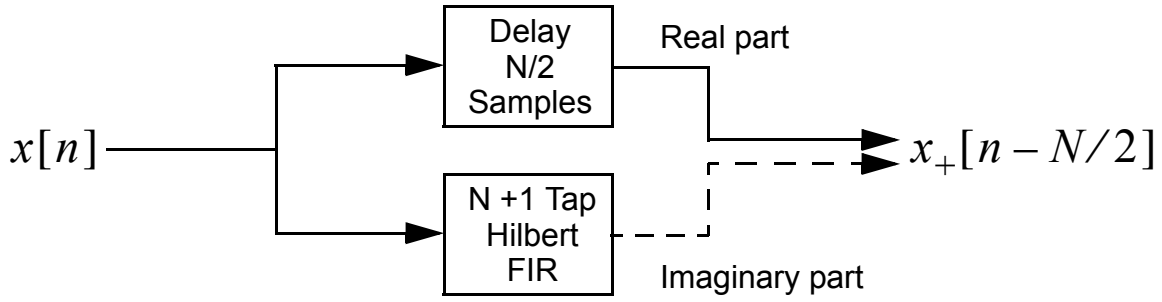
- The gain flatness for three different filter orders is considered below:

```
» b10=remez(10,[.1 .9],[1 1],'Hilbert');
» b20=remez(20,[.1 .9],[1 1],'Hilbert');
» b30=remez(30,[.1 .9],[1 1],'Hilbert');
```



  - Real-time implementation of the above is easy

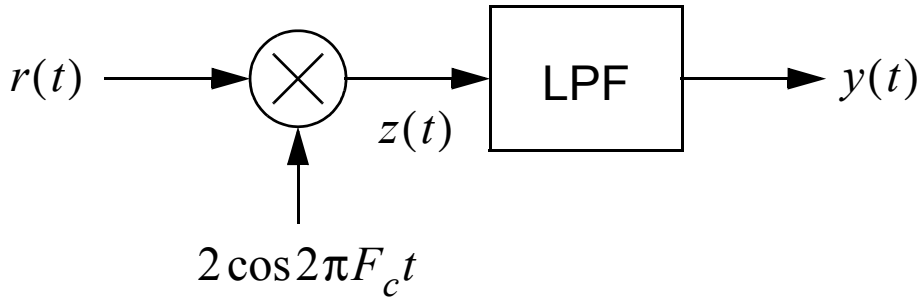
- To properly create an analytic signal using an HT FIR filter also requires that the real part be delayed by $N/2$ (assume N is even, number of taps is odd)



- Once we have the analytic signal we can obtain the complex baseband signal via (10.22)

System Application: A DSP Based Costas Loop for Coherent Carrier Recovery

- In the BPSK example given earlier, the receiver requires knowledge of the carrier frequency and phase to perform coherent demodulation
- Ideally we have:



- Assuming no noise is present

$$r(t) = a(t)\cos 2\pi F_c t \tag{10.26}$$

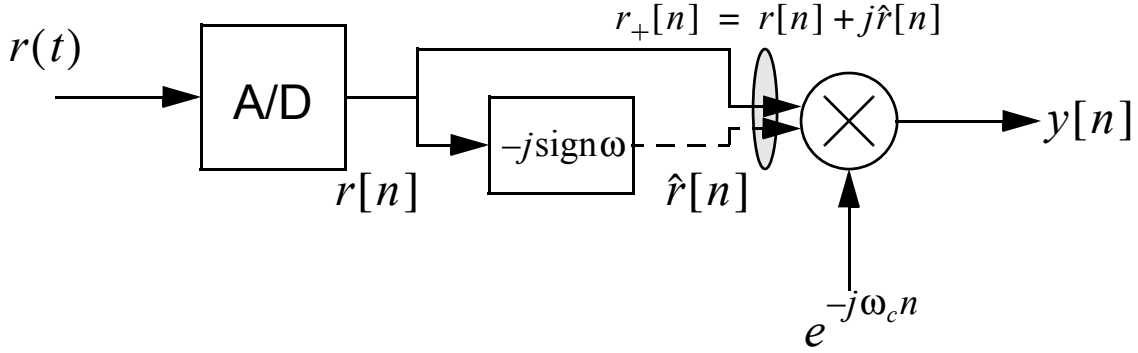
and

$$\begin{aligned} z(t) &= 2a(t)\cos^2(2\pi F_c t) \\ &= a(t) + a(t)\cos[2\pi(2F_c)t] \end{aligned} \tag{10.27}$$

- Following the lowpass filter the $\cos[2\pi(2F_c)t]$ term is removed leaving

$$y(t) = a(t) \tag{10.28}$$

- Another approach is to first form the analytic signal and then multiply by the complex exponential $\exp(-j2\pi F_c t)$
- If this was being done in the discrete-time domain it would be as follows



- From (10.19)

$$\begin{aligned} r_+[n] &= a[n]\cos(\omega_c n) + ja[n]\sin(\omega_c n) \\ &= a[n]e^{j\omega_c n} \end{aligned} \tag{10.29}$$

so

$$y[n] = r_+[n]e^{-j\omega_c n} = a[n] \tag{10.30}$$

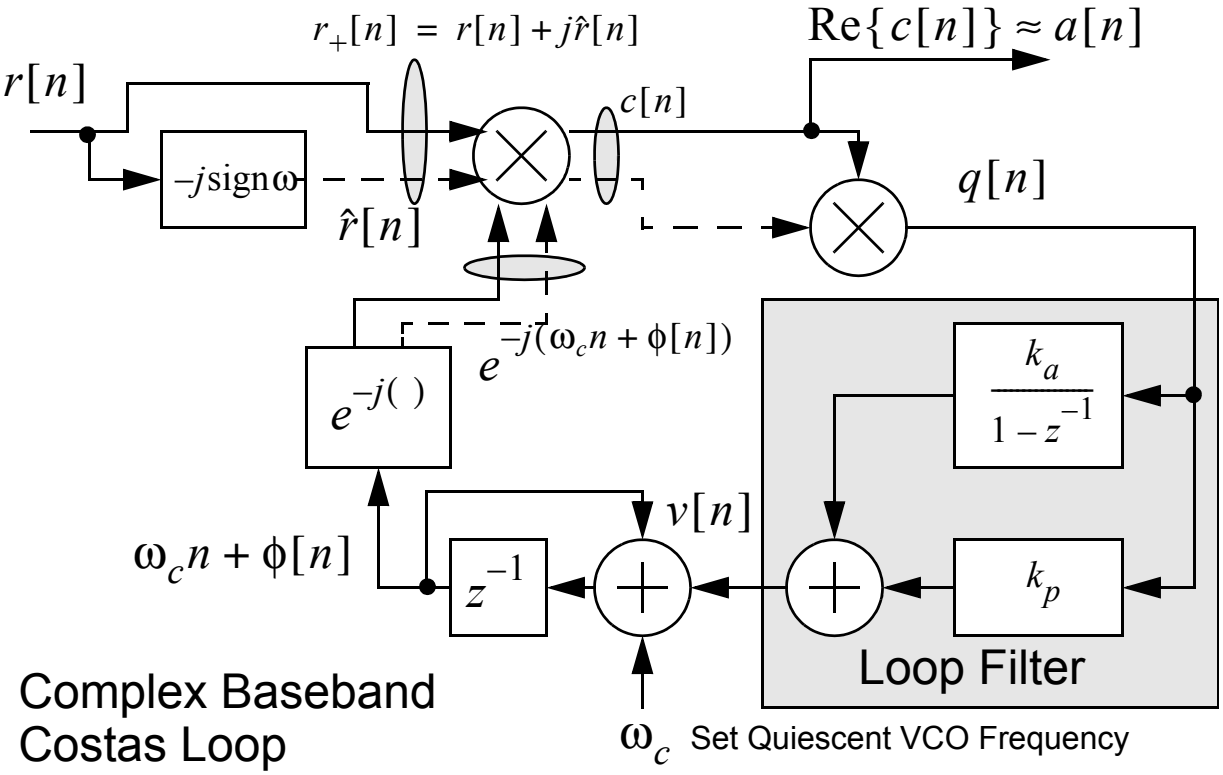
- A more practical model for $r[n]$ is

$$r[n] = a[n]\cos(\omega_c n + \theta[n]) \tag{10.31}$$

where ω_c is still the nominal carrier frequency (in the DSP domain), and $\theta[n]$ is a constant or slowly varying phase

