Hideo Okawara's
Mixed Signal Lecture Series

DSP-Based Testing – Fundamentals 36
Filter Response Simulation by F-matrix

Verigy Japan
May 2011

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) that employs a D/A converter inside, and analog signal is measured by a digitizer or a sampler that 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 article 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 website at www.verigy.com/go/gosemi.

Preface

The previous article discussed the basics of F-matrix. This issue takes up how it is used in our SmarTest environment. This is an exercise of the F-matrix idea. The example circuit is a low pass filter used in the VHF AWG in the MB-AV8.

Model Circuit

Figure 1 illustrates the circuit diagram of the 150MHz LPF that is integrated in the VHF AWG in the MB-AV8. It is a 7-component Butterworth LPF designed for the 50Ω characteristic impedance.

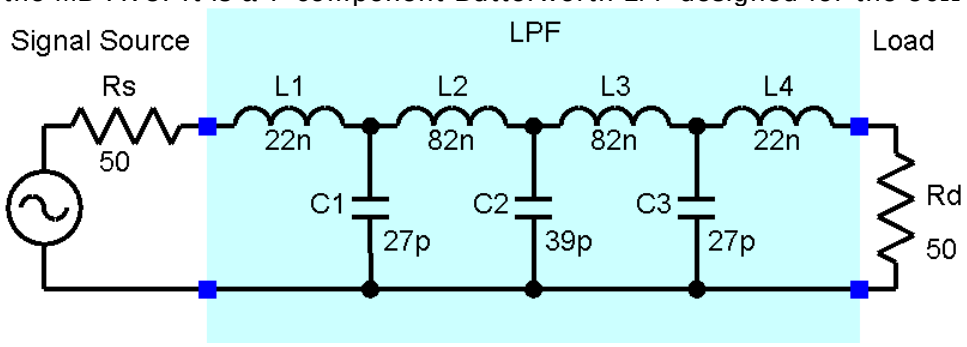

Figure 1: LPF Circuit Diagram

F-matrix

The F-matrices discussed in the previous newsletter article[1] can simulate the performance of the filter. Figure 2 illustrates the whole circuit expressed with cascaded three F-matrices of F1, F2 and F3. They should be combined into a single matrix F0. Then you can estimate the frequency response as 1/A of F0, the input impedance as A/C of F0, and so forth.

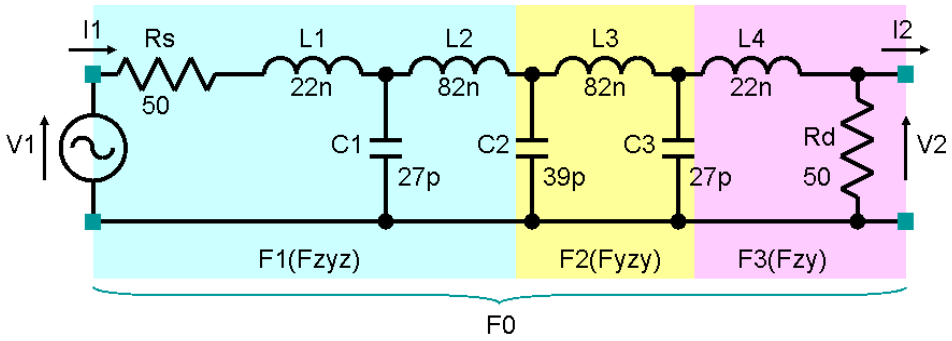

Figure 2: F-matrix Assignment

Frequency Response

List 1 illustrates the part of F-matrix calculation. In terms of the definitions and coding of *Fmatrix*, *Fzyz, Fyzy*, *Fzy*, and so on, refer to the previous newsletter. Lines from 40 to 53 are the core part of the calculation. Multiplication and division of COMPLEX parameters and F-matrices can be

