Chapter 3
Narrow Band & Wide Band Processing Basics

Abstract In this chapter we describe and implement the narrow band processing elements NCO and FIR decimators, and how to easily find the image frequency that should be rejected. In this chapter we describe and implement the wide band processing elements of various shaping windows and FFT implementation types.

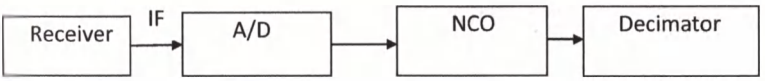
Keywords NCO · DDC · FIR · Decimation · CIC · Compensator · Aliasing · Image · Nyquist · Convolution · Shaping window · wola · FFT · bro · dro · Twiddle factor · Radix 2 · Radix 4

RF frequencies in some certain band or preselector are usually converted by a receiver to a fixed IF frequency followed by an analog filter at the desired bandwidth, higher bandwidth enables more frequencies in parallel but is more sensitive with respect to dynamic range and spurs.

Using digital signal processing, the IF analog information is converted to bits by an a2d converter, sampling at Fs Hz. Usually the IF frequencies are located at Fs/4, 3*Fs/4, 5*Fs/4... called Nyquist zones so that a maximum and symmetric bandwidth is attained.

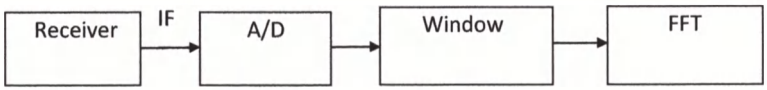
Higher IF helps the receiver designer to avoid spurs but needs a more expensive a2d that has acceptable SNR and ENOB, as the ratio between Fs and IF is lower, jitter increases and decreases SNR at the a2d output.

There are two basic ways to process the IF frequency band, one is the narrow band processing, described by the following diagram:



The digital IF samples stream is shifted to baseband, where it is filtered and decimated to the desired output bandwidth and sampling frequency, each channel requires dedicated processing.

The second way is to use wide band processing as described by the following diagram:



The digital IF sample stream undergoes a frequency shaping window that renders separation of channels (bins), then an N size FFT is performed which results in one complex number per bin.

If a stream of such complex numbers per bin over time is desired, then consecutive FFTs shifted in time are performed.

3.1 Narrow Band Processing

3.1.1 Introduction

In this chapter we describe and demonstrate design considerations and implementation of numerically controlled oscillator (NCO) and decimators which are the elements of narrow band processing, and how to easily find the image frequency that should be rejected when shifting frequencies.

3.1.2 Numerically Controlled Oscillator

The numerically controlled oscillator (NCO) shifts digitally the digital stream from the a2d by f1, making the operation in Matlab as follows:

```
j = sqrt(-1);
t = (0:N-1)/Fs;
Nco_out = a2d_out.*exp(-j*2*pi*f1*t);
```

This operation generates an image frequency in addition to baseband, this image should be rejected by the first LPF (low pass filter) after the NCO, since the passband and stopband frequencies are also determined by the decimation rules then the strict requirement determines.

The generated image frequency is equal to twice f1 and aliased to (3*Fs – 2*f1), in order to not bother with calculating the resulting frequencies, a simple Matlab code renders the correct image frequency as follows:

Code 3.1.1

```
Fs = 160e6;
f12 = 220e6;
f11 = f12 – 2e6;
f13 = f12 + 2e6;
N = 8192;
t = (0:N-1)/Fs;
inp = sin(2*pi*f11*t) + sin(2*pi*f12*t) + sin(2*pi*f13*t);
nco = inp.*exp(-j*2*pi*f12*t);
win = chebwin(N)';
win = win/sum(win);
Sigw = nco.*win;
Fout = abs(fftshift(fft(Sigw,N))).^2/N/N;
Fout = Fout/max(Fout);
Fx = (-4095:4096)*Fs/N;
```

That renders the following figure:

As seen from Fig. 3.1, the image frequency aliased to (3 * 160–2 * 220) Mhz should be rejected.