– Note: $\theta[n]$ may also include a frequency error, e.g., $\theta[n] = \Delta\omega n = 2\pi\Delta F/F_s \times n$

- A feedback control system for tracking $\theta[n]$ is the Costas Loop (shown here as a DSP based implementation)



- To understand the operation of the Costas carrier tracking loop we begin by writing out $r_+[n]$ and then $c[n]$

$$r_+[n] = a[n]e^{j(\omega_c n + \theta[n])} \tag{10.32}$$

$$\begin{aligned} c[n] &= r_+[n]e^{-j(\omega_c n + \phi[n])} \\ &= a[n]e^{j(\theta[n] - \phi[n])} \end{aligned} \tag{10.33}$$

- The demodulated data signal is the real part of $c[n]$

$$\mathrm{Re}\{c[n]\} = a[n]\cos(\theta[n] - \phi[n]) \approx a[n] \tag{10.34}$$

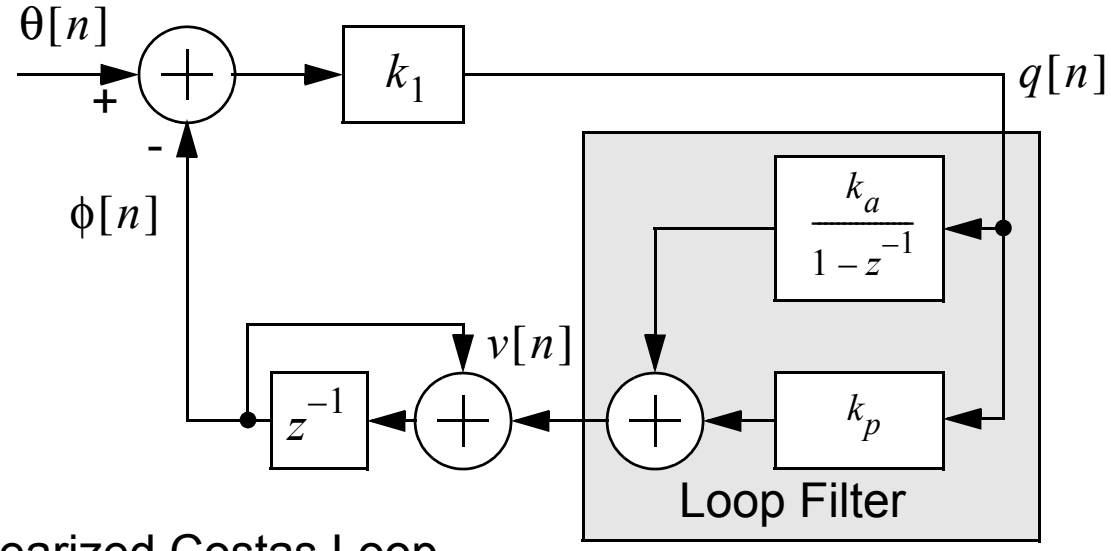
for $\theta[n] - \phi[n] \approx 0$

- The control signal, $q[n]$, for the Costas loop is the product of the real and imaginary parts of $c[n]$

$$\begin{aligned} q[n] &= a^2[n]\cos(\theta[n]-\phi[n])\sin(\theta[n]-\phi[n]) \\ &= 0.5a^2[n]\sin\{2(\theta[n]-\phi[n])\} \end{aligned} \tag{10.35}$$

- For a small phase error, $|\theta[n]-\phi[n]| \ll 1$, we can linearize the control loop since

$$q[n] \cong a^2[n](\theta[n]-\phi[n]) \tag{10.36}$$



Linearized Costas Loop

  - In the linearized model, the lowpass filtering action of the closed-loop response allows us to replace $a^2[n]$ with $k_1 = E\{a^2[n]\}$, which for BPSK is just one

- The closed-loop system performance can be obtained from the system function

$$H(z) = \frac{\Phi(z)}{\Theta(z)} \tag{10.37}$$

- The loop filter chosen here is second-order and is presently in the form of an accumulator (integrator) in parallel with a gain or proportional block

$$\frac{V(z)}{Q(z)} = k_p + \frac{k_a}{1-z^{-1}} = (k_p + k_a)\frac{\left(1 - \frac{k_p}{k_p + k_a}z^{-1}\right)}{1-z^{-1}} \qquad (10.38)$$

- The equivalent to a voltage controlled oscillator (VCO) is the accumulator with input $v[n]$ and output $\phi[n]$

$$\frac{\Phi(z)}{V(z)} = \frac{z^{-1}}{1-z^{-1}} \qquad (10.39)$$

- Closing the loop we have

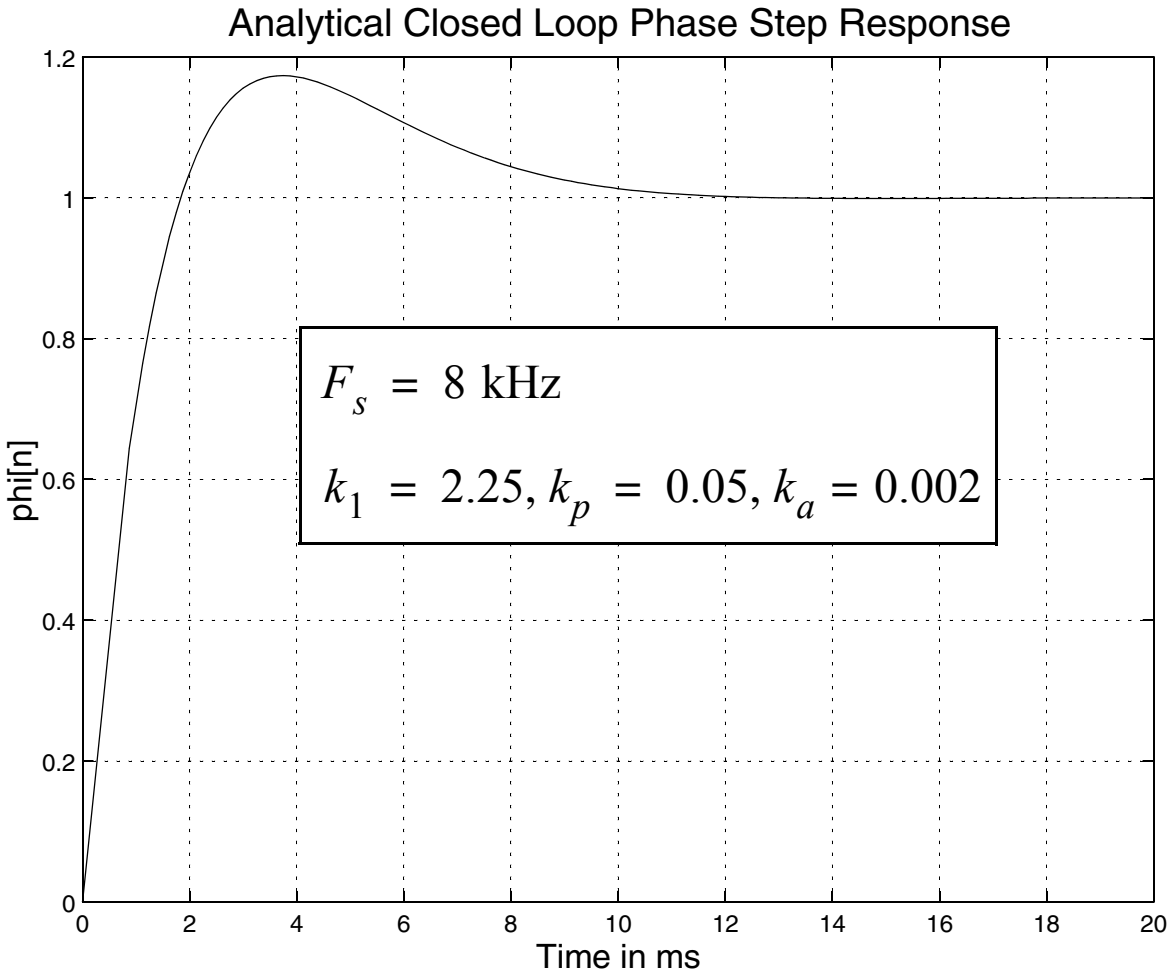
$$\Phi(z) = [\Theta(z) - \Phi(z)]k_1(k_p + k_a)z^{-1}\frac{\left(1 - \frac{k_p}{k_p + k_a}z^{-1}\right)}{(1-z^{-1})^2} \qquad (10.40)$$

or finally,

$$H(z) = \frac{k_1(k_p + k_a)z^{-1}\left(1 - \frac{k_p}{k_p + k_a}z^{-1}\right)}{(1-z^{-1})^2 + k_1(k_p + k_a)z^{-1}\left(1 - \frac{k_p}{k_p + k_a}z^{-1}\right)} \qquad (10.41)$$

$$= \frac{k_1(k_p + k_a)z^{-1} - k_1k_pz^{-2}}{1 - [2 - k_1(k_p + k_a)]z^{-1} + (1 - k_1k_p)z^{-2}}$$