[1] Mixed Signal Lecture Series – Fundamentals 35 F-matrix Simulation

operated with very simple manners. The COMPLEX array “CGain[]” contains the frequency response of the LPF. It is converted into the real number arrays of gain and phase at Line 55.

```
11:
12:    INT            i;
13:    DOUBLE         w;
14:    DOUBLE         dFreq;
15:    INT            N=1024;
16:    INT            Nsp=N/2;
17:    DOUBLE         dFs=1024.0 MHz;
18:    DOUBLE         dFresln=dFs/N;
19:
20:    DOUBLE         Rs=50.0;
21:    DOUBLE         C1=27.0 pF;
22:    DOUBLE         C2=39.0 pF;
23:    DOUBLE         C3=27.0 pF;
24:    DOUBLE         L1=22.0 nH;  // #define nH *1.0e-9
25:    DOUBLE         L2=82.0 nH;
26:    DOUBLE         L3=82.0 nH;
27:    DOUBLE         L4=22.0 nH;
28:    DOUBLE         Rd=50.0;
29:    Fmatrix        F0,F1,F2,F3,F4;
30:    ARRAY_COMPLEX  CGain;
31:    ARRAY_D        dGain,dPhase;
32:    COMPLEX        One(1.0,0.0);

40:    CGain.resize(Nsp);
41:    for (i=0;i<Nsp;i++) {
42:       dFreq=dFresln*i;
43:       w=2.0*M_PI*dFreq;
44:
45:       F1=Fzyz(COMPLEX(Rs,w*L1),COMPLEX(0.0,w*C1),COMPLEX(0.0,w*L2));
46:       F2=Fyzy(COMPLEX(0.0,w*C2),COMPLEX(0.0,w*L3),COMPLEX(0.0,w*C4));
47:       F3=Fzy(COMPLEX(0.0,w*L4),COMPLEX((1.0/Rd),0.0));
48:
49:       F4=F2*F3;
50:       F0=F1*F4;              //   F0=F1*F2*F3
51:
52:       CGain[i]=One/F0.a;     //   1/A of F0
53:    }
54:    DSP_RECT_POL(CGain,dGain,dPhase);
55:    DSP_LOG10_VEC(dGain,dGain);        //
56:    DSP_MUL_SCL(20.0,dGain,dGain);     // 20*log10(dGain[i])
57:
```

List 1: Coding of Frequency Response Calculation

Figure 3 shows the frequency response of the gain and phase calculated based on List 1.