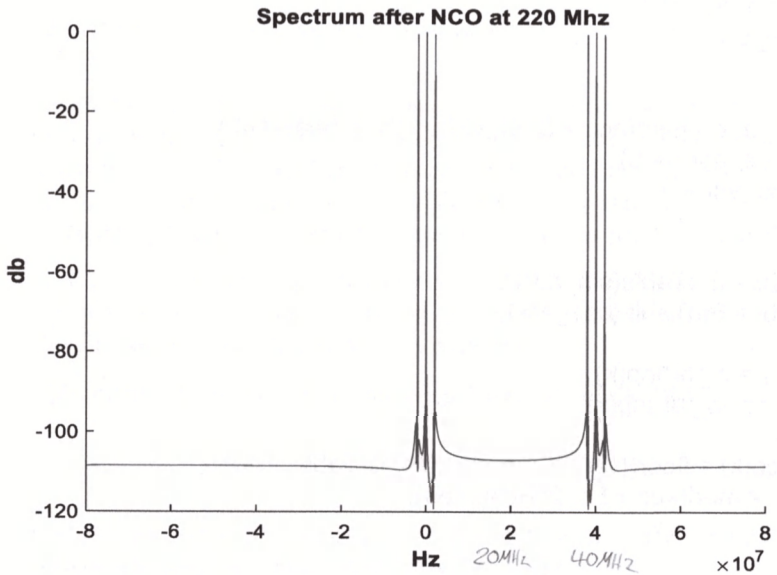


Fig. 3.1 Finding the image frequency after the NCO operation

For implementing the NCO operation, a sin table from 0 to 2*pi with 2^ScaleTab entries is used, described by a Matlab code as

```
ScaleTab = 12;
SinTable = round(sin([0:2^ScaleTab -1)]/(2^ScaleTab)*2*pi)*(2^ScaleTab – 1));
```

The table coefficients are written to an *.h file as described in Sect. 2.2.2 before, ScaleTab is chosen by the required resolution of the frequency shifting, which is the sampling frequency of the a2d divided by the table size 2^ScaleTab.

Another consideration to increase ScaleTab is that the spectrum of the complex NCO output will not have spurs or elevated noise floor.

A Matlab code that implements NCO operation is

Code 3.1.2

```
acc = 0;
IF = Fs/8;
Fc = round(IF/Fs*2^ScaleTab);

for k = 1:length(Inp)

 sin_adr = acc;
 if (sin_adr == 0)
  sin_adr = 1;
 end

 cos_adr = mod(acc + (2^ScaleTab)/4,2^ScaleTab);
 if (cos_adr == 0)
  cos_adr = 1;
 end

 s_tbl = SinTable(sin_adr);
 c_tbl = SinTable(cos_adr);

 var1 = c_tbl*Inp(k);
 var2 = -s_tbl*Inp(k);

 Xnco(k) = floor((var1 + j*var1)/2^ScaleTab);
 acc = mod(acc + Fc, 2^ScaleTab);

end
```

The cos() value is found from the sin table using the relation cos(a) = sin(pi/2 + a). Suppose that a 16 bit a2d is used, the sin table we used has 2^15 elements of type

Code 3.1.3

```
void nco(short *x, int n, int dp, short *y)
{
int sin_val, cos_val, acc, i;

acc = 0;
for (i = 0; i < n; i++)
 {
 sin_val = Sin_Tab[acc];
 cos_val = Sin_Tab[(acc + 8192) & 0x7fff];
 y[i] = (cos_val*x[i]) >> 15;
 y[I + n] = (-sin_val*x[i]) >> 15;
 acc = (acc + dp) & 0x7fff;
 }
}
```

The phase increment dp is round(f1/Fs*2^ScaleTab), the software produces a short array of the real part followed by a short array of the imag part.

If f1 is negative, then dp is round((Fs + f1)/Fs*2^ScaleTab), the above verified code consumes 31 nS per sample on an I5 2.9 GHz.

3.1.3 *Finite Impulse Response Decimators*

A decimator reduces the input sampling frequency to a lower output one, this operation requires a low pass filter (LPF) at the input sampling rate, followed by a suitable decimation. The highest passband frequency Fp at the LPF output should not exceed half of the output sampling frequency if the decimator is real, and not exceed the output sampling frequency if the decimator is complex, as per Nyquist theorem.

As we will see later, the most efficient way to design a decimator in terms of processing time is to divide the decimation between several stages or multirate method, the first stage should follow the rule of thumb

Fp = min(decimation constraint as above, image frequency of the NCO)

$$\text{Fstop} = \text{Fp} + 2^{*}\left(\text{Fs}/\left(2^{*}\text{decl}\right) - \text{Fp}\right)$$

where decl is the 1st stage decimation, for the next stages if any, the same rule applies besides rejecting the image frequency of the NCO.

Decimators are usually implemented by FIRs with possible addition of cascaded integrator comb (CIC) filters, they may also be implemented by IIRs, all types will be described later.

Earlier versions of the Matlab used functions such as Remez, Firls, the current one uses Firpm to design filters, we found that using the fdesign tool in the signal processing toolbox renders efficient designs in short time, a Matlab code that does it is

Code 3.1.4

```
Fs1 = 160e6;
dm = fdesign.lowpass('Fp,Fst,Ap,Ast',20e6,79e6,0.1,105,Fs1);
hm = design(dm);
b13 = hm.numerator;
b13 = b13/(sum(b13) + eps);
```

Fp is the largest passband frequency at which the filter gain is -1 db, lower value relative to the sampling frequency Fs1 renders more coefficients for the designed FIR.

