Hideo Okawara's
Mixed Signal Lecture Series

DSP-Based Testing – Fundamentals 35
F-matrix Simulation

Verigy Japan
April 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) 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 a mathematical method, and the measured signal is processed with a 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

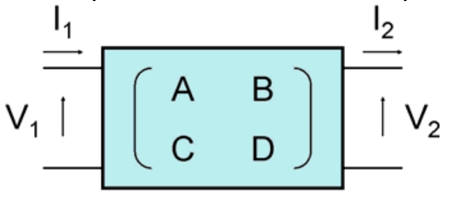
For other articles in this series, please visit the Verigy website at www.verigy.com/go/gosemi.

Preface

When you design a DUT board, you may want to analyze the characteristics and performance of transmission lines, switches, filters or their combination circuits. If you had simulation tools such as Spice or MATLAB available in your environment, you could use them for analysis. However, even if you have nothing like those tools, you could do considerable analysis utilizing our SmarTest environment. You might have learned about F-matrix in the lectures of electric circuit theory in your school days. It is very traditional and conventional theory. Many of you may have forgotten it. However, we could utilize it for our simple circuit simulations using our SmarTest APIs and arithmetic instructions. This scheme could be embedded in our application if necessary. From this month, we will discuss the F-matrix applications. Let's begin with the basics of the F-matrix.

F-matrix

Figure 1 illustrates a four-terminal network or two-port network of an electric circuit, which is linear time-invariant and does not contain any source inside. The input voltage V_1 and current I_1 can be related to the output voltage V_2 and current I_2 with the ABCD parameters[1] in the F-matrix as illustrated. The F-matrix stands for four-terminal matrix. The input/output voltages, currents and ABCD parameters are all complex numbers in this article and the simulation program.



$$\begin{cases} V_1 = A \cdot V_2 + B \cdot I_2 \\ I_1 = C \cdot V_2 + D \cdot I_2 \end{cases}$$

Figure 1: F-matrix

```
struct Fmatrix {

 COMPLEX a;
 COMPLEX b;
 COMPLEX c;
 COMPLEX d;

 Fmatrix operator*(Fmatrix &);
};
```

List 1: F-matrix Definition

First of all, you have to define the F-matrix type. Since an F-matrix contains complex number ABCD parameters, it is realized as the traditional **struct** definition as List 1 which contains the operator entry for the specific multiplication of F-matrices as well. This operator entry allows you to write F-matrices multiplication very easy with using the mark "*" as regular arithmetic. Now you can declare F-matrix variables and refer ABCD components as List 2 shows.

When the output port of the network is open, the output current I_2 is zero so that the VI equations can be simplified as Figure 2 illustrates. Consequently the voltage gain of the network V_2/V_1 and the input impedance V_1/I_1 can be described as 1/A and A/C respectively. These calculations can be programmed as List 2 shows. Since the components in Fx are complex numbers, the four basic operations of arithmetic (+-*/) are performed in the complex number mode in our C++ environment.

[1] Alias: Transmission Parameters

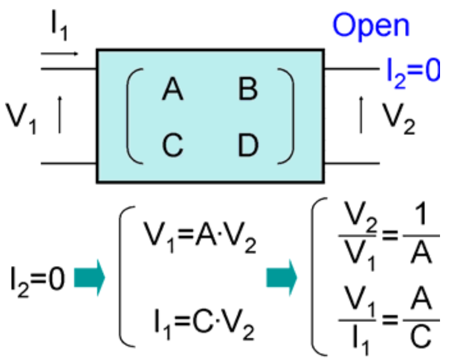


Figure 2: Output Open (I_2=0)

