GENERAL DSP CODE OPTIMIZATIONS

FIR Symmetry, Polyphase Filtering, Half-Band Filters, and Real-Valued FFTs

J. Shima

1) Filters with coefficient symmetry:

Brute-force convolution is typically implemented as:

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k]$$

If all the decimation and polyphase filters are even length (Type II FIRs), we can use coefficient symmetry to speed up processing. Using two pointers in the filter history buffer, we can now implement:

$$y[n] = \sum_{k=0}^{\left(\frac{N}{2}-1\right)} h[k] \cdot \left(x[n-k] + x[n-N+1+k]\right)$$

Effectively cutting the number of multiplies in half.

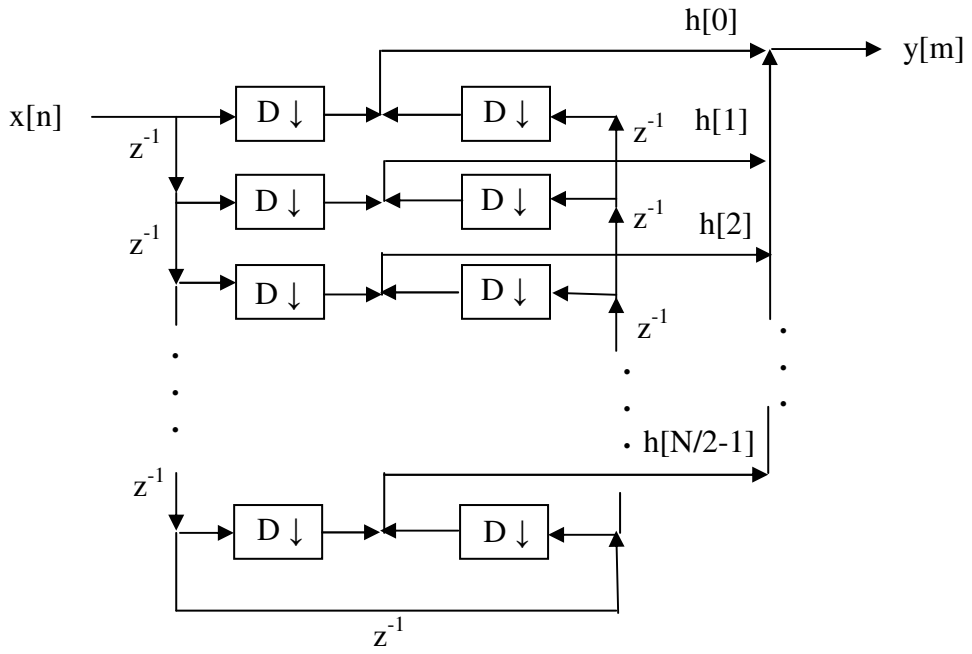
It is more efficient to implement a direct-form decimator, filling the history buffer and then computing an output point for every D samples, effectively implementing the equation:

$$y[m] = \sum_{k=0}^{N-1} h[k]x[Dm-k]$$

And using filter symmetry in our case:

$$y[m] = \sum_{k=0}^{\frac{N}{2}-1} h[k]\left(x[Dm-k] + x[Dm-N+1+k]\right)$$

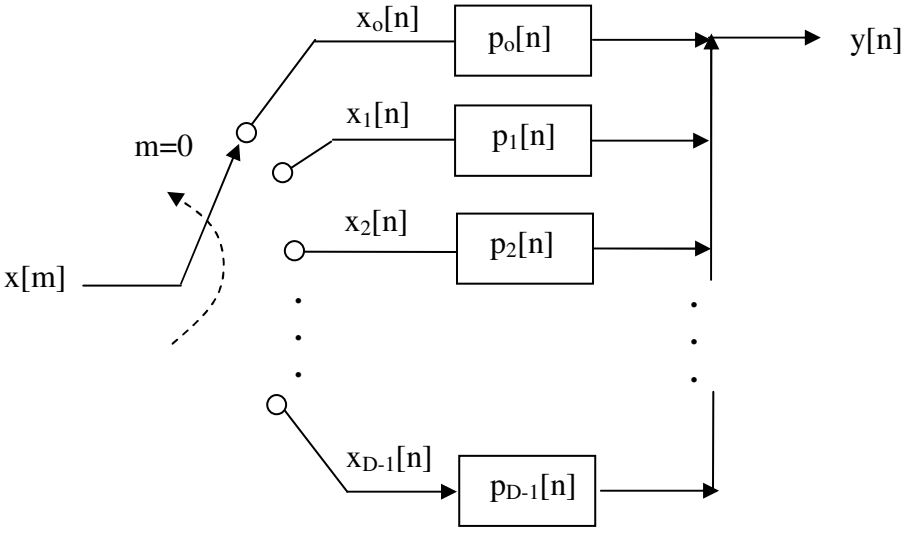
In block diagram form:



Direct-form D-to-1 FIR decimator using filter symmetry (N even)

Many decimation filters implement a polyphase structure. The polyphase structure is just another efficient filter for decimation (like the direct-form decimator), where the filter output is computed at the decimated rate. This is a reduction of *D* operations when compared to the brute force method of computing all filter outputs at the original sample rate, then discarding every *D-1* out of *D* samples.

The polyphase structure is best viewed using a commutator model for a D-to-1 decimator[1].



Polyphase D-to-1 FIR decimator: Commutator model

[1] Rabiner and Crochiere, *Multirate Digital Signal Processing*, Prentice-Hall

In the polyphase structure, the N-tap FIR coefficients are split into branches given by:

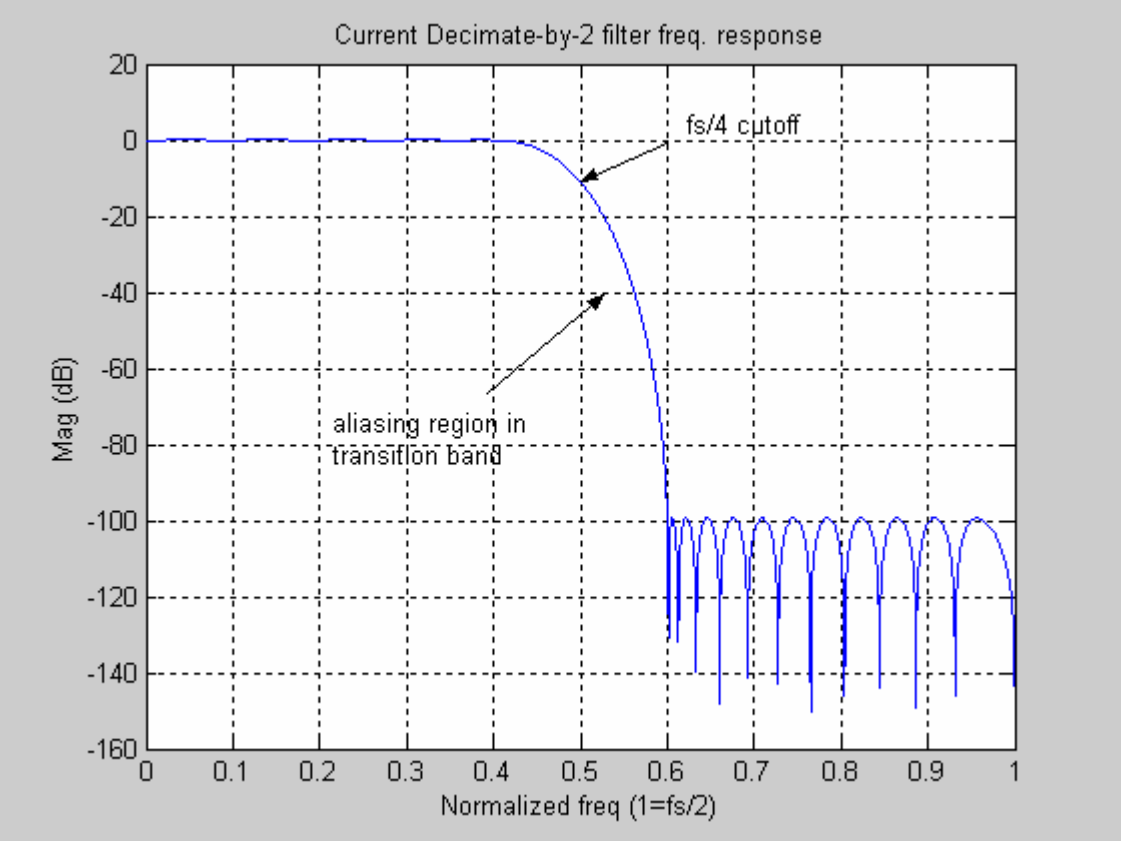
$$p_r[n] = h[nD + r], \text{ for r} = 0,1,2,\ldots,\text{D-1}$$

Then the high-rate input samples x[m] are commutated into each branch counterclockwise. After *D* samples have been injected into the filter structure (i.e. when each polyphase branch has a new sample), the filter is then computed at the decimated rate, summed, and output into y[n]. The branch filters are simply different all-pass phase delays of the original filter, which gave rise to the term "polyphase". A requirement that the number of filter taps *N* is an integer factor of the decimation rate *D* allows one to construct an efficient polyphase structure (all branches will have equal number of coeffs).

