Tom McDermott, N5EG

3950 Southview Ter, Medford, OR 97504; n5eg@tapr.org

A Flexible 2-Port Network Calculator Tool

Computer program analyzes complex circuits using lab measurements of component networks.

Circuit simulation (such as SPICE) is used extensively in the design and optimization of analog circuits. For the large majority of circuit analysis problems it's an effective technique. However there are a few common RF analysis problems that are not easily addressed with circuit simulation. Another general technique of analysis has been developed based on 2-port networks. A 2-port is a network with two ports – input and output, each of which is usually ground-referenced. Commonly the 2-port will be drawn with two input connections (port + ground) and 2 output connections (port + ground). These 2-ports are commonly used for microwave circuit analysis, but the actual techniques are easily applied as well to HF and VHF/UHF RF networks commonly used by amateurs.

A modern Vector Network analyzer (VNA) can measure networks and extract the measured results represented as a 2-port network. Most commonly these are the S-parameters of a network, but it is also possible to extract the network transfer function as Z-parameters, Y-parameters and others. The VNA can measure the network response over a range of frequencies, and the response of the network is usually a set of the 4 parameters at each discrete measurement frequency. It is sometimes difficult to replace the measured response of the network with a simple circuit model that matches the response well over the measured frequency range. Instead the table of measured parameter values is many times the only practical description for the network.

Circuit simulators such as SPICE are difficult to use when we have empirical measured data tables that are frequency-dependent; in that case performing 2-port network computations may be easier and more indicative of the actual network values.

We sometimes know of stray circuit elements that are unavoidably present between our test equipment and the networks we are trying to measure. In this case it may be possible to subtract out the influence of the corrupting stray circuits mathematically, and this is especially so if the stray can be simply described (that is, it's not too complicated, and it does not bridge the network under test). While it is impossible to construct a physical network 2-port with negative component values, mathematically it's easy and actually fairly intuitive. Of course any such fictional compensation networks need to resemble as closely as possible the actual stray values in the circuit (except the sign), otherwise they will incorrectly compensate for the stray. If not, the compensation might be correct at one frequency but wrong at other frequencies. Interestingly, we can also use a model of a transmission line with negative length to provide compensation of both cable loss and phase delay.

On occasion we are able the measure the total response of a network, but we only know the values for some of the 2-ports that make up the network. It is possible to derive the response of the unknown 2-port block (or of the collection of unknown blocks assuming they are contiguous) by subtracting the known blocks from the overall measurement to leave just the unknown block.

These types of circuit analysis are difficult to do in circuit simulators such as SPICE, which may not converge well, may not handle negative component values well (or at all), are awkward to use with tabular parameter data, or don't compute the response of an unknown element easily.

Finally, we may want to specify a set of boundary conditions on 2-port networks (for example, the ratio and phase of drive currents to Z-parameter network). While this can be done in SPICE using a number of current sources, it is still difficult to closely model these conditions when the network itself is frequency-dependent (thus affecting the network port drive impedances) in a way that is modeled by simple circuit elements. In this case, 2-port analysis provides us some quite useful results. This case is particularly important when trying to model phased antenna arrays where there is a considerable amount of mutual impedance between the elements, and that mutual impedance changes with frequency.

Thus 2-port analysis can be a useful tool in a toolkit along with circuit simulation and other computation tools (such as a spreadsheet). This article will focus on 2-port modeling, and a flexible software tool written in Microsoft C# to make it fast, easy, and intuitive to do; then illustrate with three examples.

Definition of common 2-port Networks

Two-port networks have been described and used for many years. Some references that are or may still be available are Pozar[1], Matthaei, Young, Jones[2], and Gehrke[3]. The most commonly used 2-port networks are the S, Z, and Y networks, and the ABCD matrix representation. Each 2-port has a different set of port termination conditions used for defining and measuring each parameter. Each type can be mathematically converted to any of the other forms, but

[1]Notes appear on page 00.

the ABCD matrix turns out to be very easy to use computationally. In the included C# tool, each kind of 2-port network *Tile* (an object that the software uses to represent a single 2-port network) can have its ABCD matrix extracted. That extracted ABCD matrix is what is used for all calculations.

A 2-port impedance (Z) matrix relates the voltages measured at the ports of a 2-port network in response to the currents injected at the ports. Mathematically, we can describe the resultant voltages at ports 1 and 2 to the currents injected at ports 1 and 2 as:

$$V_1 = Z_{11}I_1 + Z_{12}I_2$$
$$V_2 = Z_{21}I_1 + Z_{22}I_2$$

Or, using matrix notation:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} * \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} \quad \text{or} \quad [V] = [Z] * [I]$$

We measure each term in the Z matrix by first injecting a current into port 1 and open-circuiting port 2 (zero current), then measuring the voltage at port 1 (V1/I1 = Z11) and port 2 (V2/I1 = Z12). Then we reverse the network, injecting current at port 2 and open-circuiting port 1, and measure the voltage at port 1 (V1/I2 = Z21) and at port 2 (V2/I2 = Z22). With phased antenna arrays we many times are concerned with the element currents so an impedance matrix describing the antenna array is commonly encountered and measured using this method, or other methods. Similarly, the admittance matrix relates the resultant currents at each port to the voltages injected at the other ports:

$$[I] = [Y] * [V]$$

S-parameters are defined when all ports are terminated in a matched impedance, for example 50 Ω. The S-parameters relate the resultant voltage reflected (V–) out a port compared to the voltage injected (V+) into the various ports.

$$\begin{bmatrix} V_{1-} \\ V_{2-} \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} * \begin{bmatrix} V_{1+} \\ V_{2+} \end{bmatrix}$$

ABCD Matrix description

An ABCD matrix describes the input voltage and current in terms of the output voltage and current of a 2-port network. This is backwards of the way that we normally think about circuit analysis — where we start at a generator and work our way towards the load. In analyzing ABCD matrices, we start at the load, and work our way towards the generator, asking essentially the question: "what voltage and current need to be supplied by the generator at the input to a 2-port network in order to cause a known voltage and current at the output of the 2-port network?" The terminology ABCD matrix is derived from the arbitrarily-chosen names of the 4 coefficients in the matrix. The two equations that describe the input node (V1, I1) in terms of the output node (V2, I2) are:

$$V_1 = A * V_2 + B * I_2$$
$$I_1 = C * V_2 + D * I_2$$

Where all the variables are complex numbers. It's convenient to represent this in matrix form as:

$$\begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} * \begin{bmatrix} V_2 \\ I_2 \end{bmatrix}$$

In the software tool the {V, I} pair representing a node is called a *NodeSet*, and the matrix holding the parameters of a two-port is a *ParameterSet*. The ParameterSet contains the 4 complex numbers of the ABCD representation of a Tile, at one single specific frequency. Then NodeSet(in) = ParameterSet * NodeSet(out). By using a matrix and vector to hold the values, and by defining a multiply operator, this reduces the description of the computation in the source code to a single multiply operation – just like the expression above because the software has defined matrix and complex operations for the *ParameterSet* and *Complex* object types.

An example derivation of a simple component network shows how to describe a 2-port via an ABCD matrix. Assume that the network is a simple series impedance between port 1 and port 2, with no impedance to ground. The voltages and currents are shown in Figure 1. The definition of direction of the currents shown in this figure allows direct chaining of the computations. The output current must be the same as the input current, and the input voltage must be equal to the output voltage plus the I*Z drop across the series impedance.

By inspection, the term 'C' must be zero and 'D' must be one so that the input and output currents are the same. The term 'A' must be one and the term 'B' must be equal to the series impedance Z. This yields the input node pair values as:

$$V_1 = 1 * V_2 + Z * I_2 \qquad V_1 = V_2 + Z * I_2$$
$$I_1 = 0 * V_2 + 1 * I_2 \quad \text{which simplifies to} \quad I_1 = I_2$$

So the ABCD matrix for a series impedance is:

$$\begin{bmatrix} 1 & Z \\ 0 & 1 \end{bmatrix}$$

Z can be composed of a combination of series or parallel components (resistors, inductors, and capacitors), or it could be the impedance presented by a transmission line. Thus Z generally is frequency-dependent, and we need to run the network chaining computation at each frequency of interest.

