# A Low-Cost 100 MHz Vector Network Analyzer with USB Interface

*This article describes a low-cost vector network analyzer that operates from 200 kHz to 100 MHz, and connects to a personal computer using a USB 1.1 interface.*[1]

By Tom McDermott, N5EG, and Karl Ireland

**Introduction**

One of the more useful pieces of test equipment for designers and experimenters is the vector network analyzer (VNA). The VNA allows measuring the forward and reverse gain and phase response of a circuit, as well as the input and output reflection properties (complex impedance). Traditionally, the VNA has used s-parameters to describe the four properties of a two-port circuit being measured. The VNA is used to measure and adjust filters, coaxial cables, amplifiers, antenna input impedance vs. frequency, and so forth. A full VNA consists of two measurement sections: one in the forward direction that measures $s_{21}$ (forward gain and phase) and $s_{11}$ (input reflection magnitude and phase), and a duplicate circuit in the reverse direction that measures $s_{22}$ (output reflection magnitude and phase) and $s_{12}$ (reverse gain and phase, usually called reverse isolation). To save cost, many instruments only provide enough hardware to measure in one direction. Then, the device under test is physically reversed and the measurements rerun. Most properly, this simplified piece of equipment is called a transmission-reflection test set; but most of the time, it is still referred to as a VNA, the same as its big brother.

One significant difference between the two instruments is that a true VNA can be more accurately calibrated through true two-port techniques, whereas the transmission-reflection test set relies on precision standards (open, short, 50 Ω) to calibrate the reflection measurement. A two-port calibration based on the TRL (Through-Reflect-Line) technique can theoretically dispense with the need for precision load standards.

Another related piece of equipment is the scalar network analyzer. The difference between the scalar analyzer and the VNA is that the scalar analyzer does not include the additional circuitry to measure the phase component of the transmission and reflec-

[1]Notes appear on page 14.

265 Daniel Dr
Plano, TX 75094
**n5eg@arrl.net**

2517 Lawndale
Plano, TX 75023

tion parameters of the circuit under test, while the vector analyzer measures both the magnitude and phase components. Thus the VNA is quite a bit more complicated. The vector properties are often of great interest, such as when measuring the input impedance of an antenna, or the group delay of a filter, and thus the vector analyzer is a more useful instrument for many types of measurements.

This article will describe a vector transmission-reflection type of instrument; but we'll borrow the more grandiose title VNA, since most people are more familiar with that terminology. Fig 1 is a picture of the first prototype board—before the debug/fixes! The TX and RX BNC connectors are on the left and the Universal Serial Bus (USB) and DC power connectors are on the right. The directional coupler (metal case) is next to the TX connector. Figures 2 and 3 respectively show front and rear views of the completed unit.