Notice in the polyphase structure, it is not straight forward to account for filter symmetry, since the filter branches are split up into *D*-tap filters to lower the computational load. However, using the proposed half-band filter (presented below) negates the need for the polyphase branches. Since the half-band filter has half the number of non-zero taps as a normal decimate-by-two filter, the computational distribution between a polyphase filter and a direct-form half-band filter is equivalent for the same order filter!

2) Half-band filters:

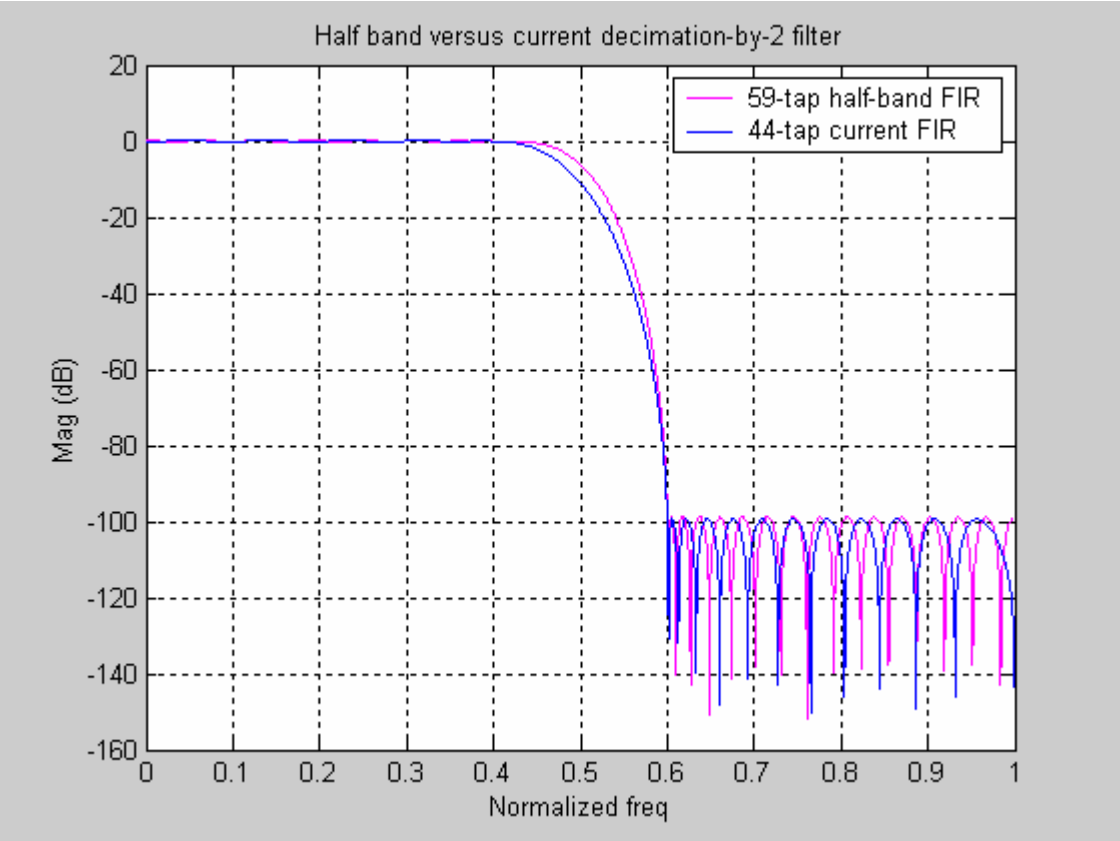
Half-band filters can be designed using the standard sinc windowing technique (using a cutoff fc=fs/4), via the Chebyshev minimax approximation, or via a Lagrange interpolator.



44-tap FIR decimate-by-2 filter

The above plots shows a 44-tap. This is the decimate-by-2 frequency response. You can see the cutoff freq is around fs/4, but you still will get some aliasing after decimation in the new fs/4 to fs/2 frequency band due to the non-ideal rolloff of the filter.

Next is an example of a half-band filter (designed via Parks-McClellan) that is practically equivalent to the current filter but with a reduced computational load:



You can see in the above plot that the current filter uses 44 taps, while the 59-tap half-band filter has *28 taps that are exactly zero*, effectively only needing compute 31 multiply-accumulates (MACs) versus 44 multiply accumulates (not accounting for filter symmetry). Thus using a half-band filter saves 13 MACs per output point in this case. Furthermore, we can most likely relax the stop-band attenuation to shorten the half-band filter length even further.