For a shunt network, the voltage on the input and output terminals are the same, but the input current is the sum of the shunt current and the current out of the network. Letting Z be the shunt impedance, then Y is the shunt admittance, equal to 1/Z. This yields the input node pair values as:

$$V_1 = 1 * V_2 + 0 * I_2 \qquad V_1 = V_2$$
$$I_1 = (1/Z) * V_2 + 1 * I_2 \quad \text{which simplifies to} \quad I_1 = Y * V_2 + I_2$$

So the ABCD matrix for a shunt impedance is:

$$\begin{bmatrix} 1 & 0 \\ Y & 1 \end{bmatrix}$$

Using an ABCD Matrix

Essentially, the ABCD matrix tells us that if we know the output *NodeState* (the voltage and current at the output of the tile), we can derive what the voltage and current at the input of the tile would have to be in order for that known tile output voltage and current to be what they are. By computing and storing the node state at each node, we can quickly select the display of impedance, return loss, or other node parameters at any node, and similarly we can compute the voltage or power transfer between any pair of nodes by walking the computation from the output of the last tile to the input of the first tile.

The various types of 2-port networks can be converted back and forth between forms. From a computational perspective, it's easy to use the ABCD format for chaining the calculations across multiple 2-port devices. While it is possible to chain other format calculations (such as using the chaining rules for S-parameters) the computation is straightforward when using an ABCD matrix.

When using ABCD matrices, the total network response is simply

the cascade of the various 2-port networks, achieved by progressive matrix multiplication of all the 2-port networks starting at the last 2-port. For the C# tool described in this article, the approach chosen was to save the computations for all of the nodes in the network. The { voltage, current } pair are computed at each node and saved, and the input nodestate of the previous computation becomes the output nodestate of the next. This is accomplished by matrix multiplication of the network ABCD matrix times the output node voltage, current, resulting in the input node voltage, current, and then iterating to the front of the network (saving all the node pairs along the way).

The software defines an object known as a *ParameterSet* which contains the frequency of the dataset, and the 4 complex parameters of a tile at that one frequency. The ABCD parameters are extracted from the *Tile* depending on what kind of 2-port network the set is describing. The tile can optionally contain a collection (in a list) of *ParameterSets*, one per frequency, or it can be described at just one frequency if the tile is easily described analytically and the frequency-dependent behavior can be easily computed (such as a resistor, capacitor, inductor, or combinations of them). Additionally a *NodeState* object contains the complex voltage and currents at a node.

The calculation computes a list of NodeStates, one for each node in the network (a network has one more node than there are tiles). The ParameterSet multiplication operator is defined so that a 2x2 matrix times a 2x1 vector (ABCD matrix times NodeState) or 2x2 matrix times a 2x2 matrix (ABCD matrix times ABCD matrix) multiplication is performed depending on the types of the objects being multiplied. This makes the ABCD chaining calculations look simple in the source code while in reality a significant quantity of matrix multiplication and complex number computations are happening as a result of that single multiply instruction.

Computing Network Response

To compute the network response (a cascade of tiles) we start at the last tile in the network, and set the output voltage and current (the nodestate after the last tile) to {V=1+j0, I=0+j0}. This means that the output voltage is 1 volt at a phase angle of zero degrees, and the output current is zero — essentially we terminate the last tile into an open circuit. Then we work our way back towards the source (at node zero) through all the tiles computing each node voltage and current pair as we go, and saving them as a list of NodeStates. At any point, the voltage transfer function is simply the last node voltage (1+j0) divided by the node voltage at the present point. We can compute the impedance at any node as V node / I node.

The power transfer can be different than the voltage transfer, because the impedance at each point in the network can be different. When dealing with complex voltages and currents, we need to take into account the phase angle of each. The power delivered into a load is:

$$P = \frac{1}{2} Re\left(V * I^{*}\right)$$

The real part of the product of the voltage V and the complex conjugate of the current I is the power into the load. The factor of one-half arises because V and I represent the peak voltages of a sinusoidal wave since they are complex vectors. We can compare the power at any point in the network to any other point — except the last node. The last node (which is after the last tile) is an open circuit, and no power can be delivered to an open circuit (because the current is zero). The C# tool will signal an error if we try to compute the power transfer from some node to the last node, since it is not possible to deliver any power into that open circuit.

Terminating Tile

In fact, the last tile is somewhat special in the network analysis. Many times we will make an S-parameter measurement of a complicated load (such as an antenna). The resulting S-parameter matrix including the antenna does not have a defined set of output terminals (it's really a one-port measurement), and thus no port-to-port transfer function exists. S12, S21, and S22 are many times just set to zero in the resulting measured S-parameter file by the VNA when a 1-port measurement is made. We cannot derive an input voltage and current that can cause the output voltage and current of 1+j0, 0 because there's nothing actually connecting port 2 to anything (we would attempt to compute infinite voltage times zero transfer).

The software tool makes a special case for the last tile – the input term S11 (or Z11 or Y11) is derived based on the output terminal pair not being used, and uses these modified values in the extracted ABCD parameters set rather than the normal 2-port extraction. This turns the last tile into a simple one-port termination of the network that is being analyzed, which is actually what we likely measured with our test equipment anyway.

The pseudo-code for calculating the entire network response is then:

```
Foreach (frequency)
{
  // open circuit after last tile
  Set NodeState(out) = {V=1+j0,I=0+j0}
  Foreach (tile, starting at the last tile
        and working forward)
  {
    //Calculate {Vin,Iin} from {Vout,Iout}
    // using the ABCD matrix
    //  of the tile at the current
    // frequency
    NodeState(in) = ParameterSet(in ABCD form) *
          NodeState(out)
    Append NodeState(in) to the list
        of node values for that frequency
    Update the value of NodeState(out) =
        NodeState(in) and continue
  }
}
```

Deriving the Input Impedance

An important class of problems to analyze deals with determining the input impedance of a 2-port where there is mutual impedance between the input and the output, and some driving condition between the input and output is known. This occurs for example in the case of a vertical antenna array where multiple elements are driven with various known currents but at (possibly) different phase angles. The relationships between the antenna can be described as a Z-parameter matrix (likely frequency-dependent) where the self-impedance of the elements (Z11, Z22, etc.) are known, and the mutual impedance of the elements to one another (Z12, Z21, etc.) have been previously measured. These mutual impedances change the driving point impedance of each antenna depending on the excitation magnitude and phase at the other antennas and thus affect the performance of the array, it cannot be ignored.

To determine the input impedance of the 2-port in this case requires setting a forcing condition on the Z-parameter network during analysis. Normally we compute [V1,I1] based on the [V2,I2], but in this case we may know both I2 and I1 magnitude and phase (or their complex ratio), while not knowing V2 and V1. We must specify I1 in relation to I2. Since we do not know I2 when we are describing

the network, we specify I1/I2, as a ratio. This ratio must include the amplitude of the currents driving the two antennas (the two ports) as well as the phase relationship of the currents to one another (are we driving the antennas in-phase, out-of-phase, phase-quadrature, or something else).

Given the ratio, we have two unknowns and two equations, and can thus compute the node voltages, and from that the network input and output impedances. If we set I2 equal to 1+j0, then specifying a condition that the ratio I1/I2 = –1 would tell us that the two elements are driven in-phase but with the same current magnitude (recall the negative sign in defining the current convention). Setting I1/I2 = 0–j1 for example would set the driving current magnitudes equal but 90 degrees out of phase. The C# program specifies the ratio in dB, and phase angle in degrees to make the ratio entry more intuitive. The program extracts at each frequency the ABCD matrix from the Z-parameter matrix (or Y-parameter or S-parameter matrix).

First we derive V2 knowing both I1 and I2:

$$I_1 = CV_2 + DI_2 \quad thus \quad V_2 = \frac{I_1 - DI_2}{C}$$

Then we can compute V1 knowing both I2 and V2 using the ABCD formula we already know:

$$V_1 = AV_2 + BI_2$$

Now the analysis of the 2-port network can proceed normally since we have computed the node conditions at node 1 and it can cascade forward in the standard manner. In order to force some current at I2, there must be a shunt network terminating the output of the tile. It really does not matter too much what that shunt network is, because the input current is defined as a ratio of the output current, and V2 will be computed as necessary to force the value of shunt current, but it's convenient to use a shunt resistor of large value. Additionally, appending a shunt network after the Z-parameter network means that it is not the last tile, and so the full set of 4 Z-parameters (or Y, or S) will be used (which is what we want).

