Hideo Okawara's
Mixed Signal Lecture Series

DSP-Based Testing – Fundamentals 16
Moving Average

Verigy Japan
August 2009

Preface to the Series

ADC and DAC are the most typical mixed signal devices. In mixed signal testing, analog stimulus signal is generated by an arbitrary waveform generator (AWG) which employs a D/A converter inside, and analog signal is measured by a digitizer or a sampler which employs an A/D converter inside. The stimulus signal is created with mathematical method, and the measured signal is processed with mathematical method, extracting various parameters. It is based on digital signal processing (DSP) so that our test methodologies are often called DSP-based testing.

Test/application engineers in the mixed signal field should have thorough knowledge about DSP-based testing. FFT (Fast Fourier Transform) is the most powerful tool here. This corner will deliver a series of fundamental knowledge of DSP-based testing, especially FFT and its related topics. It will help test/application engineers comprehend what the DSP-based testing is and assorted techniques.

Editor's Note

For other articles in this series, please visit the Verigy web site at www.verigy.com/go/gosemi.

Moving Average

When you measure a signal and it looks noisy, you may want to make it smooth with the moving average. Moving average operation is actually very similar to FIR convolution so that it behaves like a low pass filter. In this article, the frequency characteristics of the moving average and related comb filters are discussed. Different taste of filtering from the FIR convolution is discussed here.

Moving Average API

The V93000 SOC Test System provides a DSP API for the moving average. It is DSP_MOVAVG(). This API performs summing of a certain period of data which is divided by the summing length. Figure 1 depicts how the 24-point (*N*) input data stream is averaged with the window length 8 (*Length*). Each 8-point segment is accumulated and divided by 8 step by step. As shown in the figure, the valid number of data is reduced as 17 (*N*-*Length*+1) eventually. So applying moving average, you need to capture data longer by "*Length*" points than you need at the end.

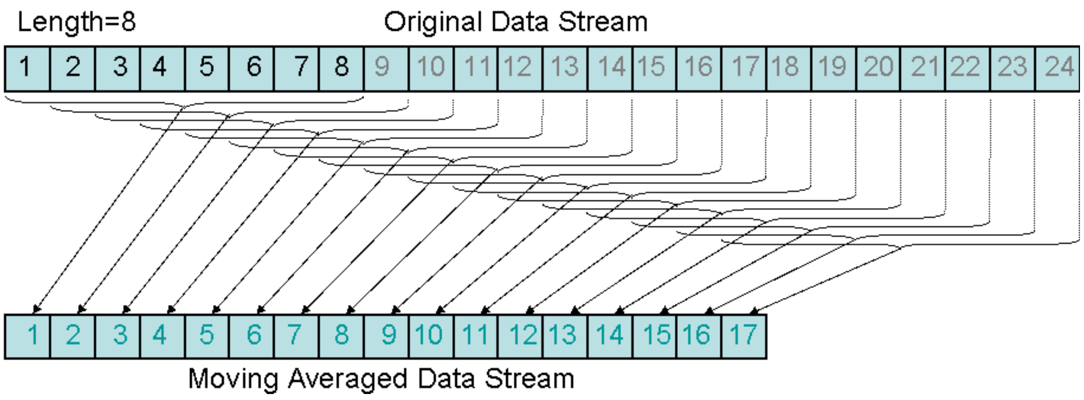

Figure 1: Moving Average

Figure 2(a) shows the block diagram of the operation. It is an 8-word shift register. Input data pushes in the register, and entire data in the register are added and divided by 8 every time.