- Assuming a sampling frequency of 8 kHz the analytical closed loop response step response is obtained using MATLAB



MATLAB Simulation

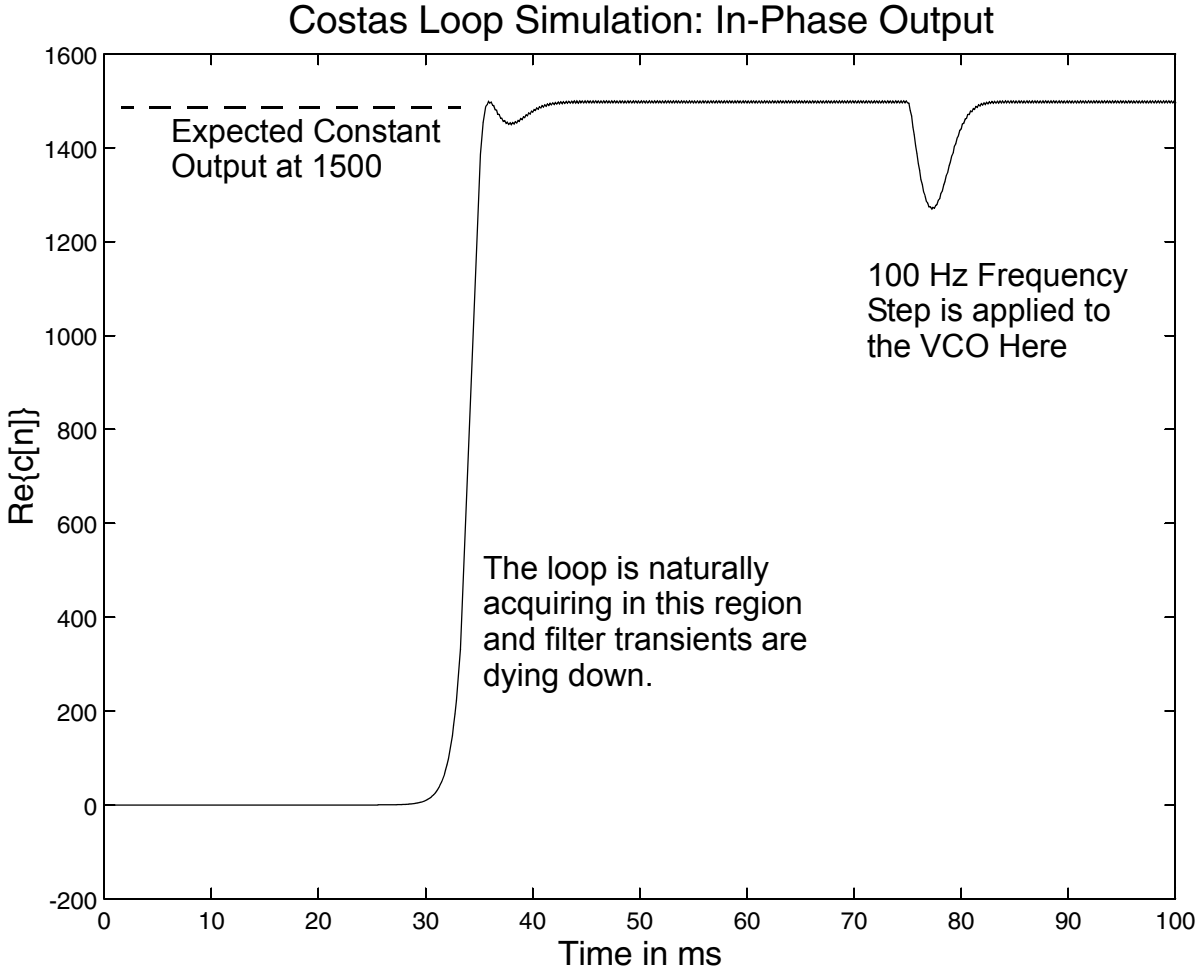
- A complete simulation of the loop using an $M = 30$ Hilbert transforming FIR filter is constructed to verify proper loop performance with simulated signals
- For the simulation an unmodulated carrier at 2000 Hz with amplitude 1500 is input to the system
  - Note: 1500 is a representative amplitude for digitized sinu-

soids once inside the DSK

```
% Costas Loop Simulation
%
% Mark Wickert 5/11/98

Fs = 8000;   % Sampling Frequency
F0 = 2000;   % Carrier frequency
N = 1000;    % Number of simulation points
kp = 0.05;   % Loop filter proportional gain
ka = 0.002;  % Loop filter accumulator gain

b=remez(30,[.1 .9],[1 1],'Hilbert'); %Hilbert 31-tap FIR

n = 0:N;
%Generate carrier at F0 with amplitude similar to DSK
%x = 1500*cos(2*pi*50/Fs*n).*cos(2*pi*F0/Fs*n);%with/DSB Mod.
x = 1500*cos(2*pi*F0/Fs*n); %Unmodulated carrier
%Generate analytic signal
xa = filter([zeros(1,15),1],1,x) + j*filter(b,1,x);

q = 0; q_old = 0; v = 0; v_old = 0;
accum = 0; %NCO accumulator
out = zeros(1,N+1) + j*zeros(1,N+1);
out_i = zeros(1,N+1);
q = zeros(1,N+1);
v = zeros(1,N+1);
%Begin simulation loop to model feedback system
for i=1:N+1
   out(i) = xa(i)*exp(-j*accum);
   out_i(i) = real(out(i));
   q(i) = real(out(i))*imag(out(i));
   q(i) = q(i)*1e-6;
   %Loop filter
   v(i) = v_old + (kp+ka)*q(i) - kp*q_old;
   %Update filter states
   v_old = v(i); q_old = q(i);
   %Apply a frequency step at 600 samples &
   %Wrap the phase in accumulator
   if i <= 600
      accum = mod(accum + 2*pi*F0/Fs + v(i),2*pi);
```

```
   else
      accum = mod(accum + 2*pi*(F0-100)/Fs + v(i),2*pi);
   end
end %Simulation loop
```