Transmission line

The response of a transmission line is described by several fundamental parameters: loss, velocity factor, and characteristic impedance. The latter two properties can be used in an equation to describe the input impedance of a transmission line that is terminated in an arbitrary load impedance. Zo is the characteristic impedance of the line, Zl is the load impedance, and beta is the phase delay characteristic of the line per unit length. A simple cable approximation is:

$$Z_{in} = Z_o \frac{Z_L + jZ_o \tan(\beta l)}{Z_o + jZ_L \tan(\beta l)}$$

However, this does not account for the loss of the transmission line, just its phase. A more general formula that includes line loss can be used if we are able to utilize complex arguments to hyperbolic trig functions. A side benefit is that the resulting equation appears simpler than the one above. In most of the classic math textbooks, the complex variable z (not impedance) is defined as x + j y. Then the trigonometric identities (in terms of x, y, and z) are:

$$\sinh(z) = \sinh(x) * \cos(y) + j\cosh(x) * \sin(y)$$

$$\cosh(z) = \cosh(x) * \cos(y) + j\sinh(x) * \sin(y)$$

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)}$$

If we define a new constant gamma, $\gamma = \alpha + j\,\beta$, which includes both the loss (alpha) and phase delay (beta) properties per unit length, then $\gamma * l = \alpha\, l + j\,\beta\, l$ represents the loss and phase delay for the entire length of the cable. $\alpha * l$ is the loss of the line section, in Nepers; $\beta * l$ is the phase delay of the line section in radians. The general transmission line input impedance equation including loss then becomes:

$$Z_{in} = Z_o \frac{Z_L + Z_o \tanh(\gamma l)}{Z_o + Z_L \tanh(\gamma l)}$$

Where Zin, Zl, and Zo are complex impedances, using the previous identities to compute the hyperbolic tangent functions. The above formulation is usually called the low-loss cable approximation.

The COMPLEX number library included in the C# program provides regular and hyperbolic trig definitions for complex arguments, which makes the source code much more readable. *Visual Studio* C# 2008 does not include complex numbers, so a COMPLEX module was written to perform the common operations.

The Zin of the transmission line computed above is inserted into the ABCD matrix. For a series transmission line, the resultant ABCD matrix is:

$$\begin{bmatrix} 1 & Zin \\ 0 & 1 \end{bmatrix}$$

Where Zin is the result of the previously derived load to the transmission line Zl. Series and shunt stubs are modeled similarly to the series transmission line, except that the termination impedance of the stubs must be known as a circuit value (rather than being computed as in the case of the load terminating a series transmission line). The ABCD matrix for a series stub is the same as above, while the ABCD matrix of the shunt stub is:

$$\begin{bmatrix} 1 & 0 \\ 1/Zin & 1 \end{bmatrix} \quad \text{also can be shown as} \quad \begin{bmatrix} 1 & 0 \\ Yin & 1 \end{bmatrix}$$

The program uses a simplified stub termination impedance consisting of a resistor in series with an inductor and the series combination of those two in parallel with a capacitor. This matches pretty well to the cases where a low value of termination resistance has some series lead inductance, or when an open circuited line has some fringing capacitance.

As with other RLC loads, setting C=0 means 'no capacitance'. The program traps out cases where setting a reactance to zero would cause a divide-by-zero error, and instead removes that device from the model.

The loss of a transmission line can be approximated above a few hundred kilohertz from an equation that fits two coefficients to a function of frequency, one directly proportional to frequency, and the other proportional to the square-root of frequency. The program provides a drop-down list of a few commonly used coaxial and twin-conductor types of cables and pre-populates the characteristic impedance (both real and imaginary parts), the loss coefficients, and the velocity factor, you must supply the length of the cable. The values used in this program come from VK1OD's on-line transmission line loss calculator[4]. The pre-populated values can be over-ridden with other values of the parameters for a particular case, if known. The characteristic impedance of many coaxial cables starts to change a lot below one megahertz becoming increasingly more reactive, so the above equations need to be used with appropriate caution.

Series and Parallel RLC Networks

Series and shunt impedances are pretty straightforward to model as 2-port networks. A series impedance could be composed of a series connection of RLC parts, or perhaps as a parallel connection of RLC parts. At each frequency we compute the complex impedance of the collection of parts, and substitute that complex impedance number into the series ABCD matrix. Similarly a shunt impedance could be either series RLC, or parallel RLC. Again the complex impedance at each frequency is computed and substituted into the shunt ABCD matrix. These are the same matrix format as shown above.

Many times we will only have one or two of the RLC elements, and a common short-hand notation is to use a component value of 0 to signify that no component is present. Thus the calculator tool needs to handle a few special cases of zero value and instead remove that device from the circuit. For example, a value of 0 μH in a parallel circuit would short out the circuit, when what we really mean is that there is no inductor present (thus, the inductor has an infinite parallel impedance rather than a zero parallel impedance).

Transformers

A perfect transformer is easily modeled just from the turns ratio, but is not very useful since they don't exist and the assumption of perfect coupling is usually poor. Good transformer models can be quite complex yet still have significant error compared to the actual RF devices. A simple compromise model was selected that requires describing the primary winding inductance, the turns ratio, and the mutual coupling between the primary and secondary, but ignores winding resistance, stray capacitance and other effects. The symbols are defined as:

L1 = primary inductance
L2 = the secondary inductance
N = the turns ratio = Secondary turns / Primary turns
k = coupling coefficient

Traditionally it ranges from zero to plus one, but in this model we allow it to range from –1 to +1. A negative value indicates that the secondary winding is opposite in phase compared to the primary, thus a value of negative 1 indicates perfect coupling between windings but the transformer inverts the polarity of the output compared to the input. A value of +1 indicates the output winding has the same polarity as the input and is perfectly coupled. A value of zero indicates no coupling between the two windings at all. M is the mutual inductance computed from the above parameters. Since the model requires L1 to be defined, we compute the value of L2 (secondary inductance as):

$$L_2 = L_1 N^2$$

And the mutual inductance is computed as:

$$M = k\sqrt{L_1 L_2} = kNL_1$$

This shows that M has the same algebraic sign as k.

We then translate the transformer model into a T network with 3 inductors having inductance values as shown in Figure 3. Note that some of the inductors will have negative inductance values depending on the value of k. Then we compute the impedance of each element at each frequency of interest. Z1 = impedance of (L1–M), Z2 = impedance of (L2–M), and Z3 = the impedance of the mutual inductor (M). The ABCD parameters are derived from those impedances as:

$$ABCD = \begin{bmatrix} 1 + \frac{Z_1}{Z_3} & Z_1 + Z_2 + \frac{Z_1 Z_2}{Z_3} \\ \frac{1}{Z_3} & 1 + \frac{Z_2}{Z_3} \end{bmatrix}$$

Reading Data From a Measured Network

An important capability of the tool is to read in 2-port networks that have been measured by external test equipment. An industry-standard file format for 2-port networks is the Touchstone S2P format[5]. Most Vector Network analyzers (VNAs, and some other types of equipment) are capable of exporting an S2P formatted file. This tool is able to read an S2P file, decide what kind of 2-port network format it represents, parse the file, and store the 2-port parameters as a function of frequency in a data structure for that network tile. Each network block independently stores its own retrieved data, as the data for each network may have different formats, frequency ranges or frequency step sizes.

Parameter Frequency Interpolation

Since the 2-port data are captured by the test equipment at specific discrete frequencies, and the 2-port analysis may occur at frequencies that are not exactly aligned with the measured frequencies, some means to interpolate the retrieved data is needed. Since a 2-port parameter is in general a complex number, it is important to account for the fact that the phase could wrap almost 360 degrees between adjacent samples. The simplest way I have found to interpolate these samples is to convert the parameters to real + imaginary format, and then perform two one-dimensional linear interpolations in frequency (one for the real component, and one for the imaginary component of the parameter). This eliminates complications related to wrapping of coordinates.