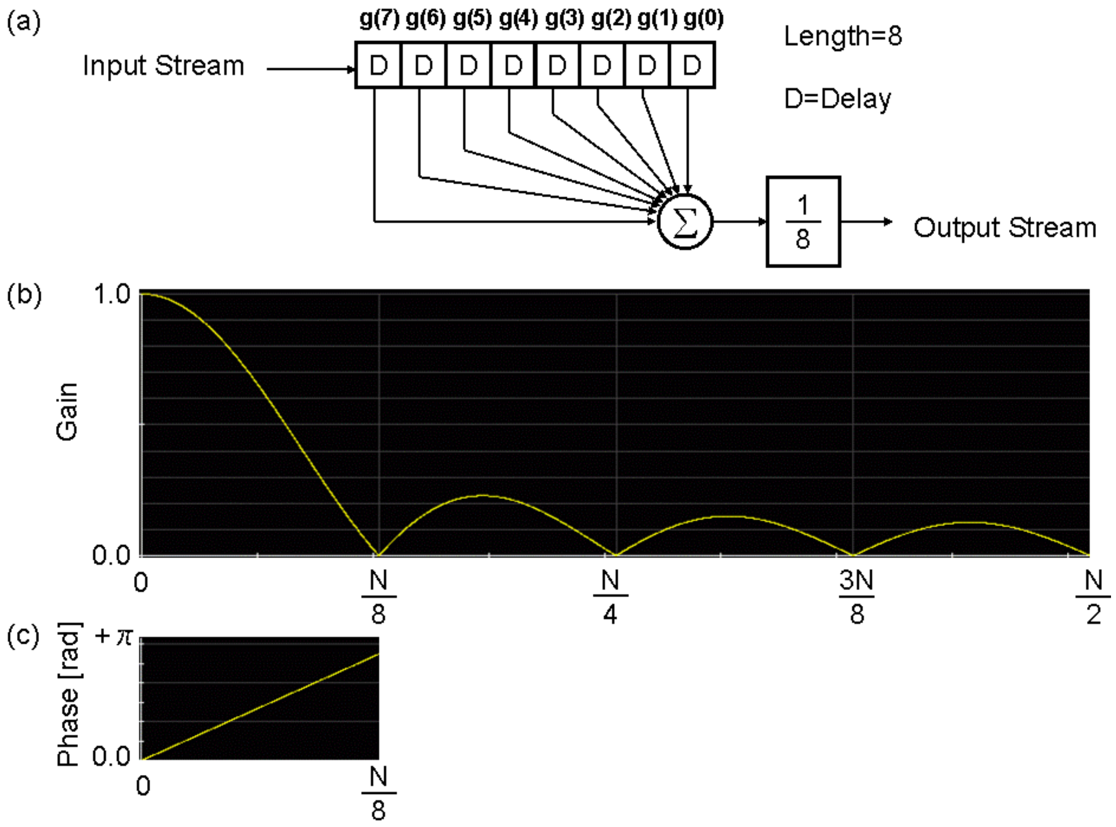

Figure 2: Frequency Response of Moving Average

Figures 2(b) and (c) show the frequency response of Length-8 moving average. Actually the curve is the same as typical SINC or sin(x)/x curve.

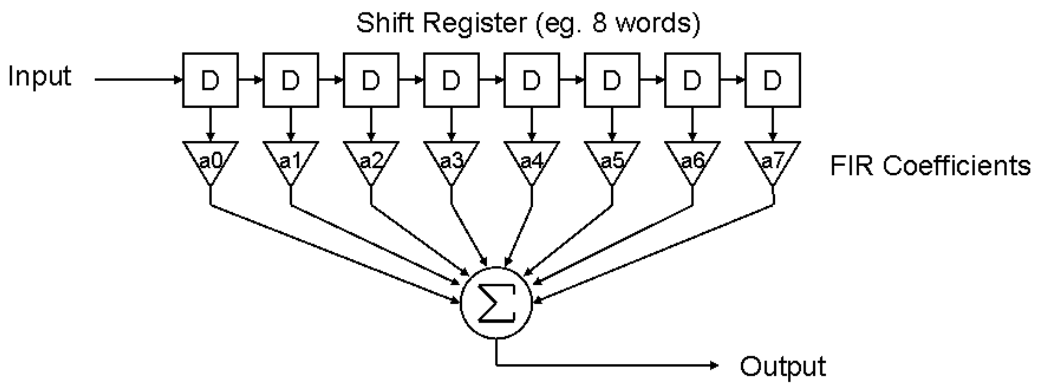

Figure 3: Block Diagram of FIR Convolution

FIR convolution was discussed in a previous Newsletter article. Its block diagram can be described as Figure 3, and it may be called a transversal filter. It looks very similar to Figure 2(a). You can see moving average is a special case of FIR convolution. The entire coefficients are 1/8 here.

Simple Digital Filters

Digital filters can be realized with other methods than FIR convolution. Figure 4(a) and (b) depict simple filtering operation. (c) is an integrator operation.