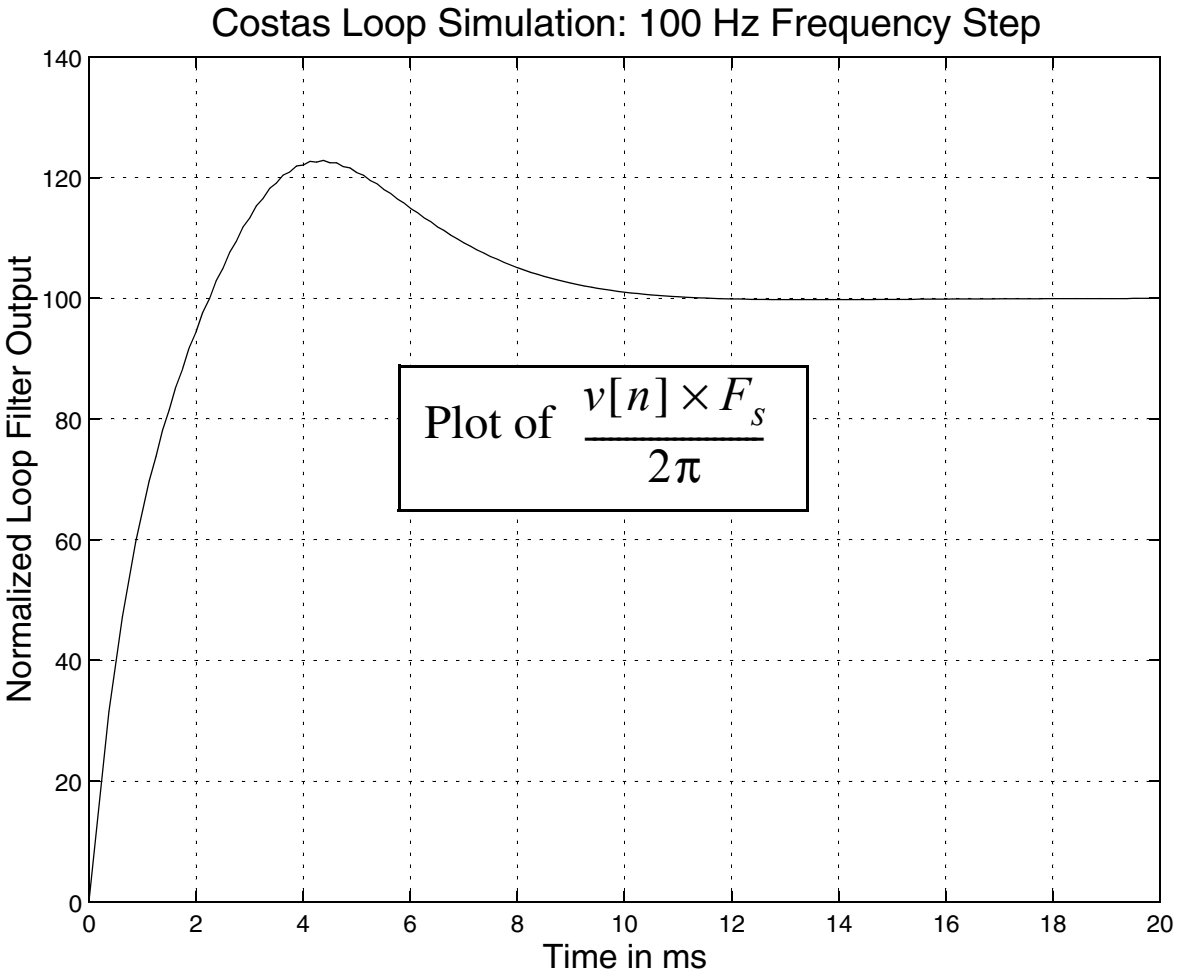
- A nominal sampling rate of 8 kHz is assumed
- A VCO quiescent frequency of 2000 Hz is used to allow the loop to naturally acquire phase lock
- At 600 samples into the simulation the VCO quiescent frequency is shifted down by 100 Hz
- The recovered modulation, in this case a constant is shown below



- The loop control signal $v[n]$ response with a corresponding

positive step to drive the loop back into lock

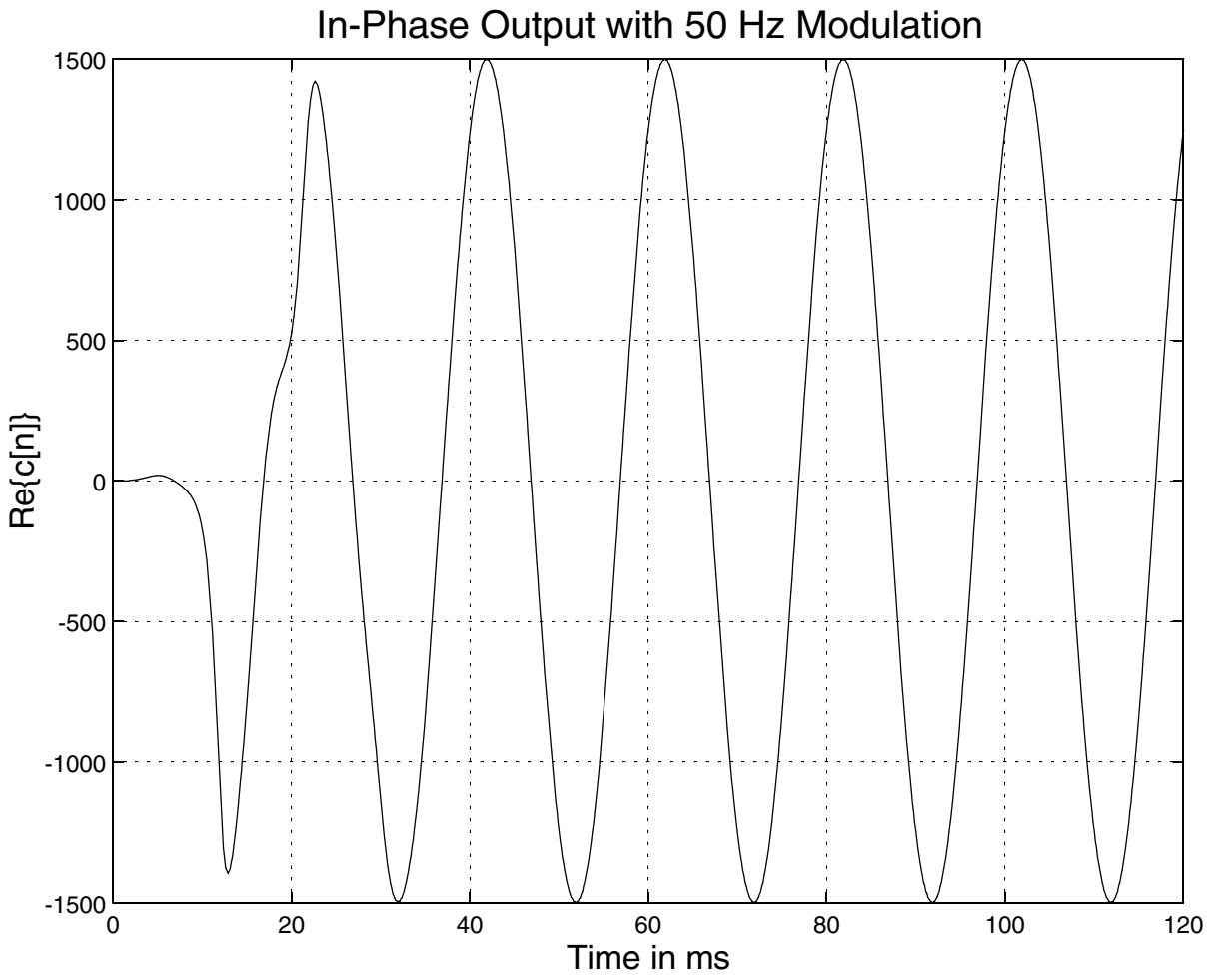
- The signal $v[n]$ scaled to represent frequency and time shifted so zero corresponds to the start of the 100 Hz step is shown below



- If the unmodulated carrier is replaced by a 50 Hz sinusoidal modulation carrier, and the loop is given an initial 0.1 Hz step to aid acquisition, e.g.,

```
if i <= 1000
    accum = mod(accum + 2*pi*(F0+.1)/Fs + v(i),2*pi);
```

the following output is obtained



Old C67x Material (needs rework)