It's possible to describe the output node in terms of the input node (backwards from what we have been doing so far) by using matrix inversion, although with the proviso that sometimes the ABCD matrix cannot be inverted (sort of like trying to divide by zero). We multiply both sides by the inverse matrix to work backwards.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} * \begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} * \begin{bmatrix} A & B \\ C & D \end{bmatrix} * \begin{bmatrix} V_2 \\ I_2 \end{bmatrix}$$

thus

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \begin{bmatrix} V_2 \\ I_2 \end{bmatrix}$$

since a matrix times its inverse is the unity matrix (similar to multiplying by one).

The C# program defines an inversion operator for the ParameterSet matrix because it turns out to be useful when trying to remove a known ABCD matrix's impact when we have an unknown matrix. Additionally, an "IsSingular" test is implemented for the matrix inversion. A matrix which is singular cannot be inverted, this test allows us to abort a computation before error.

This has been left in as a hook if a capability to derive unknown 2-port network properties is added to the program in the future.

C# Program Implementation

The 2-port tool was written in Microsoft *Visual Studio* 2008 C#, which supports the NET 3.5 runtime (which is needed by the chart control). It has also been tested with *Visual Studio* 2010.

The Microsoft C# language starting with version 2.0 provides a "generic" capability, similar to the C++ "template." This allows the use of data structures, such as *List*, that can apply to a wide range of

data types. The List<Type T> data structure is very easy to use in C#. The tiles, parameter sets, node sets, etc. are each contained in lists, which do not have specific size constraints. There are ways to search through and re-order lists, and to identify an element in a list by index (position in the list) and to retrieve an index or a copy of the list element itself.

The design element types are marked as "*Serializable*" in the software. This allows a very simple implementation for streaming (saving) the data out to disk and retrieving it back. However each and every type that ultimately is sent to disk (even when embedded within another type) must be serializable. For example the *Complex* number type must be serializable in order to make a *ParameterSet* (which contains 4 complex numbers) serializable.

Additionally, each data structure has "Properties" associated with most of the internal object variables. By explicitly marking *Property* meta-data in the source code, the structures are "browsable" by an object browser control which accesses various Properties meta-data. The object browser is a tool that can be added to a windows form to permit run-time inspection and change of object variables. It's a little clunky to need to implement a property for each variable, but the ease of testing and debugging with the object browser is usually a good trade-off. In earlier editions of *Visual Studio*, the object browser tool needs to be loaded into the toolkit, as it is not installed by default. The toolkit is used only when compiling programs. When running the executable part of the program the browser tool is embedded within the EXE.

Required Runtime Packages

To run the software, the .NET 3.5 SP1 framework needs to be downloaded and installed. The NET 3.5 SP1 framework is a free download.[6]

Microsoft released a free very flexible chart control for .Net applications [MS Chart][7]. The chart control also needs to be downloaded from the Microsoft Web site and includes examples, help, etc. The control comes in two pieces: the runtime package, and the *Visual Studio* designer integration package. Only the runtime portion is needed to run the software.

Required Compiler Packages

A free version of C# 2008 (Express Edition) is available from Microsoft. There is a newer version available, C# 2010 Express, and the package has been compiled and tested under both versions[8]. The code in this tool was developed using the 2008 professional version. The free version does not support building an MSI installer package, but this 2-port tool does not interact with device drivers or the underlying Windows API's and does not have any specific installation requirements – thus an installer does not need to be built. The free Express Edition tool is all that's needed to change, compile, and experiment with the source code. One limitation of *Visual Studio* 2008 compared to predecessor versions is that it no longer produces EXE files that work with Windows 98, ME, NT, or 2000. The compiled code only works with Windows XP, Vista, and Windows 7.

C# 2008 Express does not support the integration of the MSChart control within the *Visual Studio* designer (thus allowing placing the control on a form, changing the size, etc.) but it will compile the code provided with this article since the control is already embedded in the designer resource. Thus you can change the code, but cannot interact with the control very much via the designer. If you have *Visual Studio* 2008 (not the free 2008 C# Express Edition), then you can also download the MSChart designer integration package which allows you to make all design-time changes on the chart control.

The express edition does not install the *Visual Basic* Power Packs 2.0 (*Visual Studio* 2005 version) by default. You will need to install version 2.0 of the control[9] so that the Microsoft.VisualBasic.PowerPacks.VS namespace reference can be found. Different versions of C# may or may not link properly to this reference. If it cannot resolve the reference (compile errors, and an error exclamation mark in the references folder of the solution explorer), first delete the reference, then add it back in again which will prompt you to provide a link to the proper location on disk.

Using the Program

The Design tab of the program is where designs are entered. Initially the program brings up a grid of 24 slots. The slots are in series, from upper left to lower right, across each row left-to-right then down to the next row. The last (right-most) tile on the first row is just before the first (left-most) tile on the second row. Right-clicking the mouse allows inserting or deleting tiles. The first tile can be inserted using either Insert Right or Insert Left, and it will be inserted between nodes 0 and 1. The insert operation brings up a menu that allows selection of the type of tile (2-port) that is to be placed. Tiles that have been placed on the design tab can be selected by clicking the mouse over the desired tile, and that tile will be highlighted. The selected tile can be deleted by right-clicking the mouse. The selected tile can be browsed in the tile browser tab, and its parameter type, and component values can be quickly changed. Once a tile is selected, a new tile can be inserted to the left of or to the right of the currently selected tile. If a new tile is inserted when no tile is selected, it will be inserted at the first possible position (between nodes 0 and 1) and all the rest of the tiles will get bumped to the right. The node numbers are shown on the background of the design tile to make it easier to identify and remember node numbers when specifying them on the analysis tab.

The tile selection menu allows direct input of the parameters, selection of a forcing current ratio (for Z-parameter antenna matrices usually), and whether a 4-parameter set for a (S, Y, Z, or ABCD) tile is fixed in frequency, or is frequency-dependent. If it is frequency dependent, a menu pops up allowing the selection of an S2P file to load it from. A design can be saved to a file, the design cleared, and a design retrieved from a file. When saving a design, the entire state of the design tiles are saved, including all the frequency-dependent parameters (they lose their association to the original S2P file but all the data are saved). A List data structure holds the frequency-dependent list of parameters values, so the program does not set arbitrary limits on the number of frequency points that any tile can hold until the program runs out of memory or disk space. Designs with over a thousand frequency points have been tested (and it results in a several hundred KB file).

One limitation is that the frequency points must be in the original S2P file in increasing frequency order, but I've not yet found any S2P files that violate this ordering. The actual frequency of each analysis point does not need to line up with the frequencies read in from the S2P file, nor do different tiles need to align in frequency with any other tiles. The software searches each tile and finds the bracketing set of frequency parameters, then does a linear-in-frequency interpolation between them to the actual analysis frequency. This works well except in the case where the parameter values have a large deviation from a linear approximation between the two frequency points. In that case, the S2P file that the parameters were read in from needs higher measurement resolution and more points or a narrower frequency sweep to minimize the difference between adjacent measured samples.

The analysis parameters page allows editing the analysis parameters (start and stop frequency, selected displays, number of analysis points, etc.) in an object-browseable format, it duplicates much of the functionality of the controls on the chart tab.

The MSChart control is used in the C# program to display the

results of the analysis over frequency on the analysis tab. The chart does not have a polar-mode that can work as a Smith chart, so it's only used to display the parameters in rectangular format. The zoom, pan, and other modes of the chart are active so you can drag the mouse to zoom into the chart, or push the zoom-reset button on each axis to go back to the full chart. It's faster to do than to explain.

The Analysis Tab of the program brings up the chart control and a few buttons and check boxes to enable displaying the impedance (in several formats) at a selected node, and the voltage and power transfer functions between a pair of nodes. The analysis button will turn pink when the analysis results in some type of error (such as trying to measure the power into the last node after the last tile). The values are clipped to –308 dB when a zero-value is converted to dB to avoid a run-time exception with the chart. If any values display as –308 dB, then it probably represents a value of minus infinity. There's a control or two to set auto scale or manual scale and max/min values for the Y-axis (except phase, which has a fixed axis of –180 to +180 degrees), and to select linear or logarithmic frequency axis.