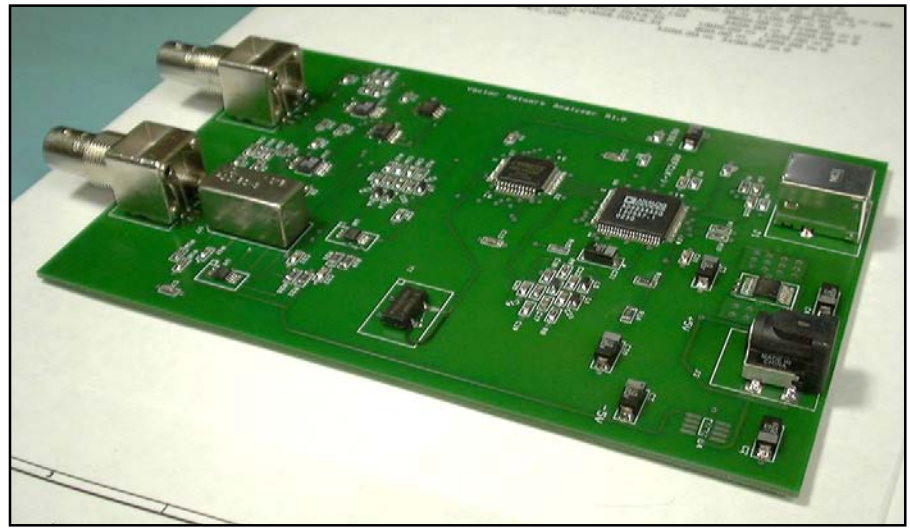

**Figure 1—Photograph of the assembled prototype board. The 4-layer board is approximately 4×6 inches.**


**Figure 2—Photograph of the front panel of the assembled unit.**


**Figure 3— Photograph of the rear panel of the assembled unit.**

### Overview of the Instrument

Commercially available VNAs are very expensive pieces of precision equipment, costing tens of thousands of dollars. These instruments provide tremendous dynamic range (approaching 90 to 100 dB), a high degree of accuracy, and many software options for manipulating and displaying data. It's possible to sacrifice some of the dynamic range and precision to save a lot on the cost and complexity of the measurement hardware. Today's personal computers, however, provide extensive ability to manipulate and display data for virtually no additional cost—just the time and effort of creating the software. So the instrument itself is kept as simple as possible by offloading much of the work to the host computer.

Fig 4 is a block diagram of the low-cost VNA measurement device. The equipment consists of a quadrature frequency synthesizer, a reflection measurement circuit, a transmission measurement circuit, a pair of phase/magnitude RF detectors (one for transmission and one for reflection), a multi-channel analog-to-digital converter (ADC), and a specialized USB-aware microprocessor.[2] Additionally, a +3.3 V regulator and a +5 to –5 V inverter provide the digital and analog supply voltages for the board.

The need for a quadrature synthesizer takes a little explanation. The recently released Analog Devices AD8302[3] device measures the magnitude ratio and relative phase difference between two RF signals (up to 2.7 GHz). The phase response is ambiguous, however. It is symmetrical

about 0°, at which point the phase accuracy is significantly degraded. This can be seen from Fig 5, a plot of the phase and amplitude detector output responses of the device vs the input phase difference, from the AD8302 data sheet. To resolve the phase sign, the VNA instrument switches the reference input to the detector between an in-phase (I) and quadrature-phase (Q) reference signal and measures the detector output for both conditions. Software on the host computer then resolves the correct phase quadrant between the two RF detector inputs. The low-pass filters are used to reconstruct the DDS output from the digital stair-step waveform produced. The Analog Devices AD9854[4] DDS is clocked by a 24 MHz sine wave; it then internally multiplies it up to 288 MHz with an on-chip PLL. This 288 MHz internal signal clocks the DDS frequency-generation circuits and the digital-to-analog (DAC) circuits on the chip. Significant aliasing of the output signal is observed on an oscilloscope even at an output frequency of less than 100 MHz. The two low-pass filters, one on I and one on Q, remove most of these aliasing and stair-stepping artifacts and produce clean sine waves in phase quadrature.

The RF detectors are broadband and have a total dynamic range of almost 60 dB, but with the restriction that the range is ±30 dB between the reference signal and the unknown signal. The performance of these detectors sets the dynamic range of the instrument. To achieve greater dynamic range, a much more expensive tuned-receiver configuration would be required. For a reflection measurement, the practical measurement limit is about 30 dB of return loss.

Two selectors are implemented with a pair of dual-input Maxim 200-MHz video amplifiers that are programmable by the target processor. One allows selection of whether the I or Q DDS reference signal is applied to both of the RF detectors. The other selects the reflected signal or a monitor signal (used only for debugging) to be measured by the reflection RF detector.

The transmission measurement process is very simple. A BNC connector (labeled RX) and terminated in 50 ohms is connected to one of the RF detectors. This measures the amplitude and phase of the signal received on the RX connector against the internal reference signals I and Q. The reflection circuit uses a Mini-Circuits 20-dB directional coupler[5] to derive the signal reflected by the load from the other BNC connector (labeled TX). The reflected signal produced by the directional coupler is the complex reflection coefficient gamma, Γ. The magnitude and phase of this reflection signal are derived by measurements against the internal I and Q reference signals.

In operation, the host processor (the PC) sends a command to the VNA over the USB port, triggering a measurement. In the command, the host sends a single frequency word as a 64-bit integer to the target. The target processor (the microprocessor on the VNA) then programs the quadrature direct digital synthesizer (DDS) to