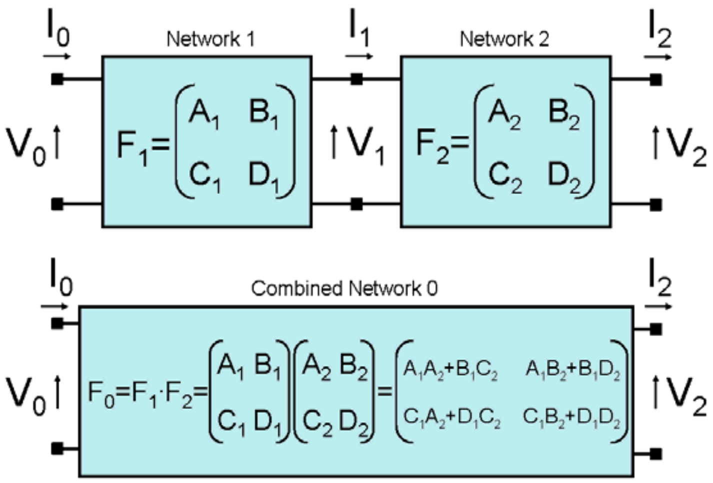
```
Fmatrix    Fx;
COMPLEX    One(1.0,0.0);      //  1+j0
COMPLEX    Zero(0.0,0.0);     //  0+j0

COMPLEX    Gain=One/Fx.a;     //  1/A
COMPLEX    Zin=Fx.a/Fx.c;     //  A/C
```

List 2: Complex Number Arithmetic

Cascaded Networks

When two networks are cascaded, they can be combined into a single network who's F-matrix can be calculated by multiplying the two F-matrices F_1 and F_2 together as Figure 3 illustrates:



```
Fmatrix Fmatrix::operator*(
        Fmatrix & F2
)
{
 Fmatrix F0;

 F0.a= a*F2.a + b*F2.c;
 F0.b= a*F2.b + b*F2.d;
 F0.c= c*F2.a + d*F2.c;
 F0.d= c*F2.b + d*F2.d;

 return(F0);
}
```

Figure 3: 2 Networks Connected

List 3: F-matrices Multiplication

List 3 shows the actual coding of the F-matrices' multiplication. The definitions in Lists 1 and 3 allow you to perform the multiplication of F-matrices using "* (asterisk)" as $F_0=F_1*F_2$.

When there are three networks cascaded as Figure 4, you can combine the networks F_2 and F_3 into F_4 and then combine the networks F_1 and F_4 into F_0. Since the F-matrices multiplication is already defined in List 3, the coding of 3 networks concatenation is simply performed as List 4 illustrates.

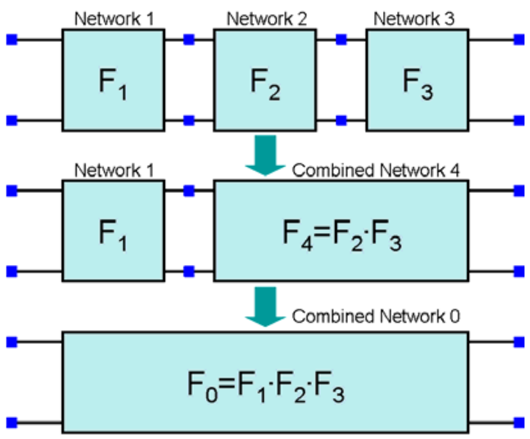


Figure 4: 3 Networks Connected

```
Fmatrix    F0,F1,F2,F3;
Fmatrix    F4;

F4=F2*F3;
F0=F1*F4;
```

List 4: F-matrices Connected

This is the way you can combine more than 3 networks cascaded into a single network by multiplying two F-matrices step by step.

Network Components

Figure 5 illustrates an impedance Z and an admittance Y. The impedance Z can be described as a complex number R+j•X which means a series connection of a resistance R and a reactance X, where "j" is the imaginary unit. The reactance X can be interpreted as ωL, where "ω" is 2·π·frequency and L is an inductance. The admittance Y can be described as a complex number G+j•B that means a parallel connection of a conductance G and a susceptance B. The susceptance B can be interpreted as ωC, where C is a capacitance. The impedance and admittance are complex numbers so that they can be coded as List 5 illustrates.