The Tile Browser allows changing the values of the selected tile (the one with a black box around it). This is very convenient for changing tile values, types, or drive conditions quickly.

Source Code

The complete C# source code tree with all designer resources, the compiled executable file, and the examples have been placed on the ARRL QEX Web site[10]. If you just want to run the program, simply download the ABCDmatrix EXE file, unzip it, and execute it. You should install the CHM (help) file into the same folder as the executable program.

The source can be edited, compiled, and built in the *Visual C# Express* 2008 integrated design environment (IDE). Make sure to check that all references are present, and resolve any that are missing (such as the visual Basic Powerpack, the MSChart control, etc.).

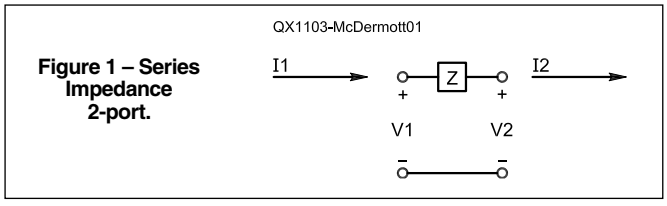

Figure 1 – Series Impedance 2-port.

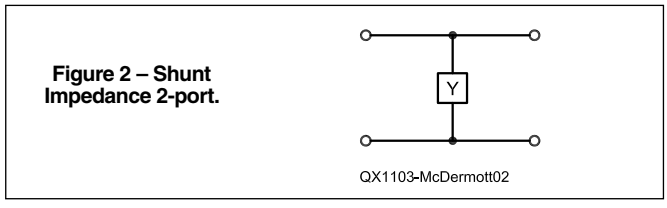

Figure 2 – Shunt Impedance 2-port.

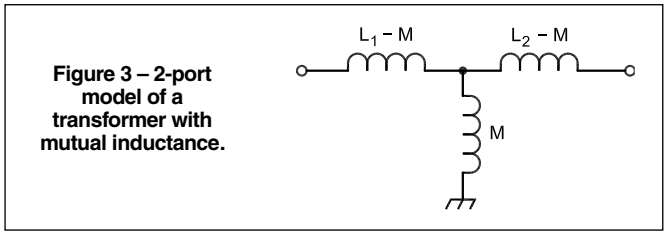

Figure 3 – 2-port model of a transformer with mutual inductance.

Menus & Operation

The program has a pretty simple interface. Select the Network Design Tab, and Right-click to bring up a context menu. When there are no tiles on the design surface, or no tile has been selected, either InsertRight or InsertLeft will insert a new tile at the first location (between nodes 0 and 1) and bring up a menu to specify it. The menu allows fixed values or frequency-variable values read from a file (for S-, Z, and Y- parameters). Left-click a tile to select it. The selected tile can be edited on the Tile Browser tab. The selected tile can be deleted, copied to the clipboard, or cut (and copied to the clipboard). InsertRight and PasteRight will insert a tile after the selected tile. InsertLeft and PasteLeft will insert a tile before the selected tile. Delete will remove a tile without copying it to the clipboard, cut does copy it to the clipboard.

The Analysis Parameters tab allows changing the analysis parameters, but largely duplicates the checkboxes and numeric windows on the analysis tab. The Analysis tab displays the results of an analysis. The 'from' box selects the node where impedance (or S11) is displayed, and is the starting point for a Power transfer or Voltage transfer analysis. The 'to' box is the terminating node for voltage or power transfer. Note the power transfer can not be computed at the output of the last tile because no power can be delivered to an open circuit. The tab will catch and correct errors in the values of the 'to' and 'from' tab (except for power analysis) and will try to select the whole design when blank or out-of-range values are selected. The Analyze button will update the display. It will turn pink if a gross analysis error is present (for example trying to display the power into an open circuit).

A design can be saved to and retrieved from disk, it uses the suffix 'til' (for 'tile'). 'New' erases all the tiles on-screen and the clipboard.

Note that you can use the mouse to drag/select an area of the chart and the view will zoom into that rectangular area. Slider controls allow scrolling horizontally or vertically, and buttons allow reverting back to un-zoomed axis.

Examples

Here are three examples and solutions worked out using the tool. The examples have been chosen to illustrate some common functions of the software tool, especially where one or two of the uses might otherwise be a bit confusing.

Example 1 – shunt and series coaxial stubs.

The first example is a 50 Ω series source feeding a shunt stub, then a series stub, then a 50 Ω shunt load resistor. Figure 4 is a screen

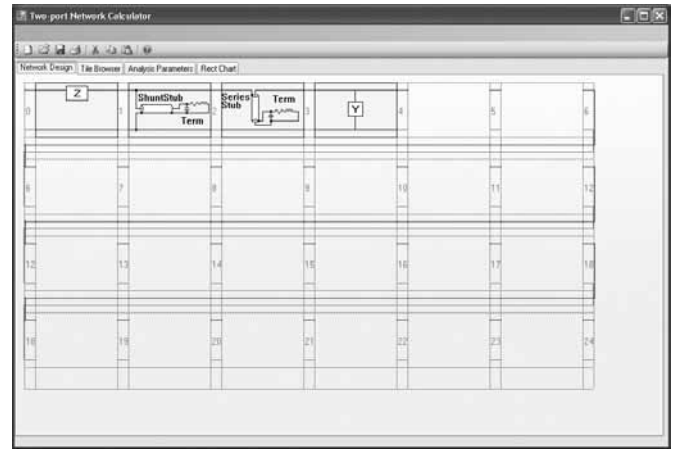

Figure 4 – Network Design tab shown the circuit of example 1, with the shunt stub selected.

shot of the Network Design Tab. As each tile is inserted, a pop-up menu allows selecting the type of tile, and the parameters of that tile. Later on, an existing tile on the Network Design tab can be copied, which copies all its parameters, or an existing tile can be selected and then edited on the tile browser tab. The first element has a series RLC of 50 Ω + 0 μH + 0 pF. The 0 pF means "no capacitor", so the first series element is 50+j0. The shunt stub has a termination R=0.2 Ω, L= 0.02 μH, and C = 1 pF. It's made from RG-58, and has a length of 10 meters. The velocity factor, characteristic impedance and loss parameters are filled in automatically when selecting the RG-58 cable type. The series stub is also 10 meters of RG-58, but has R=1 MΩ, L=0 μH (is open-circuited instead of shorted), and has 1 pF of capacitance. The shunt load is series RLC of 50+j0. The shunt stub is selected.

Figure 5 shows the Tile browser tab. Since the shunt-stub is selected, it's what is being browsed by the tile browser. All of the tile parameters can be changed using the tile browser. The Cable length field is highlighted, and the stub is 10 meters long. Figure 6 shows the Rect Chart tab — the analysis results in rectangular chart format. The voltage transfer from node 0 to node 4 is shown, magnitude in dB, and the phase in degrees. We can see that the series and shunt stubs, although both of exactly the same length, aren't exactly on the same frequency due to slight differences in their stray termination impedances. Also, the insertion loss increases slightly with frequency (more readily seen using the power transfer from node 0 to node 3 (not 4 as the transfer to node 3 is the power transfer into the 50 ohm shunt load).

Figure 7 shows the input impedance at node 0, in real ohms, imaginary ohms format. If you change the series source resistor from 50 ohms to 0 ohms, a much different transfer function results. Y-auto

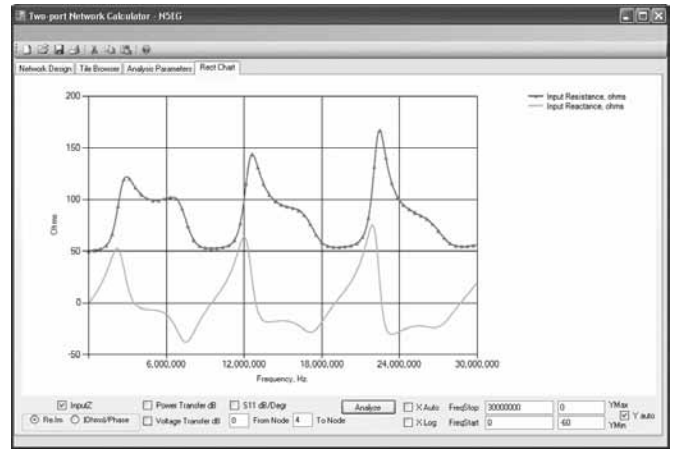

Figure 7 – Example of data input and output on the Rect Chart Tab showing real and imaginary impedance at Node 0.