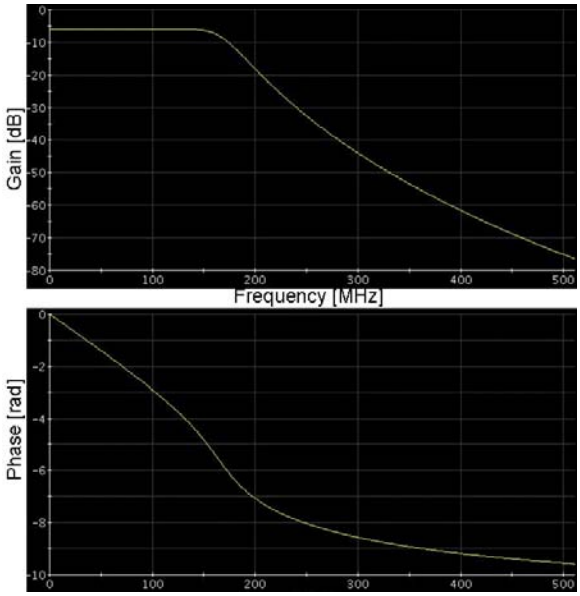

Figure 3: Frequency Response

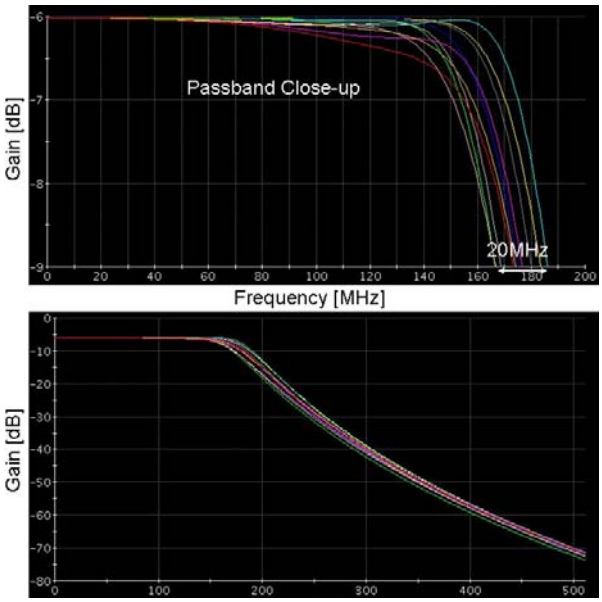

Figure 4: 10% Tolerance Simulation

LC components have their tolerance specification. The LC values listed in Figure 1 are the nominal values. In reality, those component values would vary based on their tolerance specification. Figure 4 illustrates the 10 trials of simulation results overlaid. Each one of the components is randomly modified within 90% to 110% of the nominal value. This is a kind of Monte Carlo simulation. The close-up view of the passband shows that the -3dB bandwidth would vary 20MHz range for only 10 trials. This is just an example simulation. The actual LPF response in the MB-AV8 is strictly controlled by using components with much tighter specification than 10%.

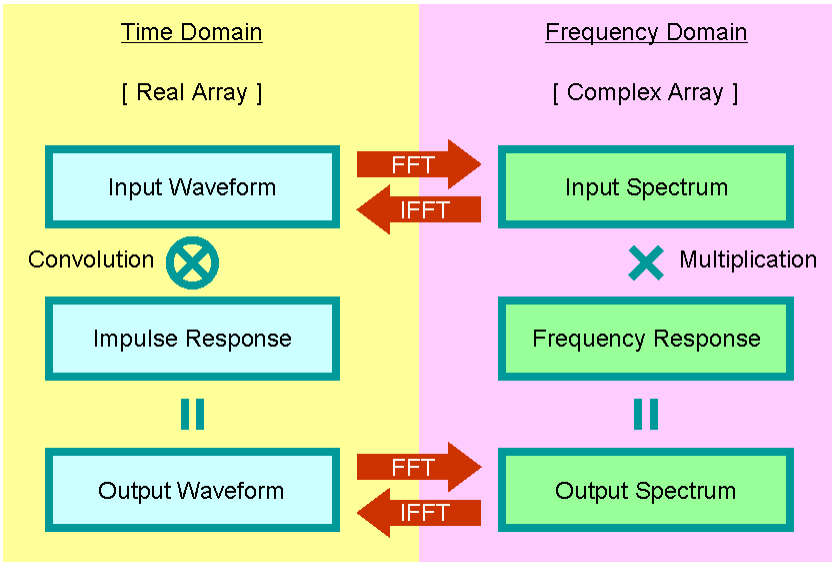

Figure 5: Filtering in Time/Frequency Domains

Filtering in Frequency Domain

When a square wave goes through this LPF, how would it be rolled off? It can be simulated with multiplying the complex spectrum to the filter response array. Figure 5 illustrates about filtering operations. Filtering is realized by multiplication of the spectrum and the gain response in the frequency domain. List 2 illustrates the procedure. “dWave1[]” is the input signal waveform that is PRBS bit stream. “dWave2[]” is the output signal waveform that looks rolled off by the filter. Figure 6 shows the simulation result. The yellow lines show the input waveform and spectrum, and the red lines show the output waveform and spectrum. The spectrum shows that the output spectrum is weighted with the LPF frequency response against the input spectrum.

```
80:
81:    ARRAY_D       dWave1,dWave2;              // Waveform Container
82:    ARRAY_D       dSp1,dSp2;                  // Spectrum Container
83:    ARRAY_COMPLEX CSp1,CSp2;
84:
85:    DSP_FFT(dWave1,CSp1,RECT);                // Spectrum (Half Plane)
86:    DSP_MUL_VEC(CSp1,CGain,CSp2);             // Multiplication
87:    CSp2.resize(N);                           // Full Plane Spectrum
88:    for (i=1;i<Nsp;i++) CSp2[N-i]=Conjugate(CSp2[i]); //***Conjugate***
89:    CSp2[0].real()=2.0*CSp2[0].real();        // DC bin x2
90:    CSp2[Nsp]=Zero();
91:    DSP_IFFT(CSp2,CWave2);                    // Freq.->Time Domain
92:    dWave2=CWave2.getReal();                  // Real Part Extraction
93:
94:    DSP_SPECTRUM(dWave1,dSp1,DB,1.0,RECT,0); // Input Signal Spectrum
95:    DSP_SPECTRUM(dWave2,dSp2,DB,1.0,RECT,0); // Output Signal Spectrum
96:
```