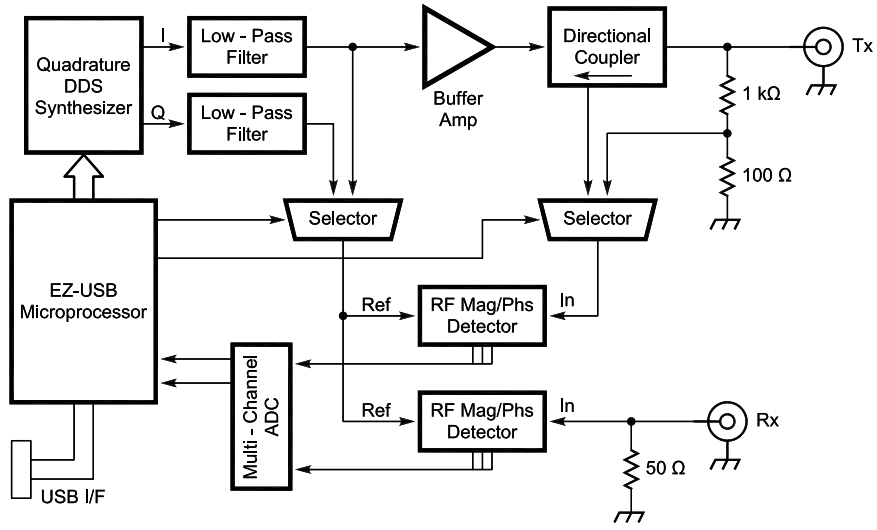


**Figure 4—Block diagram of the Vector Network Analyzer.**

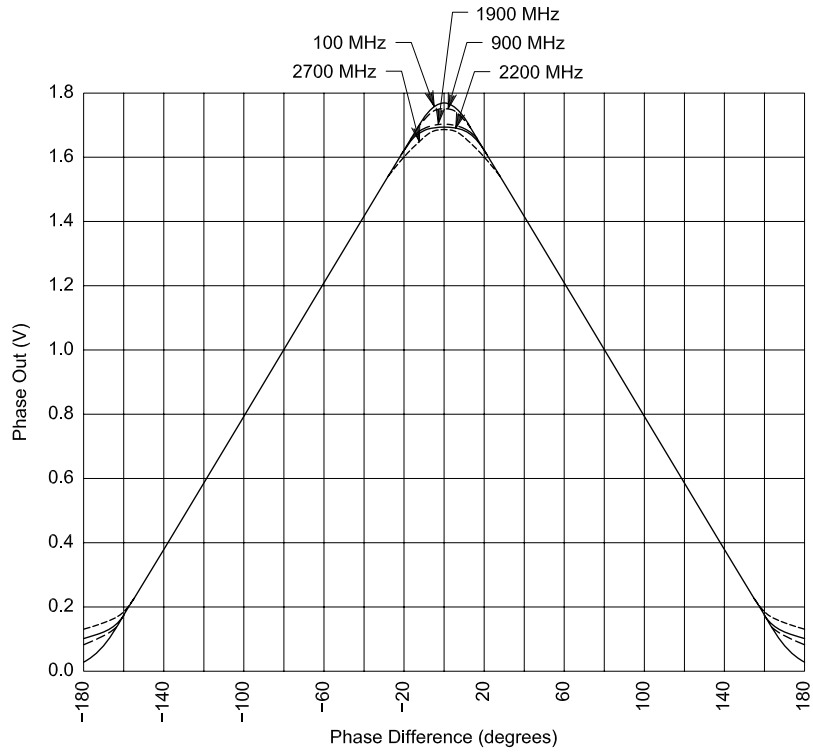


**Figure 5—Analog Devices AD8302 phase output response. The response is symmetric about 0°, and thus a quadrature technique is needed to resolve the sign of the phase angle. The accuracy is poor near 0° and 180° due to detector output saturation, necessitating compensation.**

that frequency. Once the frequency is programmed, the target processor makes a series of analog measurements. These measurements are made by using the input multiplexer of a multi-channel ADC chip to sequentially select and digitize the various RF detector analog output voltages. Additionally, the target processor switches between the I-reference signal and the Q-reference signal to the RF detectors and remeasures the detector's analog outputs. The target then repeats these two sets of measurements in the reflected direction. The RF detector's output a reference dc voltage, and the target processor measures these as well. In total, more than a dozen analog voltages are digitized and returned by the target processor. All of these measurements are assembled into a single 64-byte USB response packet.

Next, the host processor polls the target to see if it has the measurement ready. When it is ready, the target sends a response data packet back to the host containing the set of data samples that it has measured. The target is then ready to accept a new measurement request.

The host computer processes the measurement data set at each frequency and organizes the information into a useful display. A *Windows* application controls the USB device and provides a GUI interface that looks like a traditional VNA, displaying s-parameters in rectangular or polar (Smith chart) format, providing for calibration and error correction of the measurements, selection of the frequency sweep range, printing of the charts, and so on. The host application was written in *Microsoft Visual C++ .NET* version 2003.