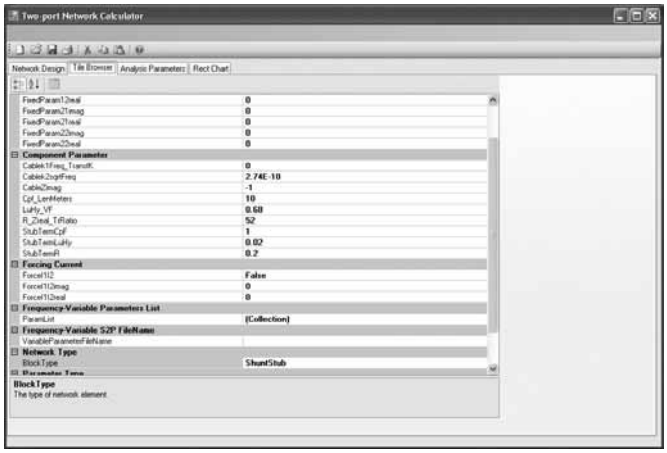

Figure 5 – Example of data input on the Tile Browser Tab for the selected shunt stub of Figure 4.

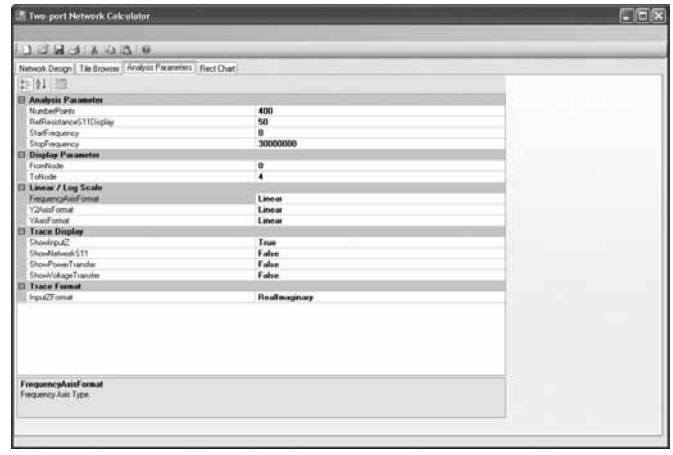

Figure 8 – Example of data input on the Analysis Parameters Tab.

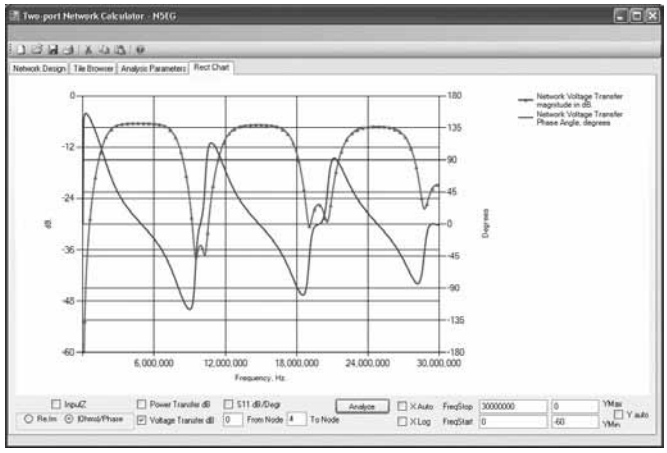

Figure 6 – Example of data input and output on the Rect Chart Tab showing magnitude and phase.

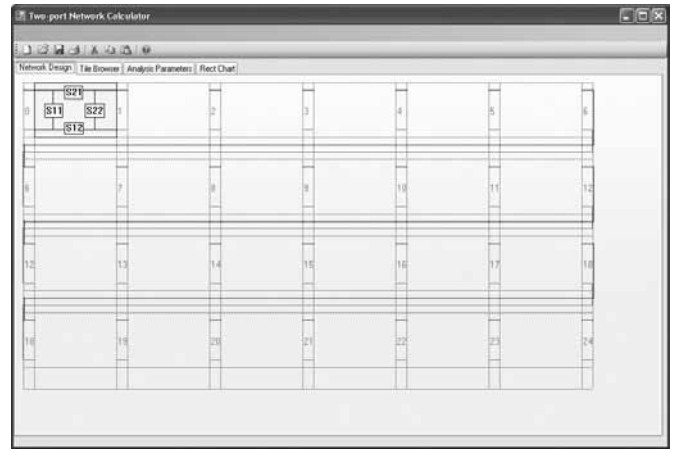

Figure 9 – The Network Analysis Tab showing S parameter input for a Tile.

scale is selected, analysis range is zero to 30 MHz.

Figure 8 shows the Analysis Parameters tab. Normally most analysis parameters can be selected from the 'Rect Chart' tab. Sometimes however the defaults may not be what are desired. This tab allows changing for example the Reference resistance for S11 analysis (the default is 50 Ω), or the number of points in the analysis (this example uses 400 points), as well as other parameters.

Example 2 - Subtracting networks

This next example shows how to mathematically remove known text fixture strays from a measured network response. The solution is hypothetical since the compensation consists of networks that in fact cannot be realized. However, its purpose is to let us discover the underlying network properties.

Figure 9 shows an S-parameter tile, which is loaded from data measured by a VNA of a termination load through some RG-58 coaxial cable. Figure 10 shows the S11 analysis at node 0, which is exactly the same as what's displayed on the network analyzer (as it should be). The number of points and the start/stop analysis range have been set to the same as the VNA measurement. Figure 11 shows the addition of a coaxial line segment in front of the measured result. The length of the coaxial cable is set to negative 1.04 meters, and the characteristic impedance is set to 50.75 – j0.4 Ω. This approximately subtracts out the effect of the cable feeding the termination load (but ignores some other error sources), letting us see that the load impedance is roughly 50.5 Ω (with some uncompensated error).

Figure 12 shows the added cable compensation parameters. This allows experimenting with a wide range of parameters values for the selected tile's values. Figure 13 shows the compensated S11 measurement. Figure 14 shows a close-up of the resultant real part of the load impedance.

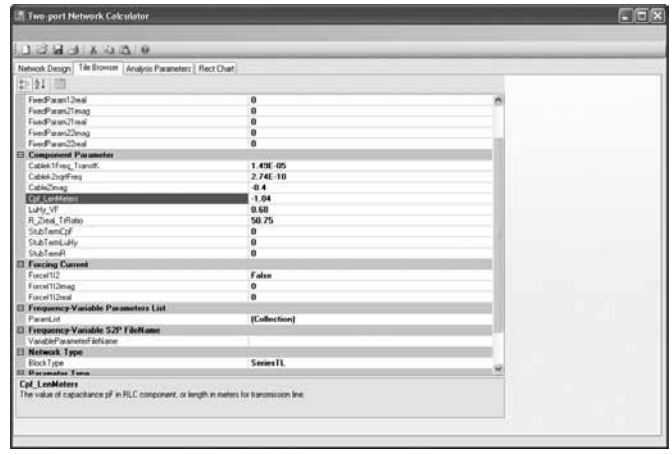

Figure 12 – The compensating transmission Line data shown in the Tile Browser.

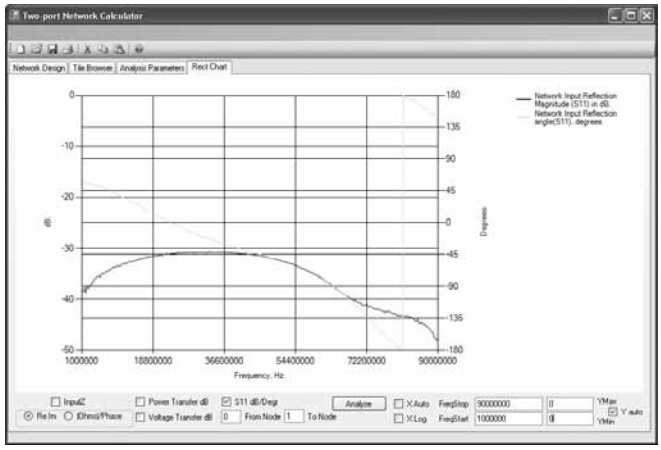

Figure 10 – Display of S11 for Tile in Figure 9.