Fstop is the smallest stopband frequency, the ratio between Fstop and Fp is called the shape factor of the filter where a value of less than 1.5 for the total response is common, reducing shape factor renders more coefficients for the designed FIR, Fstop should be less than Fs1/2.

Ap is the ripple in db at the passband, a value of 0.1 is common, reducing Ap renders more coefficients for the designed FIR.

Ast is the attenuation in db at the stopband, values between 90 and 110 are common to get actually −80 to −100 db, lower attenuation renders less coefficients for the designed FIR.

3.1.4 Low Decimation Ratio Finite Impulse Response Filters

When we say low decimation we mean 2–8.

We will demonstrate the design and verification by an example: Suppose that we need to design a decimator by 4, with final bandwidth of 30 Mhz, following a receiver with IF frequency of 120 Mhz sampled by a 160 Mhz a2d.

The efficient design in terms of processing time is to use multirate design, which is to divide the decimation as close as possible to decimation by 2 units, for our example that means 2 FIRs, the 1st closer to the a2d or NCO receives a double rate of samples relative to the second, but its shape factor requirement is lower, the second and final FIR has a shape factor of 1.47 measured from the total response.

A CIC filter, declared by the Z transform response $1/M * (1 - Z^{-M})/(1 - Z^{-1})$ where M is the filter order may be used instead of the 1st FIR, but the droop of this filter even for $M = 2$ is such that the total response cannot have a passband ripple of 0.1 db.

Trial and error method should be used by watching the figures of total frequency response and the frequency verification measure in Sect. 2.2.1, as seen from the total response the decimation rules of thumb should be kept at the end points Fp and Fstop but may be violated elsewhere.

The two FIRs design looks like

$$Fp = \frac{Fs1}{4}$$

Code 3.1.5

```
res = 2048;
Fs1 = 160e6;
dec1 = 2;

F1 = Fs1/2/dec1/res*(0:res*dec1 – 1);
dm = fdesign.lowpass('Fp,Fst,Ap,Ast',20e6,79e6,0.1,105,Fs1);
hm = design(dm);
b13 =  hm.numerator;
b13 = b13/(sum(b13) + eps);
ha1 = freqz(b13,1,F1,Fs1);
```

That renders 9 coefficients and

```
F2 = Fs1/2/dec1/res*(0 : (res – 1);
dm = fdesign.lowpass('Fp,Fst,Ap,Ast',13.87e6,22e6,0.1,105,Fs1/2);
hm = design(dm);
b23 =  hm.numerator;
b23 = b23/(sum(b23) + eps);
ha2 = freqz(b23q,1,F2,Fs1/2);
```

That renders 42 coefficients

```
hat = ha1(1:length(F2)).*ha2;
```

The second filter is computed with the same resolution as the 1st but on half of frequencies, the total response is calculated on the same points and plotted on the second range as follows (Fig. 3.2).

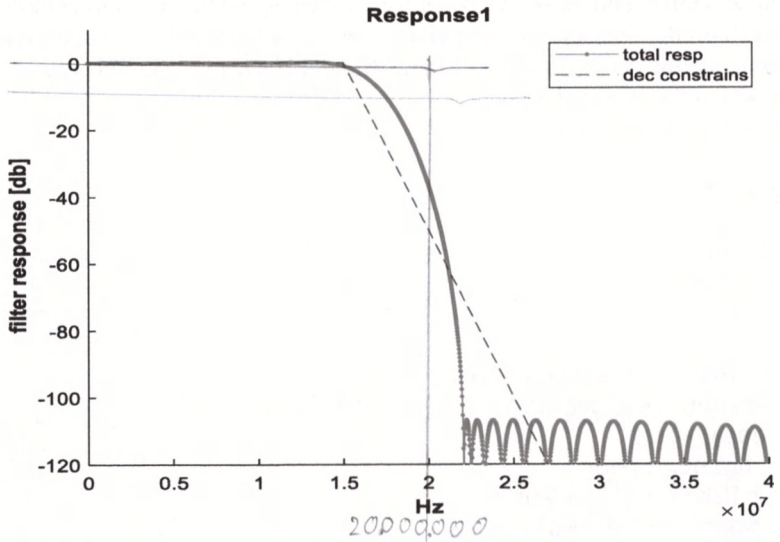


Fig. 3.2 Total frequency response for floating point FIR implementation

After design verification in frequency as described in Sect. 2.2.1 a C implementation for floating point inputs and coefficients which differs for even and odd number of coefficients looks like