The USB port is capable of supplying 100 to 500 mA (at the option of the host) to the target device. The VNA draws about 1 A, so it is powered from a +5 V dc wall-cube supply. On the VNA board, a Maxim inverter produces –5 V dc for the video selectors; a Maxim linear regulator produces +3.3 V dc for the digital parts. The voltage inverter needed to be decoupled with L-C filters to lower the noise level on the analog lines. Additionally, the video amplifiers required 10-ohm resistors and decoupling capacitors in the +5 and -5 volt lines to produce good low-level signal measurements of the return-loss signals. Power decoupling of the digital parts required just bypass capacitors. The DDS chip and the +3.3 V regulator are each heat sinked to the circuit board itself. A metalized pad on the top layer of the board is connected to the ground plane through a large number of vias. The solder mask is then opened on the top layer directly underneath each part, permitting each to be soldered directly to the circuit board itself. Without this heat sink, these two parts would overheat.

**Overview of USB**

Many newer computers do not include an EIA-232 serial interface. Many laptops now have only a USB interface for connection of the computer to outside devices. The USB interface supports a large variety of device types. It automatically identifies target devices, and then loads the appropriate *Windows* device driver. The driver is loaded when the USB target is plugged in, and unloaded when the target device is disconnected. USB can provide a minimum of 100 mA of supply current to the target device. In some cases it can supply more, up to 500 mA, but only 100 mA is guaranteed. This voltage is nominally +5 V, but resistive losses in the USB cable can drop it down to +4.4 V in the worst case.

When a USB device is first connected to a host and powered up, the host senses the additional connection via a 1.5 kohm resistor that the target asserts onto one USB data line. This process is called *enumeration*. Through a sequence of packets, the host learns the identity of the device type that connected, its configured capabilities, and which *Windows* device driver needs to be used to communicate with the USB target.

The target in the VNA is reprogrammable; it does not contain any non-volatile memory, just RAM. This allows the VNA target code to be changed quite painlessly. The VNA host application program downloads the executable code to the target each time it is connected and powered up—an extremely flexible arrangement. It permits different application loads to be sent to the VNA target. A minor drawback, however, is that the target then has to go through a somewhat more complex enumeration process.

The Cypress EZUSB microprocessor supports a minimum level of functionality in the processor at power-up when no code has yet been loaded to it—namely the ability to identify itself, to accept a download file, and to restart itself. After initially enumerating the target, the VNA host application software downloads the desired target code and then restarts the target. Then the newly downloaded code on the target takes control away from the Cypress-supplied default values and it *re-enumerates* itself by disconnecting then reconnecting the 1.5-k ohm resistor, this time with different capabilities identified. The new capabilities are specified by the newly downloaded and running target application. Cypress supplies a *Windows* device driver with their free development kit that is used by the VNA host application. Additionally, Cypress supplies a free C-language code framework for the target EZUSB processor that is USB 1.1, Section 9 compliant (named after the USB specification chapter). While there is a bit of a learning curve associated with it all, once understood, the framework makes developing a C-language application very easy—just the relevant USB endpoint handlers have to be coded. Everything else is done. The entire target application was developed in the evenings of about two weeks without requiring any debugger. Note that *Windows 95* does not support USB. (*Win95 OSR2.1* theoretically supports it, but what I've read claims that it is too buggy to use). *Win 98 Gold* (original edition) supports USB 1.0, and *Win 98 SE* (Second Edition) supports USB 1.1. All versions of *Windows* later than *Win98* also support USB 1.1. *WinXP* also supports USB 2.0. The 1.1 version of USB adds an interrupt *packet* type that is not used in this VNA target application.

The USB host sends packets to the target device and receives packets from the target device. The format of these packets is under software control. The USB interface is synchronous with exactly 1 ms time slots. During each slot, the host can send one control packet, one data packet, and one isochronous packet (useful for audio, for example). The target can return one data packet and one isochronous packet during the next slot. All USB target devices are polled. To retrieve a data packet, the host sends a control packet to the host requesting a response. The target then returns a data packet. Data packets can be up to 64 bytes in length, while isochronous packets can be 1023 bytes in length (1024 bytes in USB 2.0). A good text on USB is available.[6]

**Software**

The software consists of two independently compiled and linked software executables: the *host* software, and the *target* software. The host computer is a PC with an Intel IA32 processor (Pentium) running *Windows*. The target is a Cypress EZUSB processor (derived from the 8051) running just the target code image—no operating system or scheduler. Each

requires a different software development system.