While the half-band filter is not easily realizable in a polyphase filter implementation, it still saves computations over the current polyphase structure. The calculations are just done in a direct-form decimation filter rather than a polyphase structure. In the current code each point is fed into the next stage polyphase branch in a trickle-down fashion. Each of these input points are commutated into the polyphase FIR, and then that active phase branch in the polyphase filter is computed. For an *N*-tap decimate-by-2 filter, each branch of the polyphase structure requires *N/2* MAC operations. This technique distributes the calculations over the entire decimation time, since you are not computing all of the filter branches at once when you have collected *D* input samples.

But you still have to perform more calculations than if you used a half-band direct-form FIR decimator. In fact, using a half-band filter with the same filter order would result in the same number of multiply operations as one branch of the current polyphase filter (*N/2* MAC operations). Adding in the coefficient symmetry would reduce the number of MAC operations for the half-band filter down to *N/4*!

Next is a plot of two half-band filters that have around ½ the computational load of the current decimate-by-2 FIR. The windowed FIR was generated using a Kaiser window with β=6.5. The Remez FIR was generated using the Parks-McClellan FIR design approximation in MATLAB, using the half-band frequency constraints of:

$$H\left(e^{jw}\right)+H\left(e^{j(\pi-w)}\right)=1$$

Or in the z-domain,

$$H(z)+H\left(-z^{-1}\right)=1$$

Using the facts,

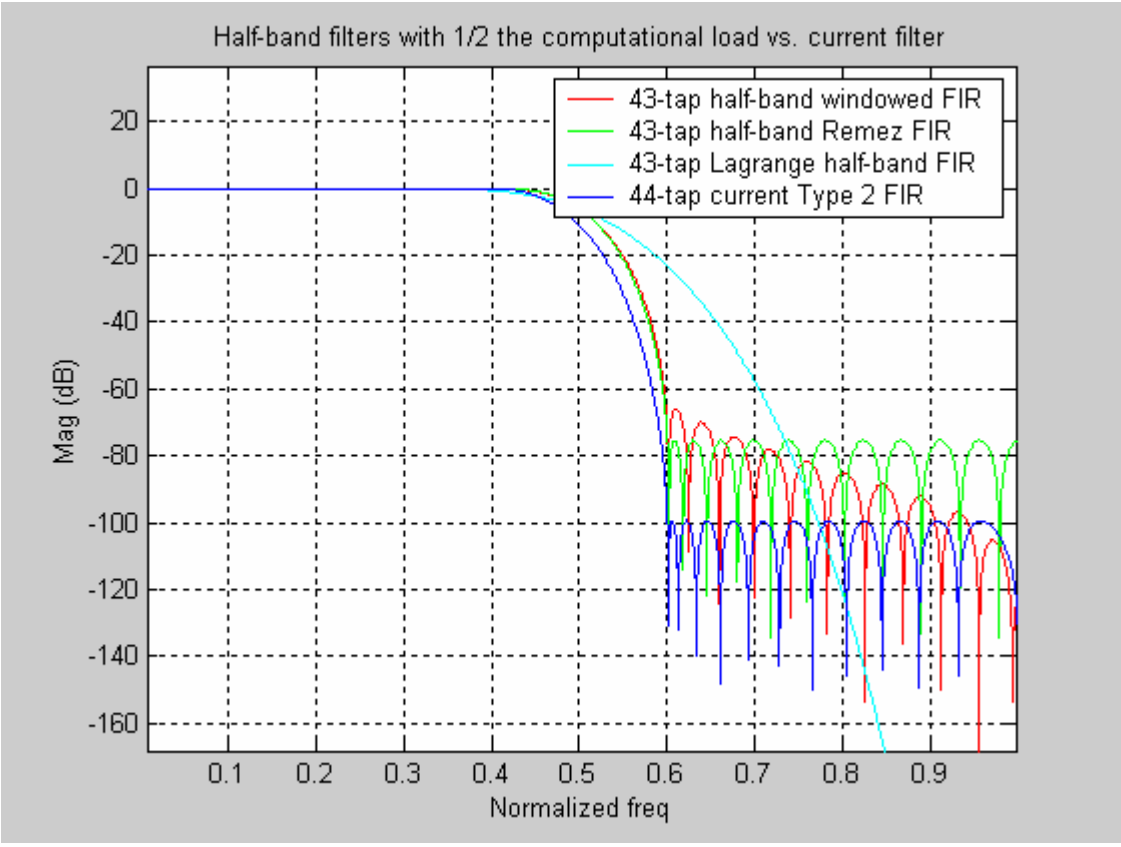
$$h[-n]\overset{z}{\longleftrightarrow}H(\frac{1}{z}) \quad \text{(time reversal)}$$

$$(z_0)^n\,h[n]\overset{z}{\longleftrightarrow}H(\frac{z}{z_0}) \quad \text{(mult by exp seq)}$$