Code 3.1.6

```
void dec_even(float *x, int n, int dec, float *y)
{
int i, k, count, len2;
float sum1,sum2;

count = 0;
len2 = 2*len_b_dec;
for (k = (len2 – 1); k < n; k += (2*dec))
 {
 sum1 = (float)0.;
 sum2 = (float)0.;

 for (i = 0; i < len_b_dec; i++)
  {
  sum1 += b_dec[i] * (x[k – i] + x[k – len2 + i + 1]);
  sum2 += b_dec[i] * (x[k – i + dec] + x[k – len2 + i + 1 + dec]);
  }

 y[count++] = sum1;
 y[count++] = sum2;
 }
}
void dec_odd(float *x, int n, int dec, float *y)
{
int i, k, count, len2;
float sum1,sum2;

count = 0;
len2 = 2*len_b_dec – 1;
for (k = (len2 – 1); k < n; k += (2*dec))
 {
 sum1 = (float)0.;
 sum2 = (float)0.;

 for (i = 0; i < (len_b_dec – 1); i++)
  {
  sum1 += b_dec[i] * (x[k – i] + x[k – len2 + i + 1]);
  sum2 += b_dec[i] * (x[k – i + dec] + x[k – len2 + i + 1 + dec]);
  }

 sum1 += b_dec[len_b_dec – 1]*x[k – len_b_dec + 1];
 sum2 += b_dec[len_b_dec – 1]*x[k – len_b_dec + 1 + dec];

 y[count++] = sum1;
 y[count++] = sum2;
 }
```

Two outputs are performed for each loop step, so the number of steps is half, if several computing units are available as in some DSPs, then more outputs could be computed every step.

Len_b_dec is half the filter length for even and half the filter length + 1 for odd length numbers.

The processing time is 74 μS for the 1st filter and 160 μS for the 2nd filter, both for a sampling interval of 60 μS (10,000 samples) on an I5 2.9G CPU, since the processing time is about 4 times larger than the sampling interval, one has to use a 4 times faster CPU to enable full throughput on a single core or use hardware implementation. Processing at 160 Mhz is too fast for current CPU technology.

If the processing unit does not have an efficient floating point support, then the computations may be done in fixed point format.

Representing the coefficients in fixed point format may be done in Matlab by

$$b13 = \text{round}\left(b13^{*}\left(2\,\hat{}\,qu-1\right)\right)/\left(2\,\hat{}\,qu-1\right)$$

where qu is the number of bits used, when the number of coefficients increases some of the coefficients are very small which requires more bits, the same applies to stopband attenuation, where larger one requires more bits.

In the next figure the response for floating point coefficients and 16 bits fixed point coefficients for the above example are presented (Fig. 3.3).

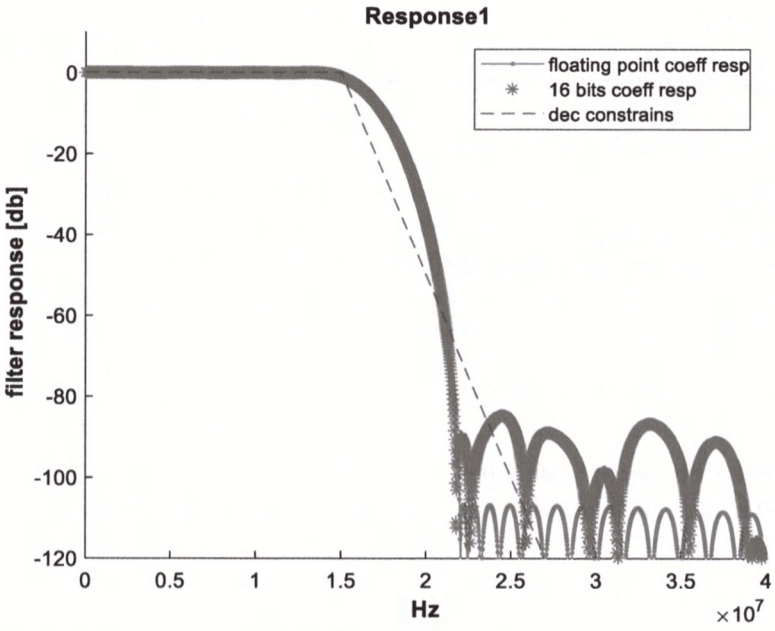


Fig. 3.3 Total frequency response for fixed and floating point FIR implementation

If the designer is satisfied with 80 db stopband rejection he may use fixed point calculations, but unless we gain with respect to processing time, the performance of floating point formats is better.

When dealing with fixed point calculations, the range of sum1, sum2 which is -2^31 to 2^31-1 must be kept, otherwise incorrect results are produced, for our example and 16 bits coefficients this is achieved by limiting the FIR's input to 15 bits signed format.

A fixed point implementation in C which differs between even and odd number of coefficients is