Real-Time Implementation on the C6711 DSK

- A real-time implementation of the costas loop was created in C using CCS
- The Hilbert transform FIR filter used in the previous MATLAB simulation was utilized
- The code listing is given below

```
//Costas_pcm.c
//A Costas loop carrier tracking loop program
//A modified version of Fir_pcm.c using the PCM 3003
```

```
//Mark Wickert, May 2002

//project Costas.pjt
#include <math.h>
#include "Hilbert_31.h"      //coefficients in float format

int rand_int(void);

float Fs = 18300;//8000.0;            //desired Fs
extern float Act_Fs;         //actual Fs from SampleRate()

// Some global variables
float DLY[NH];               //buffer for delay samples
float in, out_i, out_q;
float accum, q, q_old, v;
float twopi;
float T;
float wo;
float kv, kp, ka;
int s_indx;
float samples[128];
short select_output, select_buffer;
float noise_level;

void hilbert_xform()
{
 int i;
 float acc;

 acc = 0.0;
 DLY[0] = in;
 for (i=0; i<NH; i++)
 {
          acc += Hilbert[i] * DLY[i];
 }
 out_i = DLY[15];
 out_q = acc;
 for (i=NH-1; i>0; i--)
 {
          DLY[i] = DLY[i-1]; /*Update Buffers*/
```

```
 }
}

void cmplx_trans(float arg)
{
  float temp;
  temp = out_i;
  out_i = out_i * cos(arg) + out_q * sin(arg);
  out_q = out_q * cos(arg) - temp * sin(arg);
}

interrupt void c_int11()     //ISR
{
          //Get sample from ADC
          in = (float) input_left_sample();
          //Use the next line add noise to the input signal
          in += 0.05*noise_level*((short) rand_int());
          /*Hilbert transform the input signal*/
          hilbert_xform();
          /*Multiply by the VCO Output as exp(-j*phi)*/
          cmplx_trans(accum);
          /*Form Costas Loop Error Signal and Scale*/
          q = out_q * out_i * kv;
          /*Enter the loop filter with q[n]*/
          v = v + ((kp+ka) * q) - (kp * q_old);
          q_old = q;
          /*Drive the VCO with v[n]*/
          accum += wo*T + v;
          while (accum >= twopi) accum -= twopi;
          //Extract samples for viewing
          if (select_buffer == 1)
                                samples[s_indx] = accum;
          else if (select_buffer == 2)
                                samples[s_indx] = v;
          else if (select_buffer == 3)
                                samples[s_indx] = out_i;
          else
                                samples[s_indx] = in;
          s_indx += 1;
          if (s_indx == 128) s_indx = 0;
```

```
          /*The final outout is the in-phase or real part*/
          //Return 16-bit sample to DAC
          if (select_output == 1)
                                output_left_sample((short)
out_i);
     else if (select_output == 2)
                                output_left_sample((short) in);
     else
                                //output_left_sample((short)
(500*cos(accum)));
                                output_left_sample((short)
(10000*v));

  return;                            //return from ISR
}

void main()
{
  accum = 0.0;
  q_old=0.0;
  v=0.0;
  twopi = 8*atan(1);
  T = 1/18300.0;              //Sampling Period
  wo = twopi*4000;           //VCO quiescent frequency
  kv = 1e-7, kp = 0.05, ka = 0.002;
  s_indx = 0;
  select_output = 1; //in-phase output select
  select_buffer = 1; //load accum into samples[] buffer
  noise_level = 0;   //initialize noise level at 0
  comm_intr();               //init DSK, codec, McBSP
  while(1);                  //infinite loop
}

int rand_int(void)
{
          static int a = 100001;

          a = (a*125) % 2796203;
          return a;
}
```

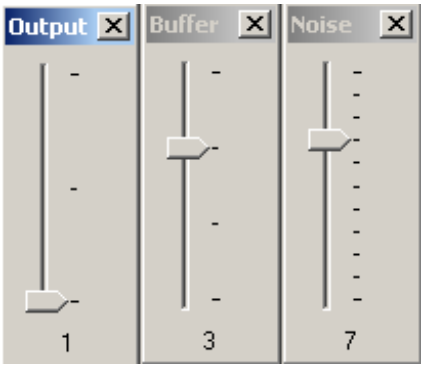
```
/*HILB_31.H-HEADER FILE COEFF M=30 FOR HILBERT Transform*/
#define NH 31         /*length of impulse response*/
const float Hilbert[NH] = {/* filter coefficients*/
-4.1956e-003, 1.3525e-015, -9.2821e-003, 5.2595e-016,
-1.8836e-002, 1.0162e-015, -3.4401e-002, 8.0018e-016,
-5.9552e-002, 1.1119e-015, -1.0304e-001, 7.7037e-016,
-1.9683e-001, 2.1310e-016, -6.3135e-001,           0,
 6.3135e-001,-2.1310e-016,  1.9683e-001,-7.7037e-016,
 1.0304e-001,-1.1119e-015,  5.9552e-002,-8.0018e-016,
 3.4401e-002,-1.0162e-015,  1.8836e-002,-5.2595e-016,
 9.2821e-003,-1.3525e-015,  4.1956e-003};

/* Sel_out.gel - Used to selct the waveform output from the loop */

menuitem "Costas Output"
// Select output to send to D/A
slider Output(1,3,1,1,sel_out)
{
          select_output = sel_out;
}
// Select output to load into samples[] buffer
slider Buffer(1,4,1,1,sel_buf)
{
          select_buffer = sel_buf;
}
// Adjust noise level applied to input
slider Noise(0,10,1,1,n_lvl)
{
          noise_level = ((float) n_lvl)/10;
}
```

- The sampling rate has been increased to 18.3 kHz and the NCO (VCO) quiescent frequency increased to 4000 kHz
- Noise may also be added to the input signal which is supplied from a function generator
- A GEL file defining three sliders is used to

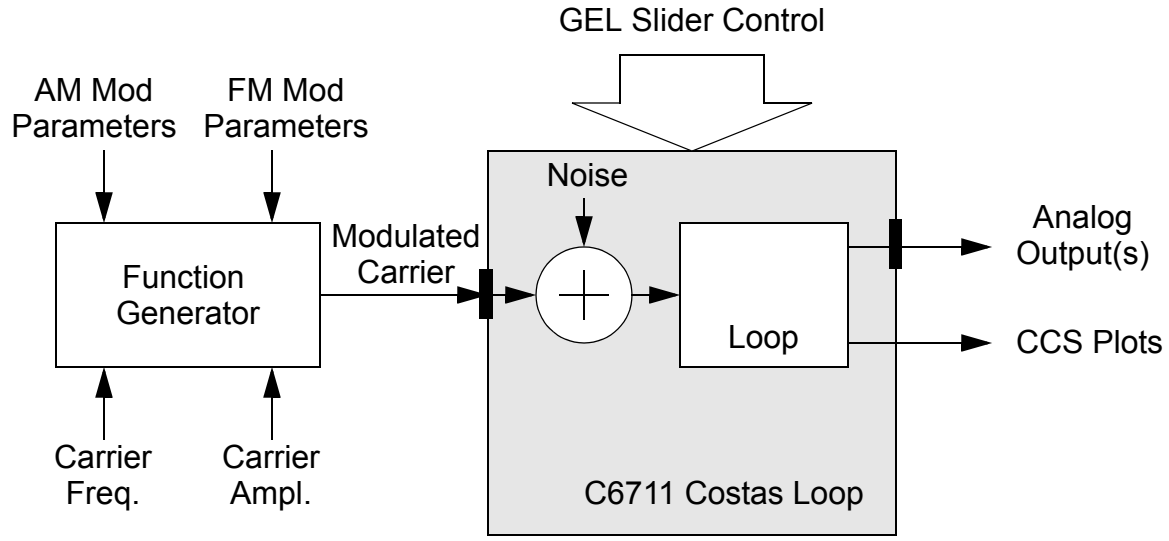
– The slider Output selects the D/A output: 1 = in-phase signal `out_i`, 2 = input + noise signal `in`, 3 = loop filter output `10000*v` or NCO output is uncommented `5000*cos(accum)`

– The slider Buffer selects the signal to be saved in the array `samples[128]` for viewing in a CCS plot window: 1 = accumulated phase `accum`, 2 = loop filter output v, 3 = in-phase output `out_i`, 4 = input signal plus noise `in`

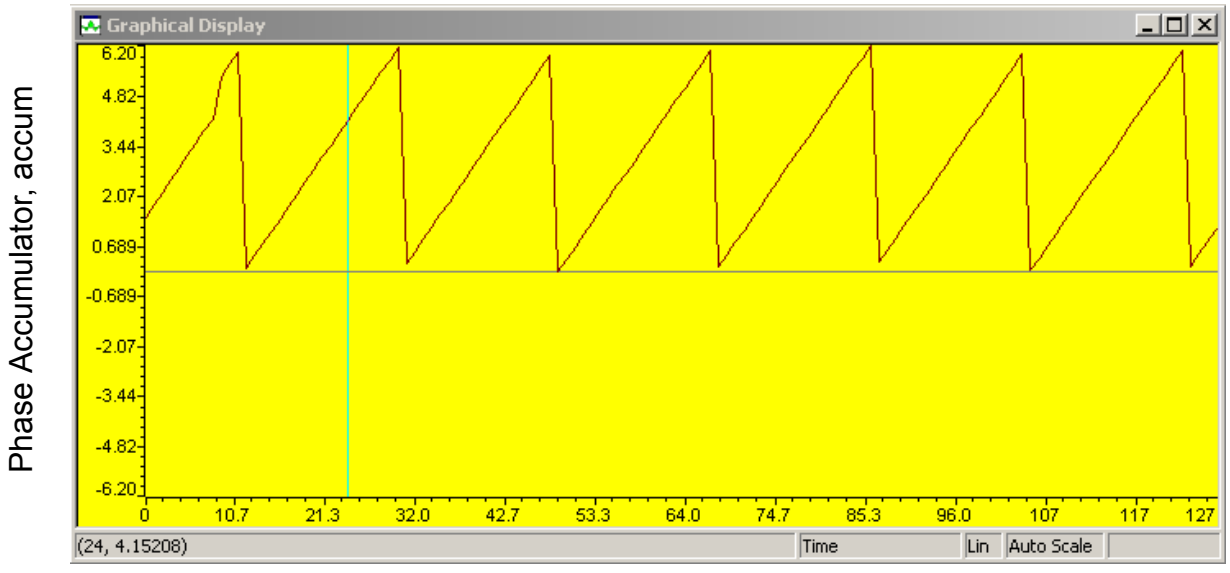
– The slider Noise controls the level of uniformly distributed white added to the input signal (Gaussian noise would be more realistic)