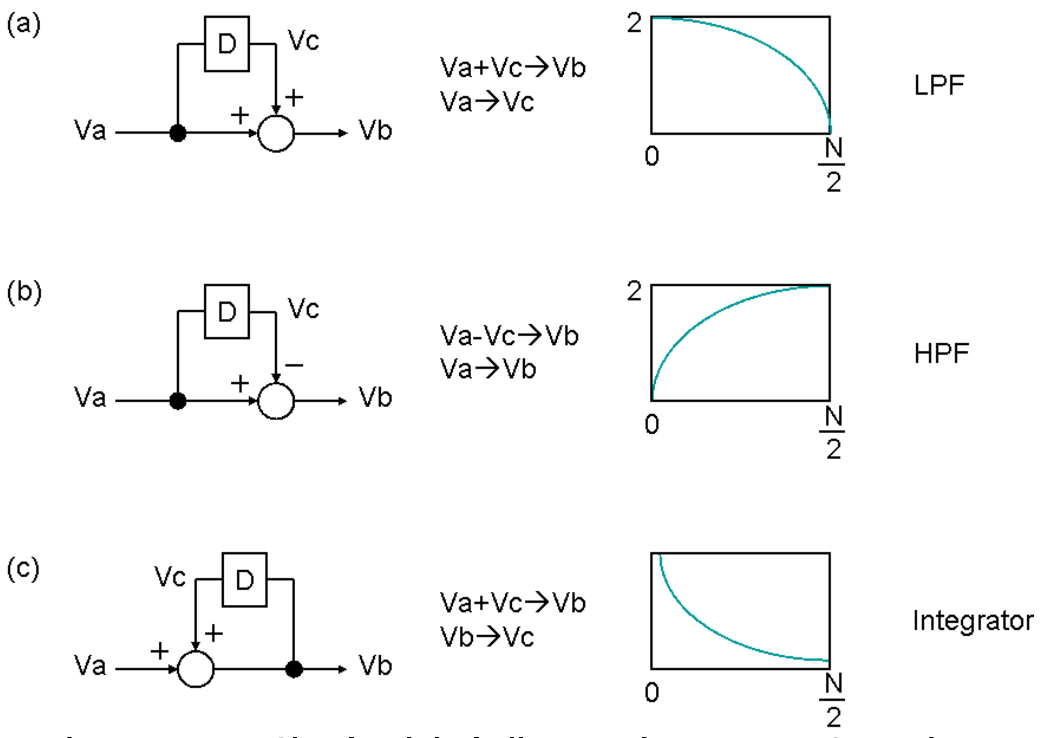

Figure 4: Simple Digital Filters and Integrator Operations

Figures 4(a) and (b) are straightforward. They can be coded as Lists 1 and 2. The difference is only the sign at Line 12. The initialization at Line 9 is based on an assumption that the data *dVdgt*[] repeats infinitely.

```
01:    INT       i, Ndgt;
02:    DOUBLE     dVa,dVb,dVc;
03:    ARRAY_D   dVdgt,dWave;
04:    . . .
05:    dVdgt=DGT("AOUT").getWaveform();  // Original Signal
06:    Ndgt=dVdgt.size();
07:
08:    dWave.resize(Ndgt);   // Output Container
09:    dVc=dVdgt[Ndgt-1];    // Initialize
10:    for (i=0;i<Ndgt;i++) {
11:          dVa=dVdgt[i];  // Input
12:          dVb=dVa+dVc;   // Addition
13:          dWave[i]=dVb;  // Output
14:
15:          dVc=dVa;       // 1 delay
16:    }
17:
```

List 1: LPF

```
01:    INT       i, Ndgt;
02:    DOUBLE     dVa,dVb,dVc;
03:    ARRAY_D   dVdgt,dWave;
04:    . . .
05:    dVdgt=DGT("AOUT").getWaveform();  // Original Signal
06:    Ndgt=dVdgt.size();
07:
08:    dWave.resize(Ndgt);   // Output Container
09:    dVc=dVdgt[Ndgt-1];    // Initialize
10:    for (i=0;i<Ndgt;i++) {
11:          dVa=dVdgt[i];  // Input
12:          dVb=dVa-dVc;   // Subtraction
13:          dWave[i]=dVb;  // Output
14:
15:          dVc=dVa;       // 1 delay
16:    }
17:
```

List 2: HPF

Figure 3(c) looks similar to (a) but it behaves totally different from (a). *Vc* is one clock delay of *Vb* that is the output signal. It is a feedback loop, and it behaves an integrator. The code is as follows. Line 15 is different from List 1.

Figure 5 demonstrates example input and output waveforms showing how the integrator works.

```
INT         i, Ndgt;
DOUBLE      dVa,dVb,dVc;
ARRAY_D     dVdgt,dWave;

dVdgt=DGT("AOUT").getWaveform();
Ndgt=dVdgt.size();

dWave.resize(Ndgt);     // Output Container
dVc=0.0;                // Initialize
for (i=0;i<Ndgt;i++) {
        dVa=dVdgt[i];   // Input
        dVb=dVa+dVc;    // Addition
        dWave[i]=dVb;   // Output

        dVc=dVb;        // 1 delay of Output
}