List 2: Filtering in Frequency Domain

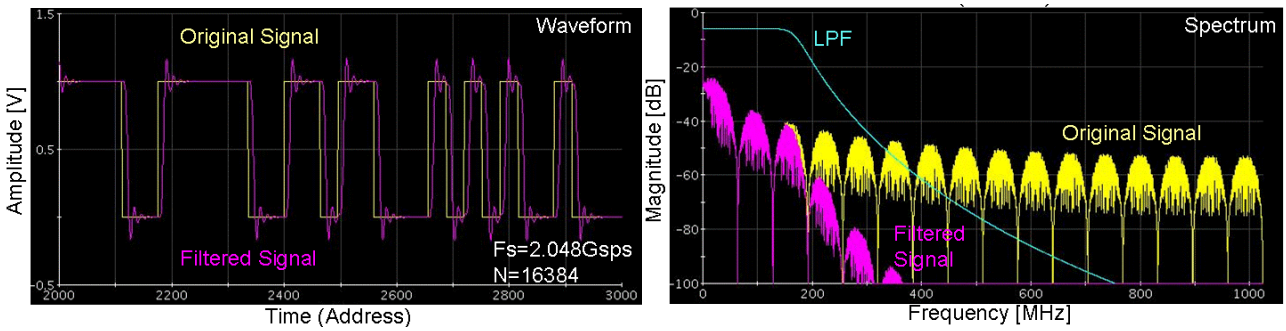

Figure 6: Multiplication Result

Filtering in Time Domain

Filtering in the time domain is realized by convolution of the impulse response of the filter and the input signal waveform. The finite impulse response (FIR) of the filter can be generated by using the frequency response of the COMPLEX gain array “CGain[]” derived in List 1.[2]

[2] Mixed Signal Lecture Series – Fundamental 14 FIR Filter

```
11:   dFs=2048.0 MHz;
12:   Nfir=256;          // FIR Length (Time Domain)
13:   Nsp=Nfir/2;
14:   dFresln=dFs/Nfir;
20:   CGain.resize(Nsp);
21:   for (i=0;i<Nsp;i++) {
22:      dFreq=i*dFresln;
23:      w=2.0*M_PI*dFreq;
24:      F1=Fzyz(COMPLEX(Rs,w*L1),COMPLEX(0.0,w*C1),COMPLEX(0.0,w*L2));
25:      F2=Fyzy(COMPLEX(0.0,w*C2),COMPLEX(0.0,w*L3),COMPLEX(0.0,w*C3));
26:      F3=Fzy(COMPLEX(0.0,w*L4),COMPLEX((1.0/Rd),0.0));
27:      F4=F2*F3;
28:      F0=F1*F4;
29:      CGain[i]=One/F0.a;
30:   }
31:
32:   ARRAY_COMPLEX CFIR;
33:   ARRAY_D       dFIR,dFIR1;                  // FIR Container
34:
35:   DSP_RECT_POL(CGain,dGain,dPhase);          // Gain Response
36:   DSP_MUL_SCL(2.0,dGain,dGain);              // 6dB Compensation
37:   DSP_POL_RECT(dGain,dPhase,CGain);          // Complex Response Curve
38:
39:   CGain.resize(Nfir);
40:   for (i=1;i<Nsp;i++) CGain[Nfir-i]=Conjugate(CGain[i]);
41:   CGain[Nsp]=Zero;
42:
43:   DSP_IFFT(CGain,CFIR);                      // Spectrum -> Waveform
44:   dFIR=CFIR.getReal();                       // Impulse Response
45:   DSP_MUL_SCL(1.0/(DOUBLE)Nfir,dFIR,dFIR);   // Normalization
46:
50:   DSP_CONVOL(dFIR,dWave1,dWave2,ON);  // Filtering in Time Domain
```