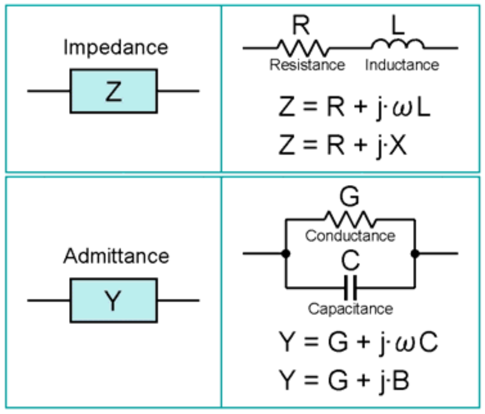


Figure 5: Impedance and Admittance

```
DOULBE     w,Freq;
DOUBLE     R,X,L;
DOUBLE     G,B,C;

w=2.0*M_PI*Freq;    // 2*pi*frequency

COMPLEX Z1(R,w*L);  // R+j*wL
COMPLEX Z2(R,X);    // R+j*X

COMPLEX Y1(G,w*C);  // G+j*wC
COMPLEX Y2(G,B);    // G+j*B
```

List 5: Z, Y Definition

Using the impedance Z and admittance Y, assorted F-matrices can be defined as Figures 6 to 9 illustrate, and their corresponding F-matrices are listed in Lists 6 to 9 respectively. Figure 9 and List 9 describe the transmission line. If the R, G, L and C parameters per unit length are given, the transmission line model can be expressed by the F-matrix "Fxline()."

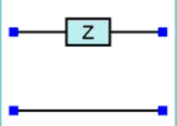
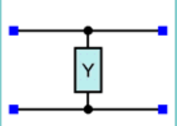
	$Fz = \begin{pmatrix} 1 & Z \\ 0 & 1 \end{pmatrix}$
	$Fy = \begin{pmatrix} 1 & 0 \\ Y & 1 \end{pmatrix}$

Figure 6: 1 Component F-matrices

```
Fmatrix Fz(
 COMPLEX Z
)
{
 Fmatrix H;
 H.a=One;
 H.b=Z;
 H.c=Zero;
 H.d=One;
 return(H);
}
```

```
Fmatrix Fy(
 COMPLEX Y
)
{
 Fmatrix H;
 H.a=One;
 H.b=Zero;
 H.c=Y;
 H.d=One;
 return(H);
}
```

List 6: Fz, Fy Routines

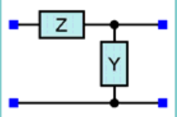
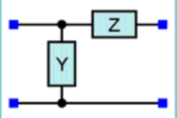
	$Fzy= \begin{pmatrix} 1+Z \cdot Y & Z \\ Y & 1 \end{pmatrix}$
	$Fyz= \begin{pmatrix} 1 & Z \\ Y & 1+Y \cdot Z \end{pmatrix}$

Figure 7: 2 Components F-matrices

```
Fmatrix Fzy(
 COMPLEX Z,
 COMPLEX Y
)
{
 Fmatrix H;
 H.a=One+Z*Y;
 H.b=Z;
 H.c=Y;
 H.d=One;
 return(H);
}
```

```
Fmatrix Fyz(
 COMPLEX Y,
 COMPLEX Z
)
{
 Fmatrix H;
 H.a=One;
 H.b=Z;
 H.c=Y;
 H.d=One+Y*Z;
 return(H);
}
```

List7: Fzy, Fyz Routines

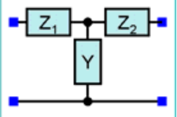
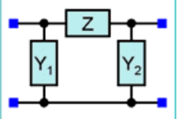
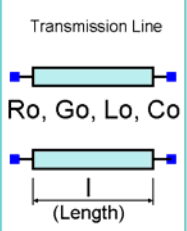
	$Fzyz= \begin{pmatrix} 1+Z_1 \cdot Y & Z_1+Z_2+Z_1 \cdot Z_2 \cdot Y \\ Y & 1+Z_2 \cdot Y \end{pmatrix}$
	$Fyzy= \begin{pmatrix} 1+Y_2 \cdot Z & Z \\ Y_1+Y_2+Y_1 \cdot Y_2 \cdot Z & 1+Y_1 \cdot Z \end{pmatrix}$

Figure 8: 3 Components F-matrices

```
Fmatrix Fzyz(
 COMPLEX Z1,
 COMPLEX Y,
 COMPLEX Z2
)
{
 Fmatrix H;
 COMPLEX Z1Y(Z1*Y);
 H.a=One+Z1Y;
 H.b=Z1+Z2+Z1Y*Z2;
 H.c=Y;
 H.d=One+Z2*Y;
 return(H);
}
```

```
Fmatrix Fyzy(
 COMPLEX Y1,
 COMPLEX Z,
 COMPLEX Y2
)
{
 Fmatrix H;
 COMPLEX Y2Z(Y2*Z);
 H.a=One+Y2Z;
 H.b=Z;
 H.c=Y1+Y2+Y2Z*Y1;
 H.d=One+Y1*Z;
 return(H);
}
```