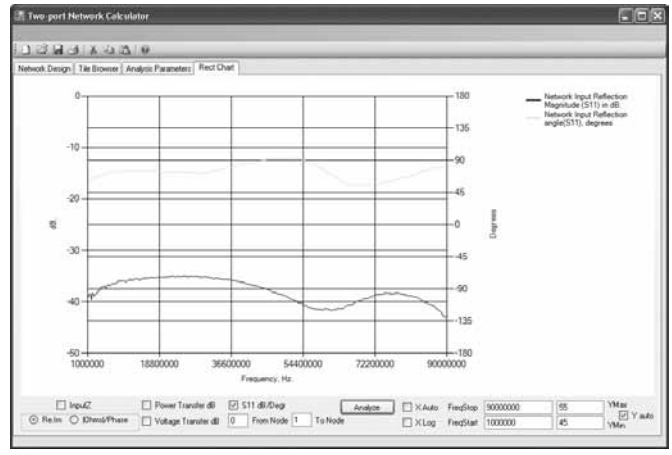

Figure 13 – The resulting S11 data after the compensating transmission line is added.

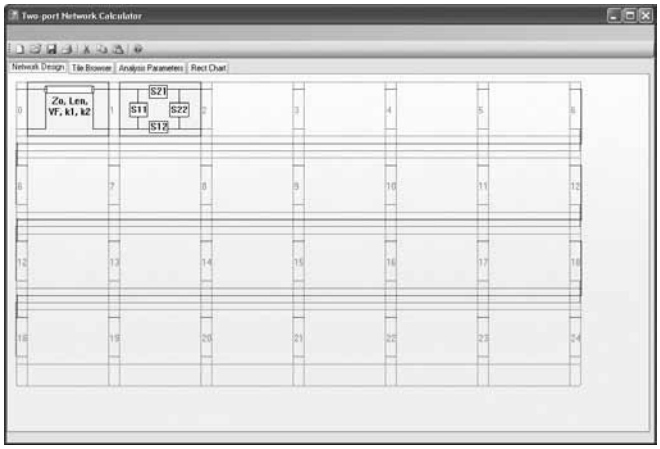

Figure 11 – Display of Network Design with a compensating transmission line added to the S parameter Tile.

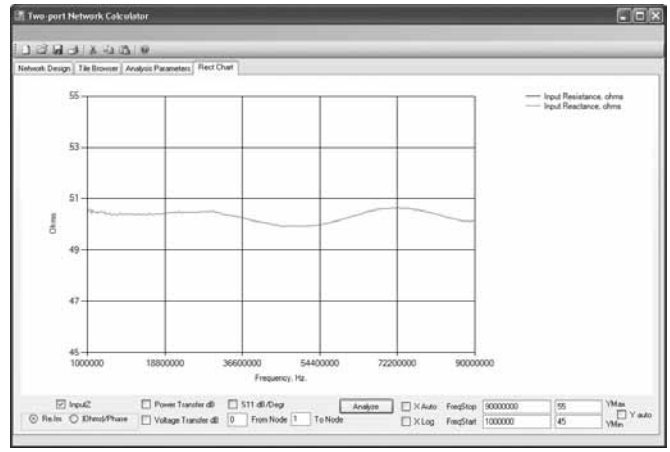

Figure 14 – A zoomed version showing the real part of the compensated S11 value.

You can experiment with the parameter values on the 'Tile browser' tab with the cable compensation tile selected, and look almost instantly at the impact to S11, Zin, etc. on the Rect chart tab. Of course just a negative-length cable can't compensate for some of the other measurement errors.

Example 3 – Phased vertical antenna

This example demonstrates the changing input impedance when trying to feed a two-element phased vertical array under varying conditions. It illustrates the impact of mutual coupling on the feed network design for the verticals. Due to the lack of available measured data, a 2-element quarter-wave vertical antenna array (each with 16 radials) over average ground was modeled in NEC2. The self (Z11) and mutual (Z12) impedances over a range of frequencies were derived, and an S2P then file created containing Z11, Z12, Z21, and Z22 for the two element array at each frequency. Because this is a theoretical model, symmetry was assumed, thus Z12=Z21 and Z11=Z22 which would not happen in a real network due to slight differences between the two elements.

We can then compute the input impedance of one of the antenna elements for a variety of feed conditions at the other element, and graph the various results. We need to place a dummy terminating resistor at the output of the array, typically a shunt element of large resistance when using a current-forcing condition for the output of the Z-parameter network — otherwise we would be attempting to force current into an open circuit (and thus creating infinite voltage) and the analysis indicates an error due to an attempted divide by zero; attempting to force zero current into an infinite impedance still results in the same error. It's easiest to just place a high-value shunt load at the output of the Z-parameter network as all practical drive conditions can be modeled without needing to change the topology.

The file "Monopole_z.s2p" contains the Z-parameter data for the modeled two-element array from 3.5 to 4.0 MHz in 50 kHz steps. We load it into the tools by selecting the "INSERT" Block Type Z, Frequency Table for Matrix Elements selections from the tile input form. Then we tell the tool which s2p file to read, and all the data points from that Z-parameter network are inserted.

Figure 15 shows the network connectivity for Example 3. In order to measure the input impedance of one antenna element with essentially zero coupling to the second element, the current forcing of the Z-parameter network is set to FALSE. Then the high value of the shunt network assures that virtually no current flows out of the network. This means that we will measure just Z11 of the network. Figure 16 shows the resultant self impedance of one of the antenna elements.

The resistance varies from just over 30 Ω (at 3.5 MHz) to just under 50 Ω (at 4.0 MHz) and the reactance varies from about

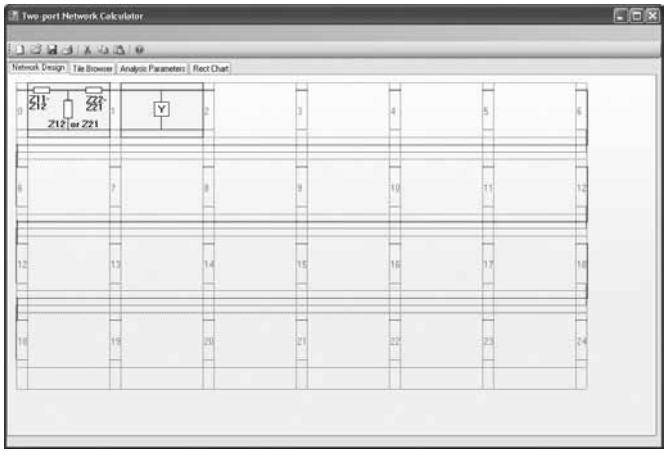

Figure 15 – The Network Design for Example 3.

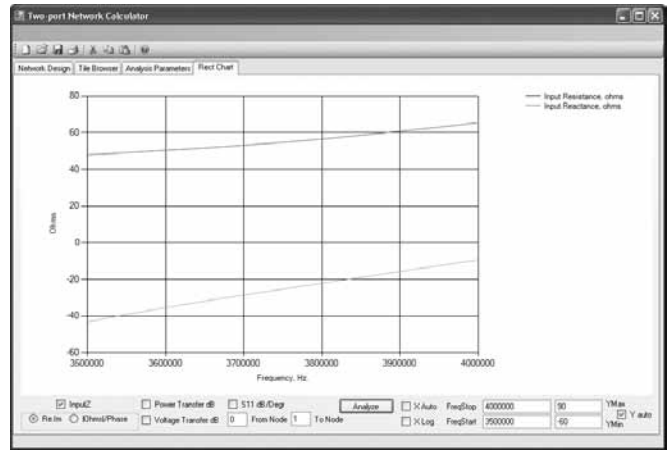

Figure 17 – The resultant input impedance of one antenna element when the two elements are fed in-phase with equal current.