- To test set-up is shown below:

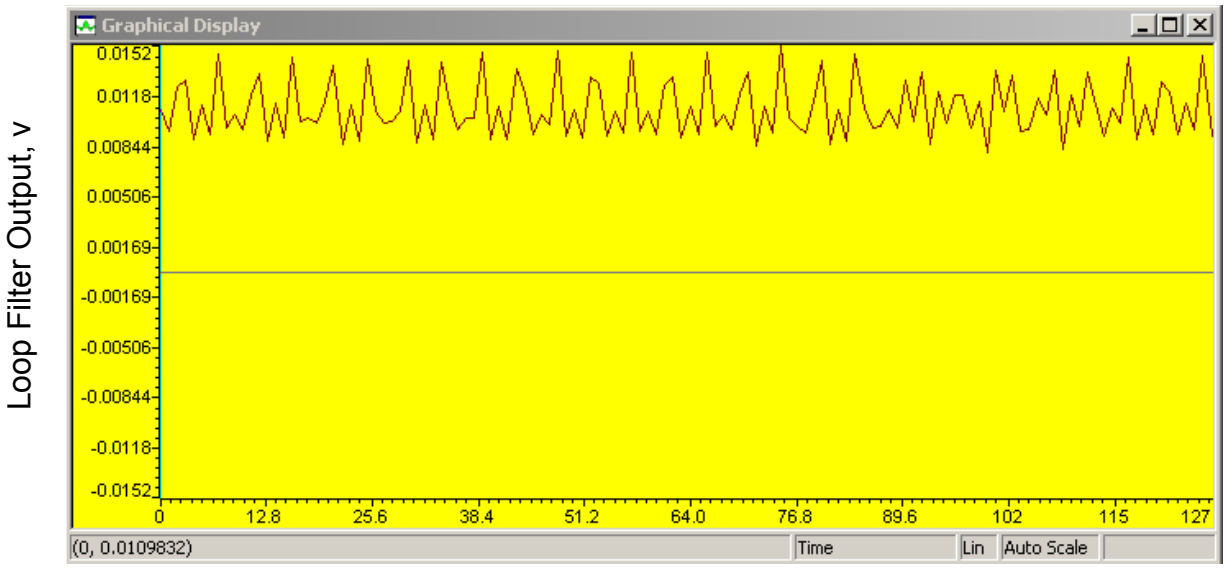


- Results are captured in the form of CCS graph windows and optionally audio files could have been captured via the sound card and imported into MATLAB
- Shown below is the NCO accumulator value, `accum`, for 1000 Hz input; as expected we have a phase ramp since no modulation is present



- Shown below is the loop filter output `v` for a 4000 Hz input;

as expected the error signal into the NCO is very small because the NCO quiescent frequency is 4000 Hz



- With no modulation applied the in-phase output is a large positive or negative constant, e.g., ± 13000, depending upon which phase the loop locks to, $0°$ or $180°$ and the amplitude of the input signal
- Now we apply amplitude modulation (AM) to the incoming carrier wave, in practice this would be digital modulation such as bi-phase shift keying (BPSK)
- The function generator cannot supply suppressed carrier modulation, e.g.,

$$r(t) = a(t)\cos(2\pi F_c t) \qquad (10.42)$$

where $a(t)$ could be a binary bit stream of the form

$$a(t) = \sum_{n=-\infty}^{\infty} c_n p(t - nT_b) \tag{10.43}$$

with $c_n = \pm 1$ equally likely random bits, $p(t)$ is some pulse shape, e.g., rectangle or square-root raised cosine, and T_b is the bit period

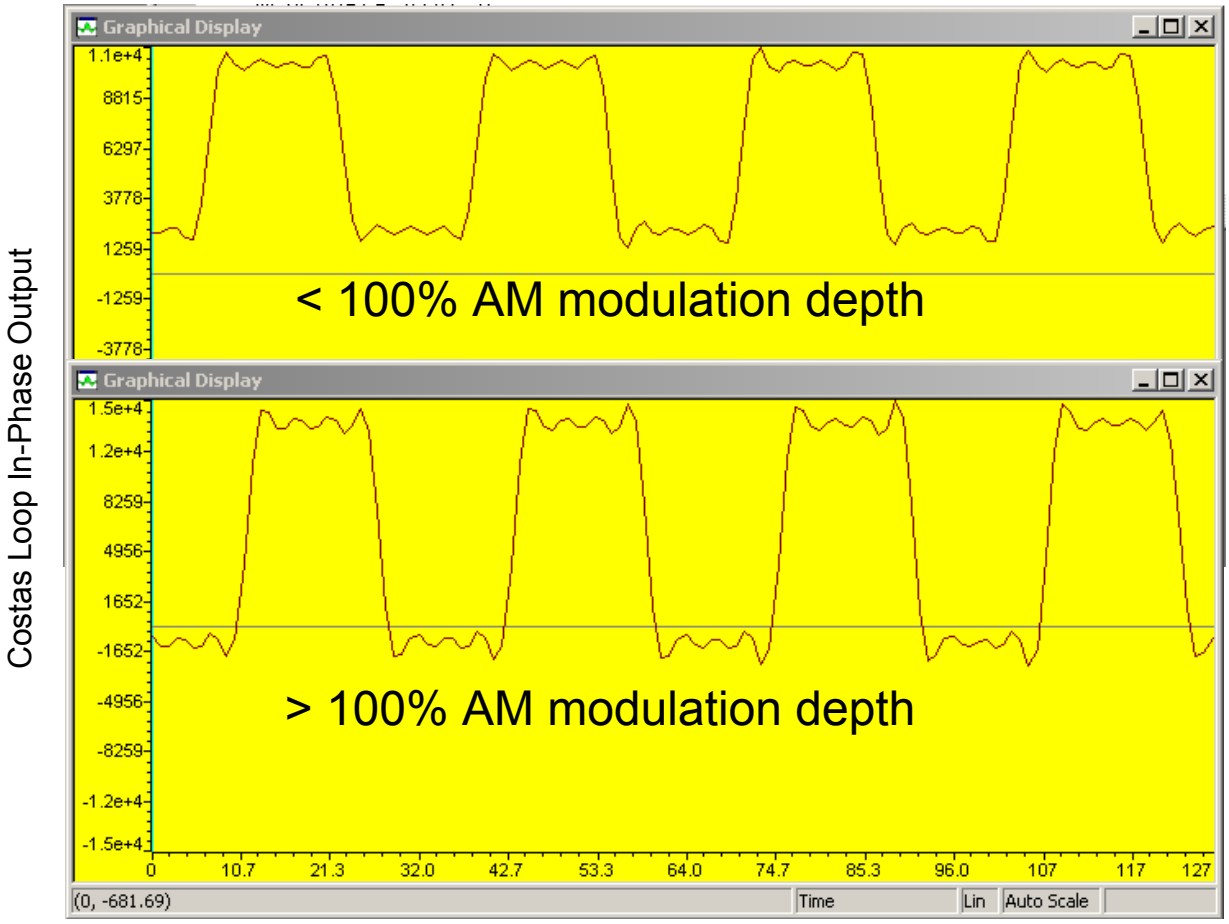
- The function generator used in the tests can only produce AM modulation with the modulation depth just able to exceed 100%, so the input supplied is of the form

$$r(t) = [\beta + a(t)]\cos(2\pi F_c t) \tag{10.44}$$

with the negative swings of $a(t)$ just slightly greater than β (being equal implies a 100% modulation index)