List 8: Fzyz, Fyzy Routines



l	[m]
Ro	[Ω/m]
Go	[S/m]
Lo	[H/m]
Co	[F/m]

$$\begin{cases} Z = Ro + j\omega Lo \\ Y = Go + j\omega Co \end{cases}$$

$$\begin{cases} \sqrt{ZY} = \alpha + j\beta = \gamma \\ \dfrac{Z}{Y} = Zo \end{cases}$$

$$F_{xline} = \begin{pmatrix} \cosh(\gamma l) & Zo \cdot \sinh(\gamma l) \\ \dfrac{1}{Zo}\sinh(\gamma l) & \cosh(\gamma l) \end{pmatrix}$$

Figure 9: Transmission Line F-matrix

```
Fmatrix Fxline(
  DOUBLE R,DOUBLE G,
  DOUBLE L,DOUBLE C,
  DOUBLE Length,
  DOUBLE Frequency
)
{
  COMPLEX Zo,cosh_,sinh_;
  Fmatrix H;

  Xline(R,G,L,C,
        Length,Frequency,
        &Zo,&cosh_,&sinh_);
  H.a=cosh_;
  H.b=Zo*sinh_;
  H.c=sinh_/Zo;
  H.d=cosh_;

  return(H);
}
```

```
void Xline(
 DOUBLE R,DOUBLE G,DOUBLE L,DOUBLE C,
 DOUBLE Length,DOUBLE Freq,
 COMPLEX *Zo,COMPLEX *Coh,COMPLEX *Sih
)
{
 DOUBLE  w,al,bl,u,v;
 COMPLEX Z,Y,zo,zy,cosh_,sinh_;
 w=2.0*M_PI*Freq;
 Z=COMPLEX(R,w*L);
 Y=COMPLEX(G,w*C);
 zo=Csqrt(Z/Y);
 zy=Csqrt(Z*Y);
 al=zy.real()*Length;
 bl=zy.imag()*Length;
 u=0.5*(exp(al)+exp(-al));
 v=0.5*(exp(al)-exp(-al));
 cosh_.real()=u*cos(bl);
 cosh_.imag()=v*sin(bl);
 sinh_.real()=v*cos(bl);
 sinh_.imag()=u*sin(bl);
 *Zo=zo;
 *Coh=cosh_;
 *Sih=sinh_;
 return;
}
```

List 9: Fxline Routines

If you have a more complex network, you can divide it into several sub-networks illustrated in Figures 6 to 9. Then the big network is expressed as concatenated sub-networks. You can combine each sub-network's F-matrix step-by-step using the multiplication method described in Figures 3 and 4. Finally the original complex network can be dealt as a single F-matrix.

Exercise

Let's do an exercise with the network depicted in Figure 10.

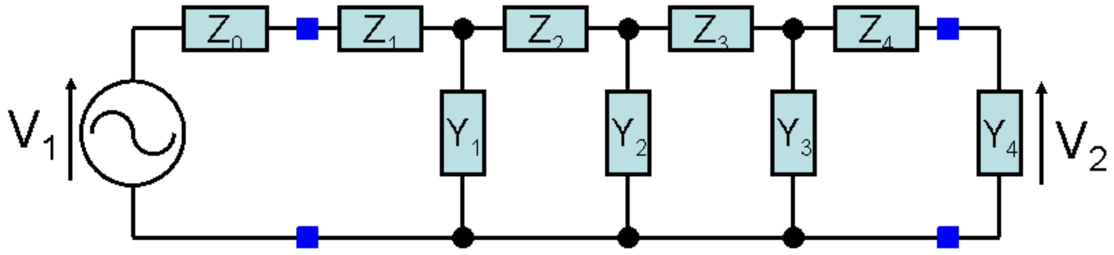
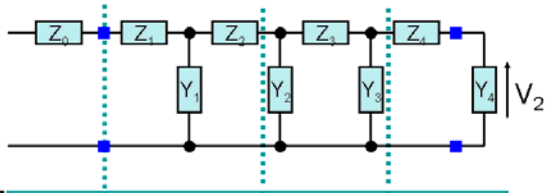