```

List 3: Integrator

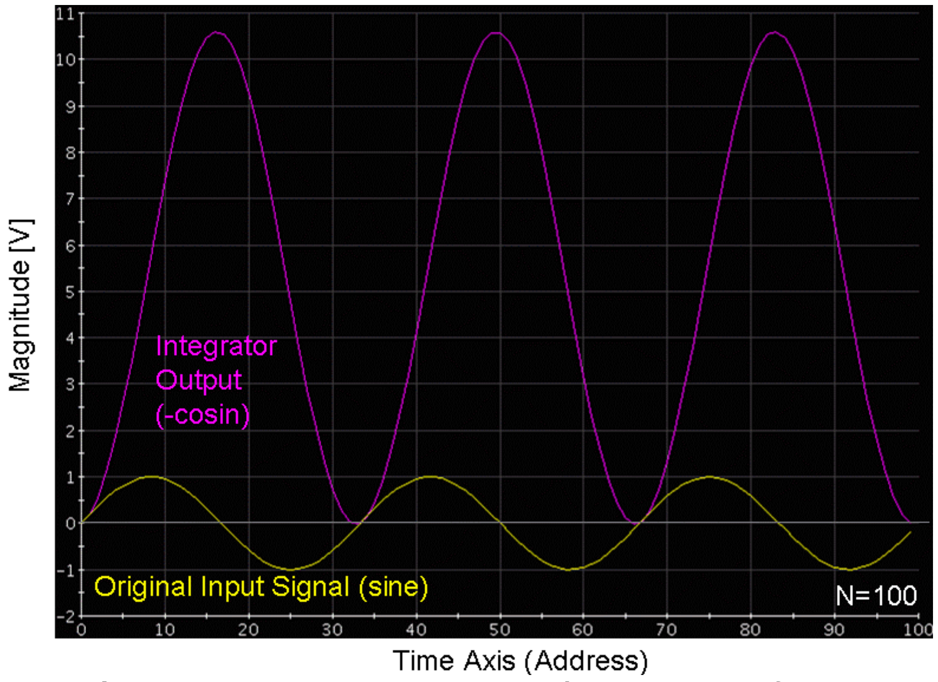

Figure 5: Integrator Input/Output Waveforms

Comb Filters

In the filter configurations in Figure 4(a) and (b), the number of delays makes their frequency response different. Figures 5 and 6 show the frequency responses versus the number of delays. Actually they are not simple LPF or HPF but they are comb filters, which contain periodic null points in the frequency domain. Therefore they can be used to suppress harmonics based interference.