Gives the relationship,

$$h[n]+(-1)^n\,h[-n]=\delta[n]$$

Which constrains the cutoff freq at fs/4 to be 6 dB down, h[0] = ½, and all the even taps are forced to zero. Plus the total number of taps in a half-band filter is equal to *4L+1* (L is an integer).



You can see in the above plot that the stopband attenuation is around 20-25dB higher for these half-band filters, but the gain in computational efficiency typically outweighs the difference in stopband attenuation and the cutoff transition region. Here the half-band filters only have 23 non-zero taps in the filter, which is about ½ the number of non-zero taps in the current 44-tap filter. The Lagrange half-band filter shown above is a special maximally-flat half-band filter. The trade-off is a flat passband for a wider transition band. The Lagrange filter also has a much higher attenuation in the stop band from normalized freqs of 3fs/4 up to fs/2.

In conclusion, if we use a half-band filter in a direct-form decimation structure along with coefficient symmetry, the computational load of this new filtering scheme decreases by approximately 4x with respect to a polyphase decimate-by-2 implementation.

Real-valued FFT:

We can use an N-pt. complex-valued FFT to compute the FFT of 2 real signals (each of length N), or pack a single signal (of length 2N) into the real and imaginary parts of a N-pt FFT. Each gives a computational advantage of using a complex FFT routine on real signal analysis.

Unpacking equations

For an N-pt complex FFT packed with two real-valued signals $x_1[n]$ and $x_2[n]$, we can extract the two freq. responses from Y(k):

$$\begin{aligned} X_1(k) &= \frac{1}{2}\left[Y(k) + Y^*(N-k)\right] \\ X_2(k) &= -\frac{j}{2}\left[Y(k) - Y^*(N-k)\right] \end{aligned}, \text{ for k=0,1,…,N-1} \quad (1)$$

And for an N-pt. complex FFT packed with a single real-valued signal x[n]:

$$X(k) = \frac{1}{2}\left[Y(k) + Y^*(\frac{N}{2}-k)\right] - \frac{j}{2}e^{\frac{-j2\pi k}{N}}\left[Y(k) - Y^*(\frac{N}{2}-k)\right], \text{ for k=0,1,…,N-1} \quad (2)$$

which is similar to the above but with the addition on one complex multiply. So using a first-order approximation, the dual signal complex-to-real data unpacking in Eq. 1 takes 4N additions, whereas the single signal complex-to-real data unpacking in Eq. 2 takes 4N additions and 4N multiplications.

It can be shown that Eq. 2 can be split into real (Y_R) and imaginary parts (Y_I), then simplified to give $X(k)=X_R(k)+jX_I(k)$:

$$
\begin{aligned}
Y_{R+}(k) &= \frac{1}{2}\left[Y_R(k) + Y_R(N-k)\right] \\
Y_{R-}(k) &= \frac{1}{2}\left[Y_R(k) - Y_R(N-k)\right] \\
Y_{I+}(k) &= \frac{1}{2}\left[Y_I(k) + Y_I(N-k)\right] \\
Y_{I-}(k) &= \frac{1}{2}\left[Y_I(k) - Y_I(N-k)\right]
\end{aligned}
\quad \text{, for k = 0,1,…,N/2} \quad (3)
$$

The sums and differences in Eq. 3 can be combined to create the original X(k):

$$
\begin{aligned}
X_R(k) &= Y_{R+}(k) + \cos(2\pi\frac{k}{N})Y_{I+}(k) - \sin(2\pi\frac{k}{N})Y_{R-}(k) \\
X_I(k) &= Y_{R-}(k) - \cos(2\pi\frac{k}{N})Y_{R-}(k) - \sin(2\pi\frac{k}{N})Y_{I+}(k)
\end{aligned}
\quad (4)
$$

And implementing this in assembly with multi-function instructions would further reduce the computations. For example, using a dual MAC instruction would require only N multiplications and 2N additions. Furthermore, we would only have to compute the spectrum to N/2 pts since we do not care about the mirror image.