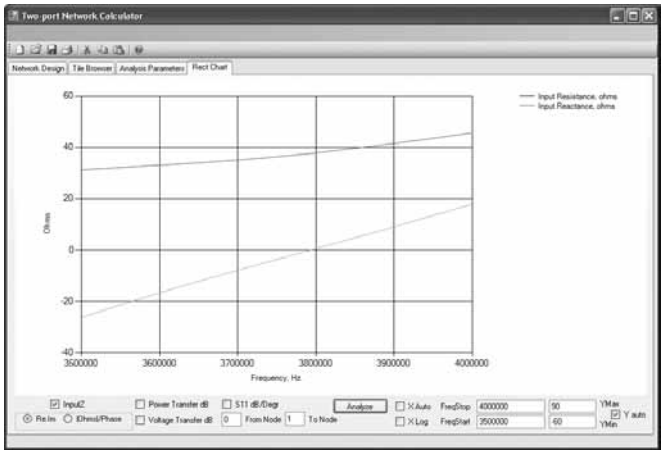

Figure 16 – The resultant self impedance of one of the antenna elements in Example 3.

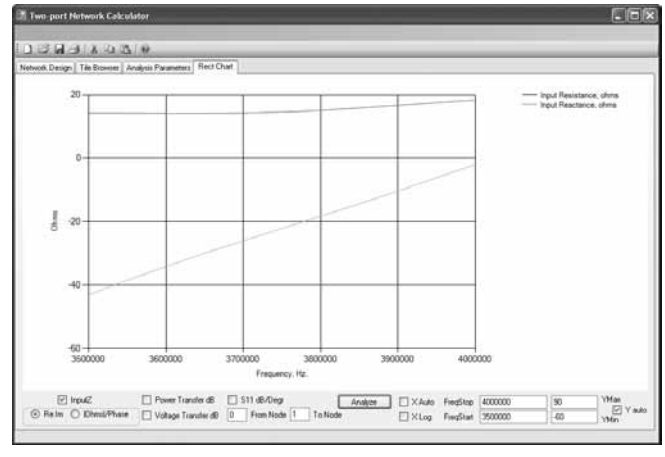

Figure 18 – The resultant input impedance when the two elements are fed 90-degrees to each other.

–25 Ω to +15 ohms, with resonance near 3.8 MHz. If we now set the current forcing function to TRUE, and set IMAG current ratio = 0 and the REAL forcing current ratio to -1 we are feeding the two elements IN-PHASE (the negative sign is because the two port currents are defined with opposite signs to allow chaining the calculations, recall Figure 1 above). The input impedance to the first element changes dramatically as a result.

Figure 17 shows the resultant input impedance of one antenna element when the two elements are fed in-phase with equal current. The input resistance now varies from 50 to 70 Ω, and the reactance from –45 to –10 Ω, and the element does not have a resonance between 3.5 and 4.0 MHz. If we feed the two elements with a 90-degree phase (forcing IMAG = –1, REAL = 0) the input impedance drops a lot. Changing the sign of Forcing_IMAG alters which element is fed in leading phase to the other, and the impedance similarly changes a lot.

Figure 18 shows the resultant input impedance when the two elements are fed 90-degrees to each other. In this case the input resistance of the fed element is around 15 Ω across the entire frequency range. If we change which element is leading in phase compared to the other, the impedances again change.

Figure 19 shows the resultant input impedance when the two elements are fed with the opposite –90-degrees phase relationship. Here the apparent resonant frequency of the fed element appears to be about 3.6 MHz, and the resistance varies from about 50 to 70 Ω.

The point of this exercise is to show that the feed networks for a phased vertical array must contend with quite different feed-point impedances as the phasing conditions of the array are altered. With these conditions now quantifiable, we can insert our desired phasing and impedance matching networks in front (to the left) of the Z-parameter network of example 3 and model whether it accomplishes delivering the desired current into element one of the array in the proper phase, and presents a proper load impedance to the transmitter, all over the desired frequency range.

For Further Work

This tool has already proven to provide extremely quick feedback for passive circuit and network ideas, sometimes much faster than trying to set up a model in the normal circuit simulator tools and providing the appropriate control statements. Rapid change of 2-port values via the tile browser allows quickly iterating through a number of design alternatives.

Further work could be done to extend this tool. Some possibilities are:

1. Provide a Smith Chart view of the input return loss, S11. While it is straight-forward to convert S11 to a reflection vector X and jY components, the Microsoft chart control does not readily allow the addition of the corresponding appropriate background axis for the polar chart.
2. Provide the ability to extend the Z-parameter matrix from 2-ports to a larger number of ports (such as 4-ports) that would allow the entry of mutual impedance and self impedance data for antenna arrays with more than 2 elements.
3. If the response of an entire network has been measured, but one (and only one) of the elements of the cascade of 2-port networks is unknown, it is possible to compute what the response of that unknown network must be.\

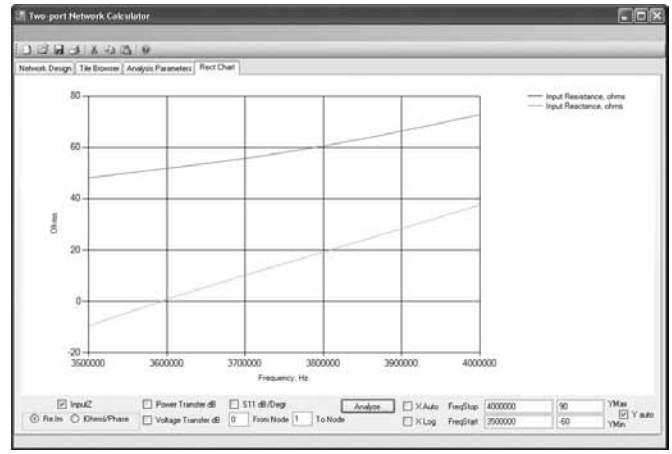

Figure 19 – The resultant input impedance when the two elements are fed with the opposite -90-degrees phase relationship.

Tom has been a licensed amateur for 40 years, and is a life member of the ARRL. He enjoys HF, SSB and RTTY operation, various technical topics in amateur radio mainly involving computers. He has a Bachelor's degree in Electrical Engineering from the University of California, Berkeley. His professional background is in high speed fiber optic transmission and switching equipment. Tom holds 12 patents and is a member of the IEEE and Internet2.

Notes

[1] *Microwave Engineering*, Third Edition. David M. Pozar, 2005 John Wiley & Sons, ISBN 978-0-471-44878-5.

[2] *Microwave Filters, Impedance-matching Networks, and Coupling Structures*, Matthaei, Young, and Jones, 1980, Artech House Books, ISBN 0-89006-099-1, Sections 2.06 through 2.13.

[3] Vertical Phased Arrays: Part 5, Forrest Gehrke, K2BT, *Ham Radio Magazine*, December 1983, pp 59-64.

[4] VK1OD transmission line loss calculator is located at: **vk1od.net/calc/tl/tllc.php** The k1 and k2 (frequency-dependent loss parameters) for a few cable types were extracted from this tool.

[5] Touchstone® File Format Specification Rev 1.1, Copyright © 2002 by the EIA/IBIS Open Forum. Available from **www.vhdl.org/pub/ibis/connector/touchstone_spec11.pdf**

[6] The Microsoft NET 3.5 SP1 Framework can be downloaded (as of September 2010) from: **www.microsoft.com/downloads/details.aspx?FamilyID=AB99342F-5D1A-413D-8319-81DA479AB0D7&displaylang=en**

[7] The Microsoft Chart Controls for .NET Framework 3.5 can be downloaded (as of September 2010) from: **www.microsoft.com/downloads/details.aspx?FamilyId=130F7986-BF49-4FE5-9CA8-910AE6EA442C&displaylang=en**

[8] Microsoft C# 2008 Express Edition can be downloaded (as of September 2010) from: **www.microsoft.com/express/downloads/**

[9] The Microsoft Visual Basic 2005 Power Packs 2.0 can be downloaded (as of September 2010) from:
www.microsoft.com/downloads/details.aspx?FamilyID=92faa81e-e9c1-432c-8c29-813493a04ecd&displaylang=en

[10] All files have been placed on the ARRL Web site at: **/www.arrl.org/qexfiles.** The file set contains the tool executable (EXE file), a help file, the tool source code (released under the GPL license), the three example design files, and the s2p files for the examples. It does not contain the Net frameworks or chart tools from Microsoft.

QEX