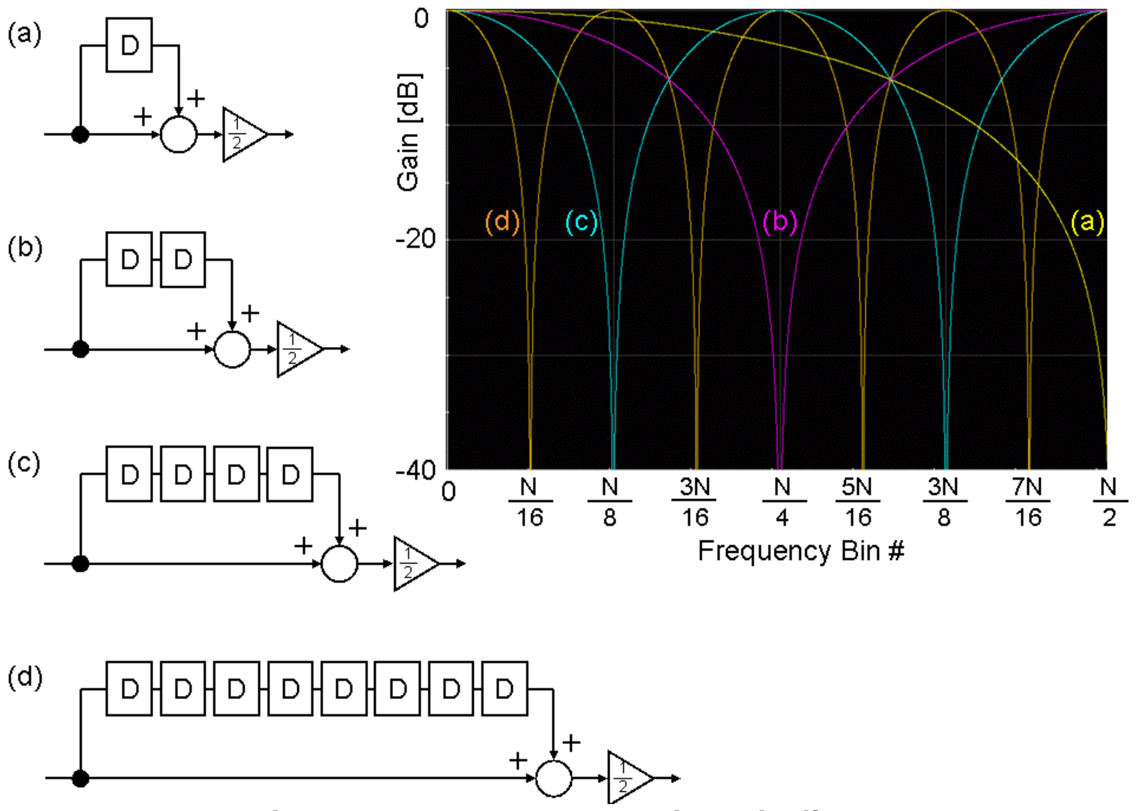

Figure 6: Low Pass and Comb Filters

List 4 shows the calculation routine of Figure 6(d). The shift register is realized with a kind of ring buffer. The input data stream is not shifted in the buffer, but the index of the buffer is incremented every time. The oldest data in the buffer is replaced with the newest data every time. The ring buffer is initialized with zeros so that the first *Nx* points of the filtered output are garbage. To fix and recover the first portion of the output waveform, Lines 22 through 29 are added. This is based on an assumption that the data *dVdgt*[] repeats infinitely.

```
01:    INT       i, Ndgt,Nx;
02:    DOUBLE    dVa,dVb,dVc;
03:    ARRAY_D   dVdgt,dWave,dBuf;
04:    ...
05:    dVdgt=DGT("AOUT").getWaveform();
06:    Ndgt=dVdgt.size();
07:
08:    Nx=8;                  // Length of Register (Nx<Ndgt)
09:    dBuf.resize(Nx);       // Shift Register
10:    dBuf.init(0.0);        // Initialize
11:
12:    dWave.resize(Ndgt);    // Output Container
13:    k=0;
14:    for (i=0;i<Ndgt;i++) {
15:          dVa=dVdgt[i];        // Input
16:          dVb=dVa+dBuf[k];     // Addition
17:          dWave[i]=dVb/2.0;    // Scaling and Output
18:
19:          dBuf[k]=dVa;
20:          k++;  if (k==Nx) k=0;   // Shift
21:    }
```

```
22:    for (i=0;i<Nx;i++) {        // Fix the start area
23:            dVa=dVdgt[i];       // Input
24:            dVb=dVa+dBuf[k];    // Addition
25:            dWave[i]=dVb/2.0;   // Scaling and Output
26:
27:            dBuf[k]=dVa;
28:            k++;  if (k==Nx) k=0;  // Shift
29:    }
30:
```
List 4: Low Pass and Comb Filter

The high pass and comb filters in Figure 7 can be calculated very similar to List 4. The only difference is the sign in the lines 16 and 24.

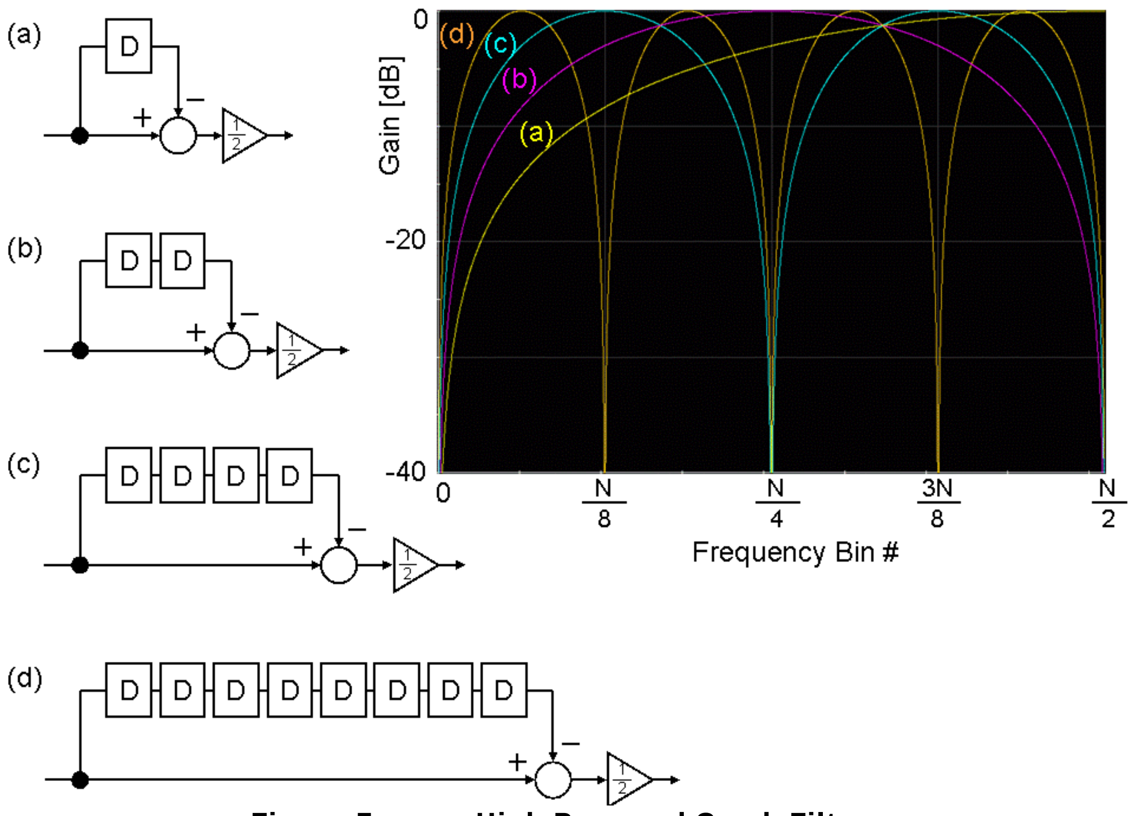

Figure 7: High Pass and Comb Filters

Cascaded Integrator and Comb Filter

Figure 8(a) depicts the combination of 8-word shift register high pass comb filter and the integrator. The total frequency response becomes the combination of the response of comb filer and the integrator. It is shown in Figure 8(b) and (c). Actually the frequency response is exactly the same as Figure 2(b) and (c). The phase (c) does not look match with each other, but it comes from the array index. In DSP_MOVAVG(), the garbage in the starting area is already trimmed so that the output data length is shorter than the input data length by the window size of averaging. Since the time axis base is shifted in DSP_MOVAVG(), Figure 2(c) shows phase advance compared to Figure 8(c). Anyway, the moving average in Figure 2(a) can be performed as Figure 8(a) too. This is a cascaded combination of the integrator and the comb filter so that this type of low pass filter may be called a cascaded integrator comb or CIC filter.