List 3: Coding for FIR Creation and Convolution

List 3 illustrates the actual procedure to generate the FIR. The length of the FIR is decided as 256 at Line 12. Lines 20 through 30 are exactly the same procedure as Lines 40 to 53 in List 1. The difference is the length of the data array. The gain response “CGain[]” for 256-length FIR is the frequency domain array so that it is converted into the time domain array by using the IFFT. The original response has 6dB loss because of the source impedance Rs. So the loss is compensated at Line 36 by multiplying 2. The derived frequency response is illustrated as Figure 7. This is the half plane characteristics so that it is extended to the full plane by generating the complex conjugate symmetrically at Line 40. The full-plane “CGain[]” is processed with the IFFT and the FIR curve is derived at Line 44, which is scaled or normalized at Line 45. The FIR curve is illustrated in Figure 8. Finally the FIR is convoluted to the input signal waveform at Line 50, and the filtering in the time domain is completed.

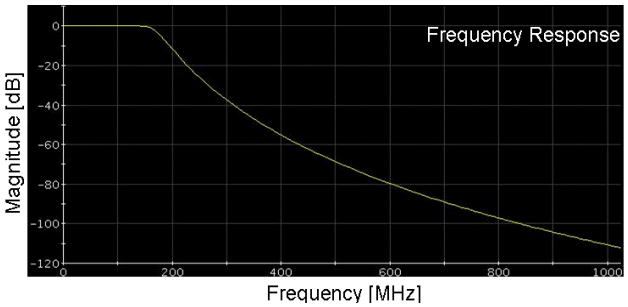

Figure 7: Frequency Response (Gain)

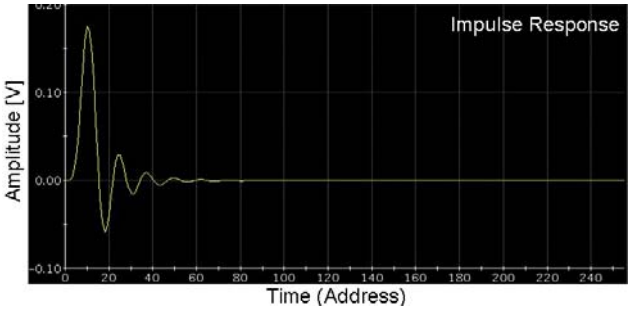

Figure 8: FIR (Normalized)

Figure 9 shows the input and output waveforms and spectra filtered in the time domain. The result is the same as Figure 6.

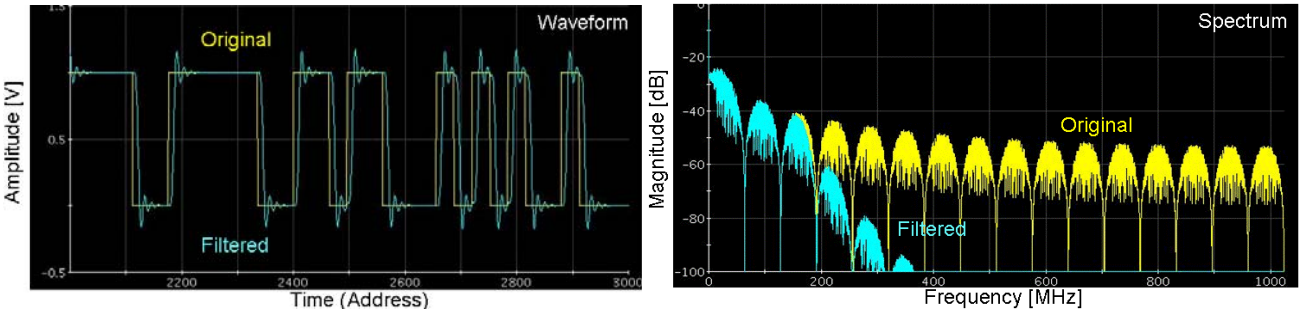

Figure 9: Convolution Result: Waveform and Spectrum