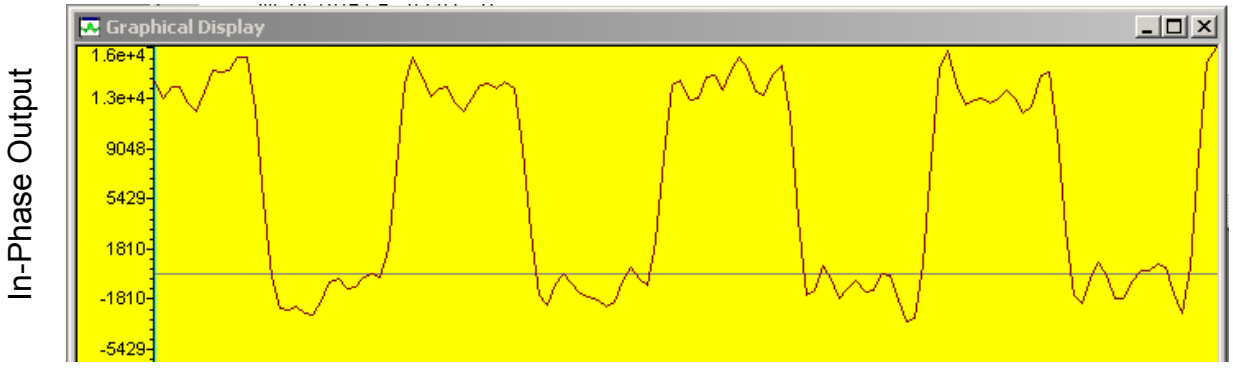
- The wave shapes available for $a(t)$ include sine, square and triangle waveforms of varying frequency
- The Costas loop is able to track regardless of the modulation index, even if the carrier frequency itself becomes fully suppressed, that is if $\beta = 0$
- Squarewave amplitude modulation (like a 1010 bit pattern) is applied and the in-phase output is observed for different AM

modulation depths



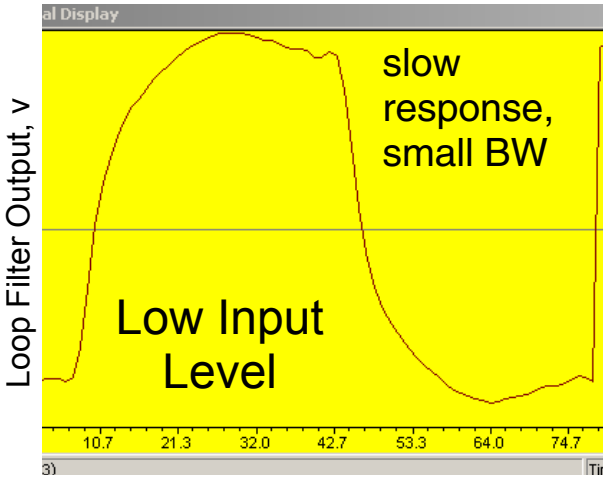
– In the above captures there is no noise present

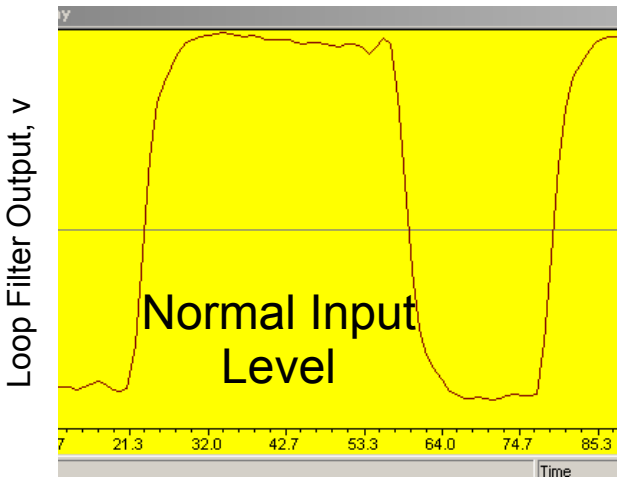
- Adding noise the signals looks and sounds *noisy*

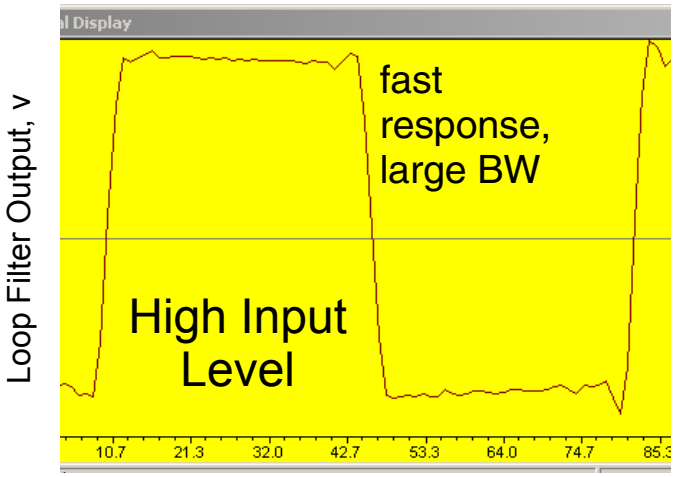


- To learn more about the tracking characteristics of the loop, the amplitude modulation is removed and apply frequency modulation (FM)

- With FM applied the in-phase output remains flat as there are no amplitude fluctuations of the input signal
- The loop error signal v must now follow the frequency deviations, and should reflect the approximate step response of the closed-loop system
- Using square wave FM with a fixed peak-to-peak deviation the input amplitude was varied to see the changes in loop bandwidth and damping factor





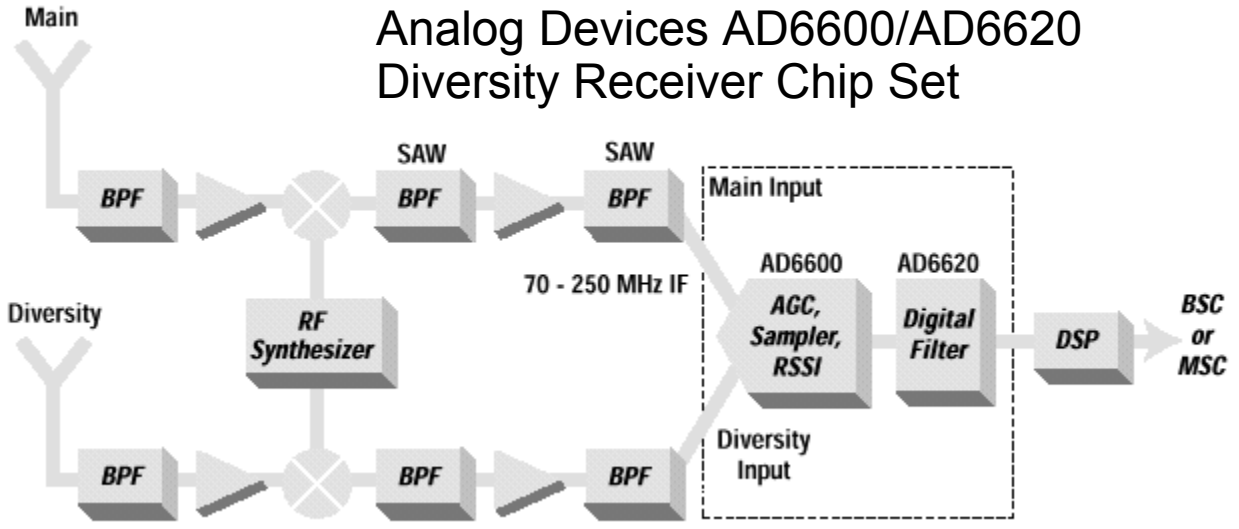


Commercial Receiver Architectures (very old, needs rework)

There are many vendors who make DSP based ASICs for communications applications. Two vendors considered here are Analog Devices and Intersil (formerly Harris Semiconductor).