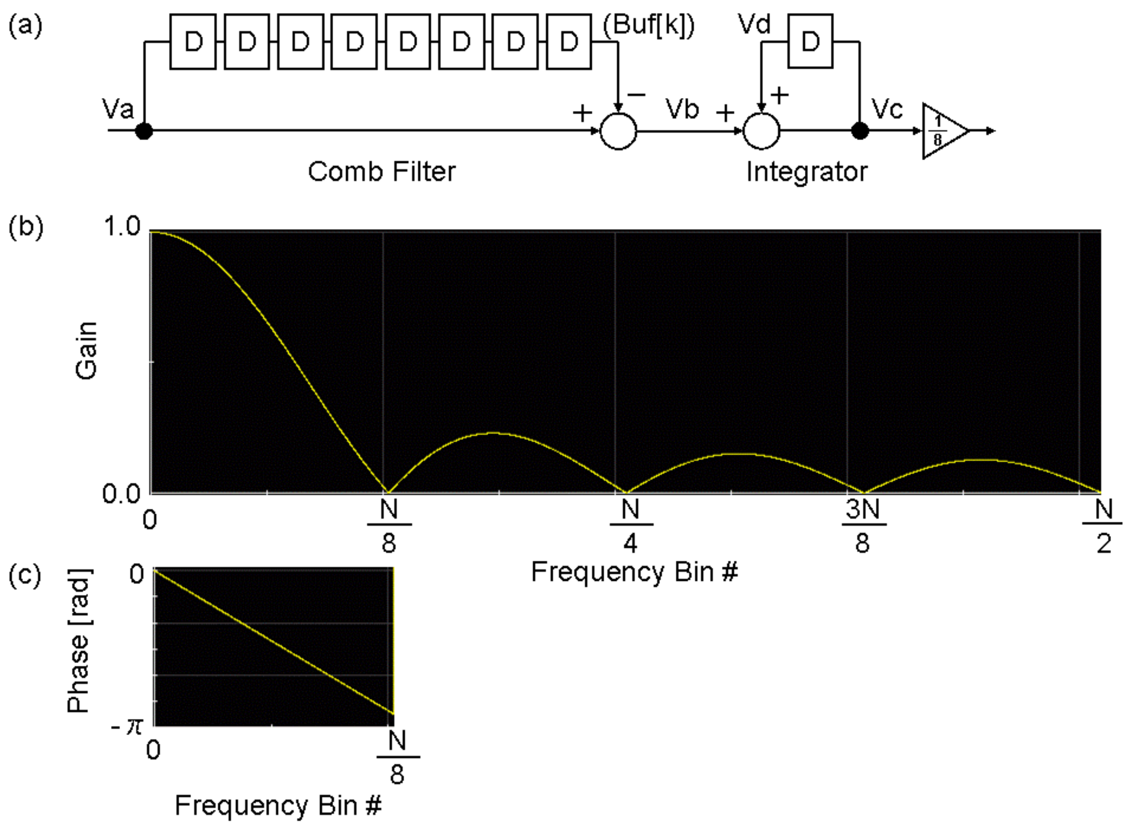

Figure 8: Combined Integrator and Comb Filter

```
INT         i, Ndgt,Nx;
DOUBLE      dVa,dVb,dVc,dVd;
ARRAY_D     dVdgt,dWave,dBuf;
 ...
dVdgt=DGT("AOUT").getWaveform(); // Original Signal
Ndgt=dVdgt.size();

Nx=8;                            // Length of Register
dBuf.resize(Nx);                 // Shift Register
dBuf.init(0.0);                  // Initialize
dVd=0.0;                         // Initialize

dWave.resize(Ndgt);              // Output Container
k=0;
for (i=0;i<Ndgt;i++) {
        dVa=dVdgt[i];           // Input
        dVb=dVa-dBuf[k];        // Subtraction
        dVc=dVb+dVd;            // Integrator
        dWave[i]=dVc/Nx;        // Scaling and Output

        dBuf[k]=dVa;            // Push Shift Register
        k++; if (k==Nx) k=0;    // Shift
        dVd=dVc;                // Integrator Delay
}
```

```
25:     for (i=0;i<Ndgt;i++) {
26:             dVa=dVdgt[i];           // Input
27:             dVb=dVa-dBuf[k];        // Subtraction
28:             dVc=dVb+dVd;            // Integrator
29:             dWave[i]=dVc/Nx;        // Scaling and Output
30:
31:             dBuf[k]=dVa;            // Push Shift Register
32:             k++; if (k==Nx) k=0;    // Shift
33:             dVd=dVc;                // Integrator Delay
34:     }
```

List 4: CIC

The source code is listed in List 4. Lines 15 through 24 are exactly the same as Lines 25 through 34. This redundancy is to fix the initial garbage. It is the same idea to Lines 22 through 30 in List 4. So you may want to adjust the number of repetition for through-put purpose. Lines 18 and 28 are the integrator. The number of delay or shift register length in the comb filter rules the pass band width and rejection characteristics as Figure 9 shows.