*Run-Time Software*

The description of the tools below is of interest if you want the compile or change the software, which most people will not want to do. To just run the VNA software, the following binaries are needed:

- VNA host executable file,
- Microsoft .NET 1.1 Common Language Runtime (CLR) package,
- Host-compiled help file,
- VNA target (EZUSB processor) executable file,
- USB device driver file—supplied by Cypress: EZUSB.SYS,
- USB .inf file (driver information file) also supplied by Cypress. EZUSBW2K.INF.

*Target Software*

The target software was developed in C using the *AnchorChips* USB framework. This is available on the Cypress Web site for free in the EZUSB development download package. Literally, the framework is a couple of C module skeletons. You just need to fill in the various USB endpoint handlers and strip out anything you don't need, which isn't much. Getting this up and running on the target was really fast. After finding a hardware bug (two missing pull-up resistors), the framework came right up and ran. From that point it was a matter of adding functionality to the target through about four prototype builds. Each prototype added features over the previous build as we noted any errors in execution. These were usually quickly tracked down to the last added changes. Two tricky parts: The DDS chip comes out of reset at a slightly higher reset voltage than does the EZUSB processor, so some delay cycles were added to the USB code before it tried to initialize the DDS. The parallel interface was used on the DDS, since the chosen EZUSB device (AN2135) has an 8-bit parallel interface. Two I/O lines on the EZUSB are used for RD and WR lines to the DDS chip. Some have reported difficulty using the serial interface on the Analog Devices DDS chip, but the parallel interface was problem free. Secondly, the USB device is happier with exactly 64-byte bulk packets, so they are always sent this size. Interestingly, only one bulk-type packet is sent per frame, so sending 1-byte or 64 does not change the effective rate of packets per second because the frame rate does not change with packet size. Thus, for maximum throughput, 64 byte buffers should be used.

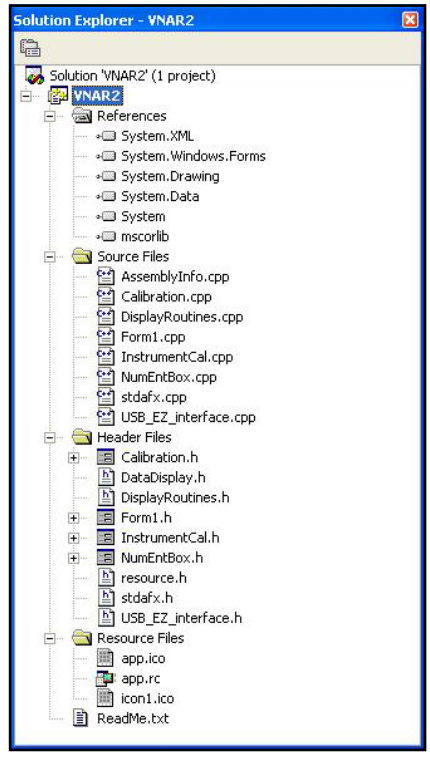


**Figure 6—Host software source tree.**

*Host Software*

The host software was written in *Microsoft C++ .NET 2003*. This is the most current version of the *Microsoft C++* compiler. The standard edition is available at a reasonable price. Microsoft used the name .NET (dot NET) to reference a lot of different products and services, which has confused people as to what it really is. The *C++ .NET* product contains a run-time library and a compiler/debugger. The common-language run-time library (CLR) provides a language-independent environment to execute the *Microsoft Intermediate Language* (IL), which is just in time compiled. It handles memory management and garbage collection (freeing up unused memory). This is pretty traditional for Basic, but novel for C and C++, which previously required the software developer to explicitly control the lifetime of objects and variables in memory. According to some, memory leaks and lifetime management constitute the largest category of programming errors in C and C++, and .NET is supposed to help minimize these types of errors. The same CLR supports Basic, C# (C-sharp) and C++, so mixing languages is possible. The 2003 edition of .NET supports a GUI designer for C++. After manually constructing several windows and dialog boxes in the 2002 edition, it's a relief to have that task partially automated. Programming in .NET is a lot different than standard C++. Reading the step-by-step tutorial book and going through the exercises is a must unless you are already familiar with .NET. Fortunately, the book and standard-edition compiler can be purchased together as a set at about the same price as just the compiler itself—if you do some shopping. The GUI designer has its own unusual way of constructing code. We chose to rewrite the initial host software prototype into a style better aligned with how that designer works. This made subsequent changes and especially additions a lot faster.