Recent Analog Devices ASICs

- The *AD6600/AD6620* chip set is design for building a DSP based diversity receiver for air interface standards such as GSM, CDMA, IS136, and others
- Two IF-Baseband channels are incorporated into the design (AD6600) using a direct IF sampling approach at 70-250 MHz (undersampling) with 11-bit 20 MSPS A/D's The AD6620 is a programmable digital filter chip which extracts the desired spectral translates from the sampler

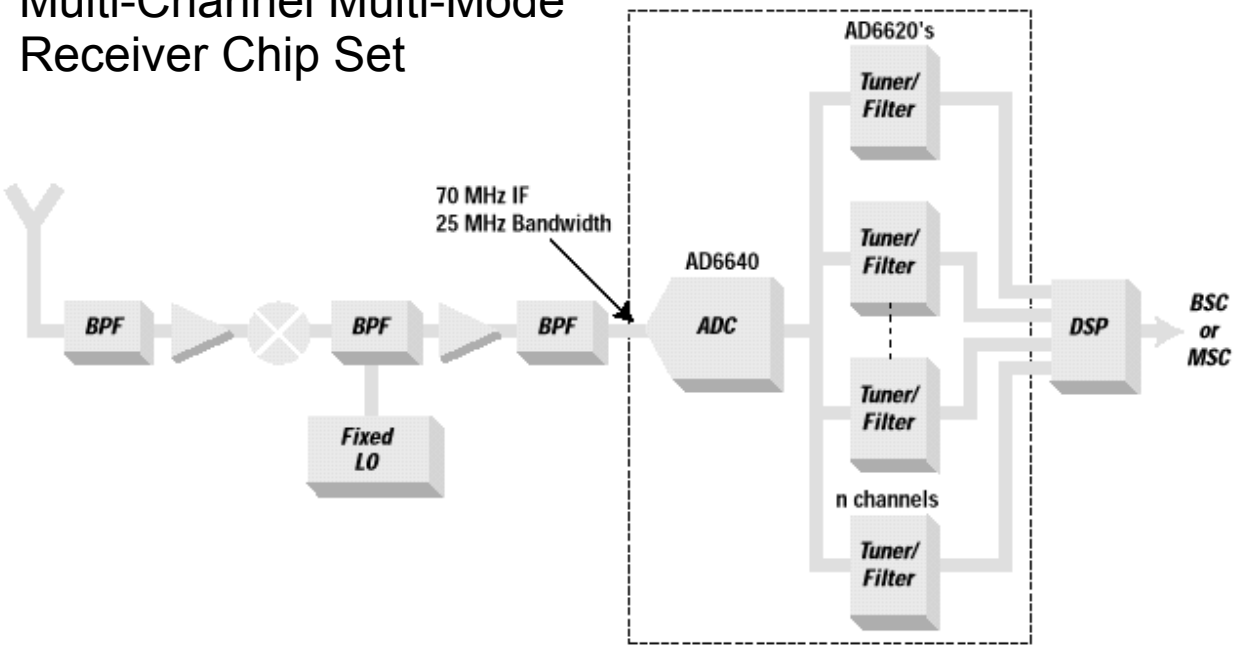


- By replacing the AD6600 with the *AD6640* wide band A/D a

different receiver architecture can be created

- The AD6640 can digitize 25 MHz of spectrum at once
- It has the needed spurious-free dynamic range (SFDR), 80-100 dB, to handle say a 30-channel AMPs system
- Following the A/D narrowband channels are obtained by individual AD6620s, thus allowing a cellular base station to only need a single RF front-end

Analog Devices AD6640/AD6620 Multi-Channel Multi-Mode Receiver Chip Set

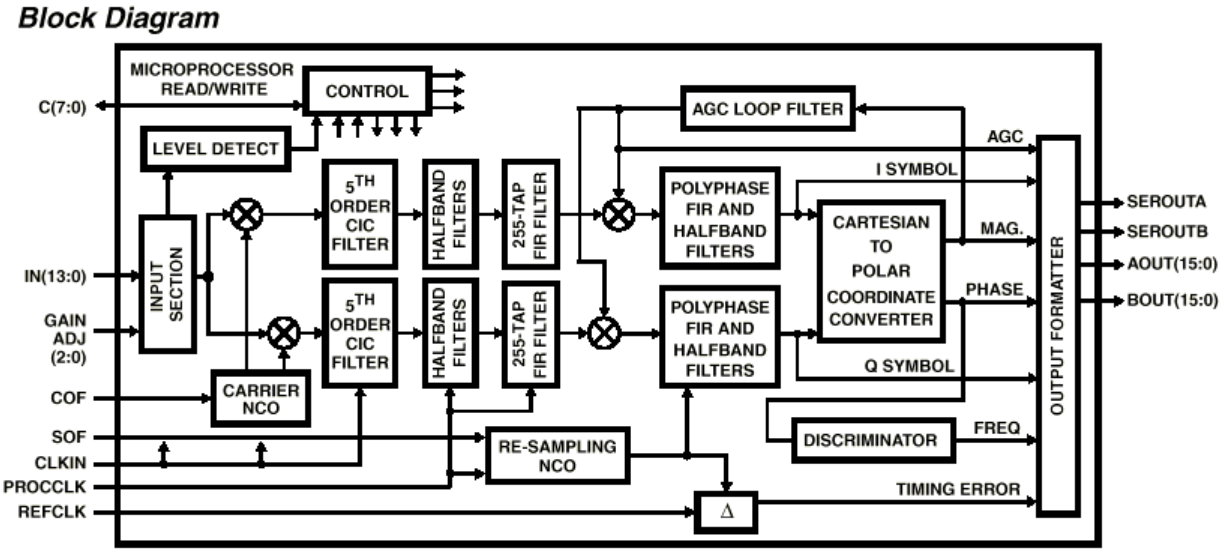


Recent Intersil (Harris) ASICs

- Harris has a whole family of DSP based communications ASICs
- A general purpose receiver front-end, less the high speed 14-bit A/D, is the *HSP50214* digital down converter
- This chip takes an IF input samples at up to 52 MHz, and per-

forms complex baseband down conversion on any subband, as large as 625 kHz lowpass

- Signal outputs are provided in seven different formats, three of which are:
  - Cartesian (I, Q)
  - Polar (R, θ)
  - Filtered frequency ($d\theta/dt$)

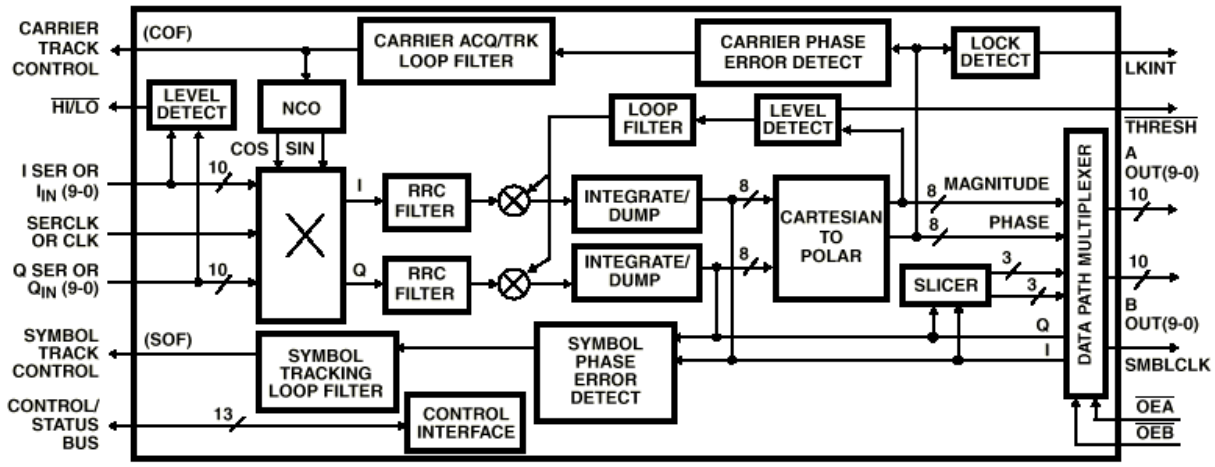


Intersil (Harris) HSP50214

- A related chip is the *HSP50210* digital Costas loop
- This ASIC can take the complex baseband outputs from the HSP50214 and perform a variety of demodulation tasks (all at up 52 MHz clock rates):
  - BPSK, QPSK, 8-PSK, OQPSK
  - FSK
  - AM, FM

- Both carrier phase tracking and symbol tracking phase-lock loops are included

Block Diagram



Intersil (Harris) HSP50210