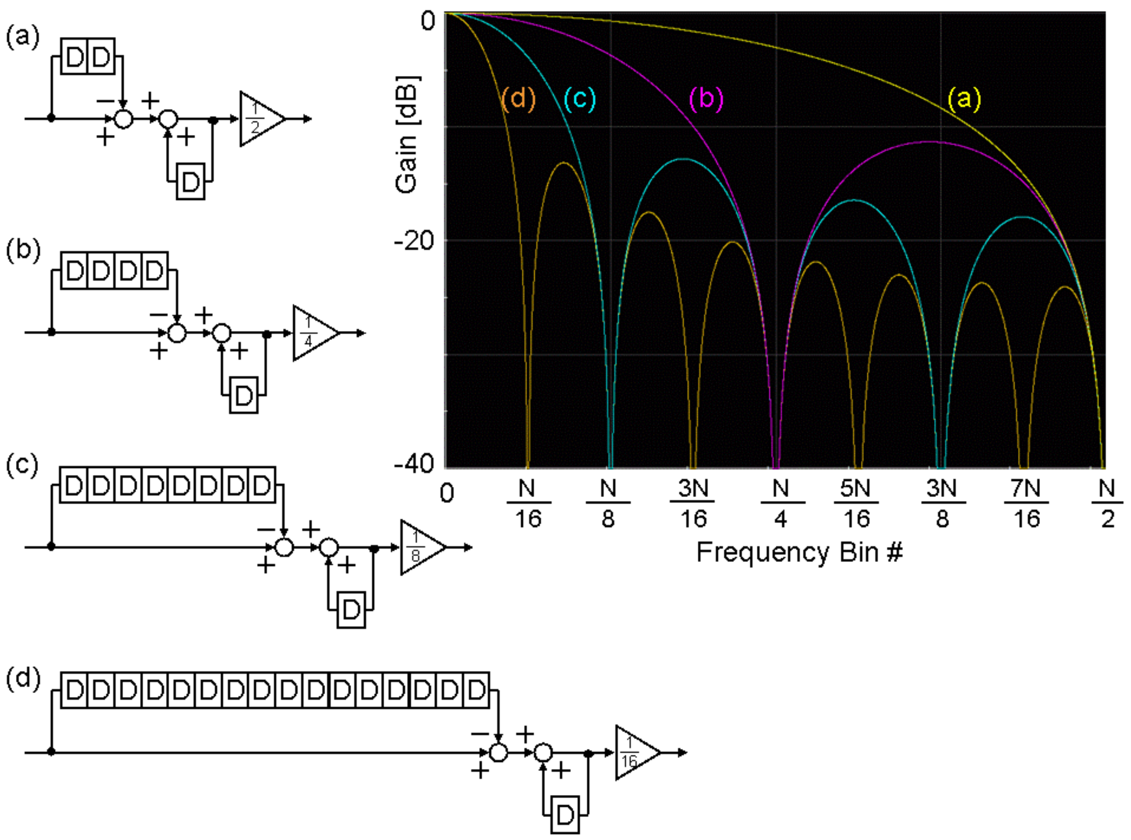

Figure 9: CIC Frequency Responses

Chroma Filter

With combining the various comb filters, you can create a specific frequency response. Figure 10(a) is an example. There are 6 comb filter sections marked [b] through [g]. Figures 10(b) through (g) correspond each frequency response. Figure 10(h) shows the total frequency response with the scaling. It is a band pass filter whose pass-band is located at around *Fs*/4.