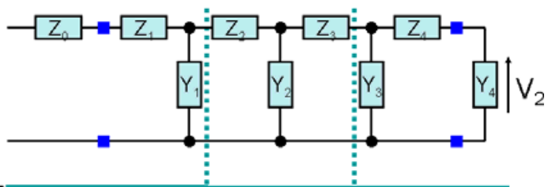
Figure 10: An Original Network

This circuit model can be decomposed as Figures 11 and 12 show. In Figure 11 it is separated into 4 sections and in Figure 12 it is separated into 3 sections. Either model would be fine. The coding is listed in List 10 and 11 respectively. Generally speaking, the fewer sections there are, the shorter the calculation time is.



Step	Fz	Fzyz	Fyzy	Fzy
1	F_1	F_2	F_3	F_4
2	F_1	F_2	$F_3 \cdot F_4 = F_5$	
3	F_1	$F_2 \cdot F_5 = F_6$		
4	$F_1 \cdot F_6 = F_0$			

Figure 11: F-matrices Decomposition



Step	Fzy	Fzyz	Fyzy
1	F_1	F_2	F_3
2	F_1	$F_2 \cdot F_3 = F_4$	
3	$F_1 \cdot F_4 = F_0$		

Figure 12: F-matrices Decomposition

```
Fmatrix F0,F1,F2,F3;
Fmatrix F4,F5,F6;


F5=F3*F4;
F6=F2*F5;
F0=F1*F6;
```

List 10: F-matrices Multiplication

```
Fmatrix F0,F1,F2,F3;
Fmatrix F4,F5,F6;


F4=F2*F3;
F0=F1*F4;
```

List 11: F-matrices Multiplication

List 12 shows the images of the gain V_2/V_1 calculation. The network in Figure 11 is used. Each RLC component values should be defined before going into the for-loop from 43 to 58. Line 57 calculates the gain of the network by the combined F-matrix component.

```

    INT           i;
    INT           N=1024;               // # of Data
    INT           Nsp=N/2;              // # of Spectra
    DOUBLE        dFs=500.0 MHz;        // Sampling Frequency
    DOUBLE        dFresln=dFs/N;        // Frequency Resolution
    DOUBLE        w;                    // "omega"= 2*pi*f
    DOUBLE        dFreq;                // Frequency
    ARRAY_D       Gain,Phase;           // Gain Phase Arrays
    COMPLEX       One(1.0,0.0);         // Constant 1+j0
    ARRAY_COMPLEX CGain;                // Gain Array (Rect. Coordinate)
    Fmatrix       F0,F1,F2,F3,F4,F5,F6; // Fmatrices

    DOUBLE        R0,R1,R2,R3,R4;       // Resistance Components
    DOUBLE        L0,L1,L2,L3,L4;       // Inductance Components
    DOUBLE        C1,C2,C3,C4;          // Capacitance Components



    CGain.resize(Nsp);                      // Gain Container (COMPLEX)

  for (i=0;i<Nsp;i++) {

    dFreq=dFresln*i;
    w=2.0*M_PI*dFreq;                       // "omega" 2*pi*frequency

    F1=Fz(COMPLEX(R0,w*L0));
    F2=Fzyz(COMPLEX(R1,w*L1),COMPLEX(G1,w*C1),COMPLEX(R2,w*L2));
    F3=Fyzy(COMPLEX(G2,w*C2),COMPLEX(R3,w*L3),COMPLEX(G3,w*C3));
    F4=Fzy(COMPLEX(R4,w*L4),COMPLEX(G4,w*C4));

    F5=F3*F4;                               // F-matrices Multiplication
    F6=F2*F5;                               // F-matrices Multiplication
    F0=F1*F6;                               // F-matrices Multiplication

    CGain[i]=One/F0.a;                      // 1/A
  }

    DSP_RECT_POL(CGain,Gain,Phase);         // Rectangular to Polar Conv.

    for (i=0;i<Nsp;i++) Gain[i]=20.0*log10(Gain[i]); // [dB] Gain

```
List 12: **Program Example**

In this article, the scheme of F-matrix application is explained. More specific examples will be introduced in future articles.