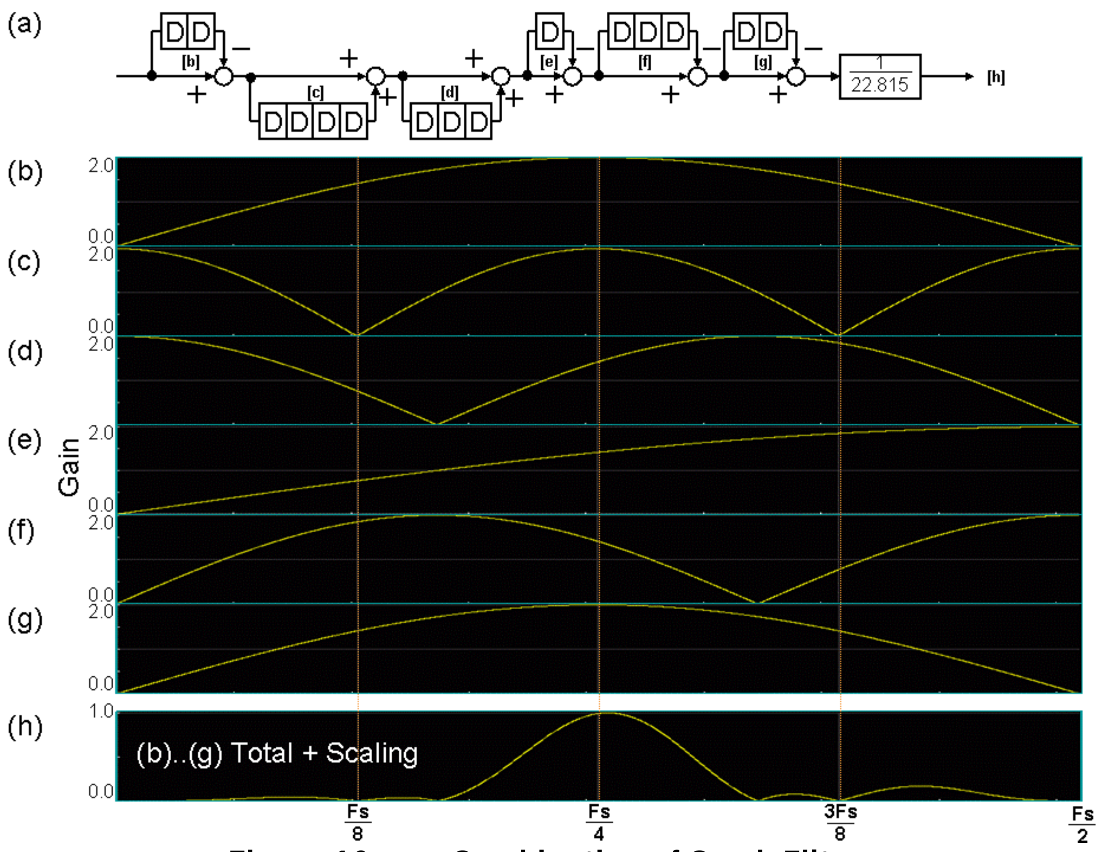

Figure 10: **Combination of Comb Filters**

The video composite signal of conventional analog color TV contains luma signal, chroma signal and signaling pulses, which are separated by appropriate filters. Figure 11(a) shows a simulated video composite signal providing for video device testing. The chroma signal is 3.58MHz in NTSC system, and the test signal is digitized with 4 times sampling rate. Therefore 4 points of data contain a single cycle of chroma signal, meaning the chroma energy is located at 1/4. When the signal of Figure 11(a) goes through the routine of Figure 10(a), the output looks as Figure 11(b), where the chroma signal is successfully extracted.

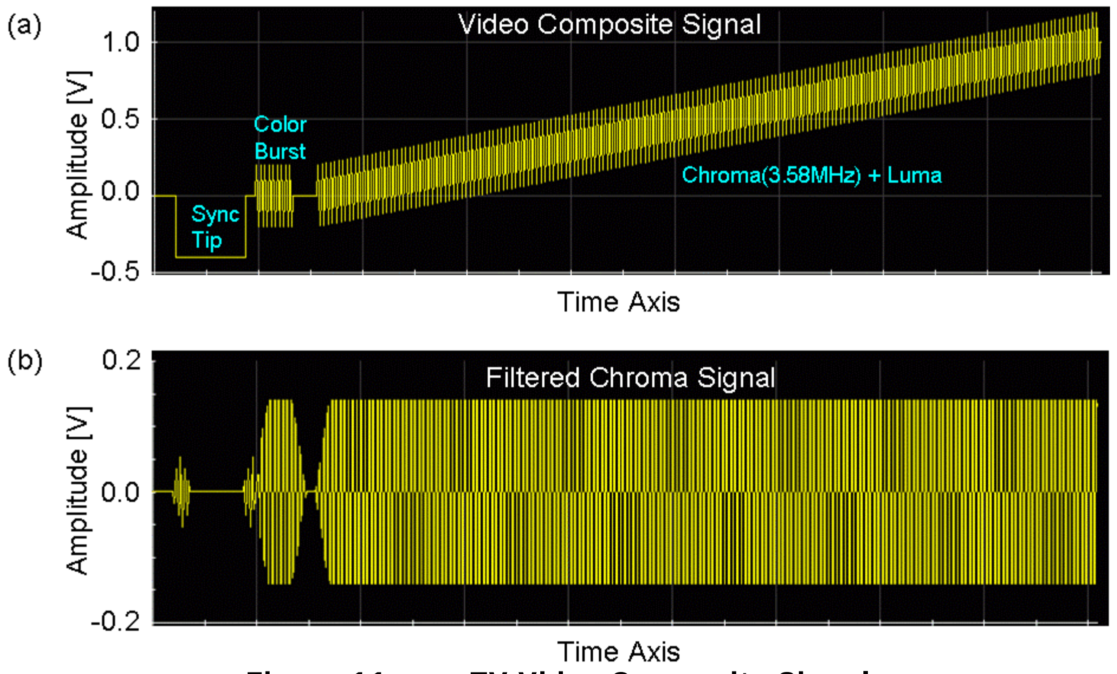

Figure 11: **TV Video Composite Signal**

In this issue, another taste of digital filtering has been discussed.