The VNA application consists of a number of software modules:

- The main program *VNAR2*, which holds the primary program control and display window
- Several dialog-box modules (calibration, numeric entry, etc.).
- Several utility modules that provide some general computations with complex numbers, and translate values into screen coordinates
- A wrapper module to encapsulate the interface to the USB device driver
- Several resource files (RESX) that hold the resources for the displayable windows

The source tree for the host program is shown in Fig 6. *Microsoft .NET C++* has an unusual approach to its file-naming convention: For the designer-generated code, the .h file holds most of the executable code, while the .cpp file contains a few headers. *Form1* is the main program, holding the top-level display and control window (Form1.h), and is also the entry point to the .NET program (Form1.cpp). For non-designer generated code, we chose to adopt the opposite convention by declaring class, module, and subroutine interfaces in .h files, putting the class methods and subroutines in the .cpp files. Old habits die hard.

*Calibration* contains the interface to the test-fixture calibration routines and the computations for deriving a virtual s-parameter error matrix equivalent to the fixture's influence on the measurement. *DisplayRoutines* contains code to derive screen coordinates for the display window given rectangular or polar display coordinates. In addition, it contains *FrequencyGrid, CalibrationDatSet,* and *Detector class* definitions and methods. The frequency grid allows decoupling the number of data points measured by the VNA in a single sweep from the display resolution of the screen display itself. *InstrumentCal* contains the menu interface and methods needed to perform the calibration of the individual AD8302 detectors.

The *NumEntBox* routine contains methods to allow direct entry of the start and stop frequencies or levels when a large numeric indicator is double-clicked. The EZ_USB_interface contains a wrapper for interfacing to the Cypress *EZUSB.SYS* low-level device driver.

In addition, several support data files are needed:

- The binary executable code for the target, which the host application reads and downloads to the target through the USB interface
- A calibration data set file. The application writes this file with calibration parameters after a calibration is performed. It can also be read by the application to retrieve previously saved calibration parameters. The data in the file is stored in binary format.
- Export files. The host software has an option to generate measured $s_{11}$ and $s_{21}$ values in a tabular format as a text output file. There are several standard file-header and polar/rectangular formats, which support several popular simulation and CAD packages. This allows measured device parameters to be directly imported to those CAD programs as an s-parameter model.
- A compiled help file was generated to provide help for the VNA; it's compatible with the Microsoft HTML help engine (which requires IE4.0 or later).

The interface to the USB driver was—and is—pretty tricky. We're not altogether satisfied with the result. The driver was written pre-.NET and it cannot be invoked directly by a .NET program. Microsoft provides a P/Invoke command that marshals data into a format compatible with direct API calls to *Windows*. The calls have to be prototyped so the compiler knows how to call them; hence, each one was done by hand. (There are about 10 calls in the driver.) The variables passed to the driver are somewhat complex, and they have to be described to the compiler as well. To describe these variables, several header files from the *Windows* Device Driver DDK need to be included. Microsoft distributed the DDK beta version free of charge, but now charges for the completed version. Worse yet, #*including* the header files breaks almost all .NET code.

The strategy was to build a .CPP and an .H file for the wrapper. The .H file describes the interface to the wrapper to the compiler and other .NET modules. The .CPP file includes the actual code, as well as the #*include* files. Then that .CPP file is compiled without the run-time library, and that .CPP file is not exposed to other .NET files, only the linker sees it; however, other modules know how to call the wrapper because of the .H descriptions which they #*include*. This worked in the 2002 edition, but broke in the 2003 edition. The 2003 edition requires that all class methods and class variables, *whether public or private*, be declared in the CPP file and be identical to those exposed in the .H file. The #*includes* needed to do this immediately to break the compilation of all the other .NET modules. A temporary solution was to bury the variable definitions inside the methods of the wrapper class as *automatic* variables. This way they are never exposed as an interface or class variable of the wrapper itself. Unfortunately, this makes the saving of driver state in a private-class variable difficult, since useful variables cannot be declared.

We ended up throwing away much of the wrapper code that made a clean wrapping of the driver, and ended up just re-acquiring driver data inside every method every time it is called. There are probably several better ways to solve this significant problem. The byproduct of our approach is that you need to have several of the header files from the DDK to compile the application, since we cannot distribute them. It is probably possible to write the driver wrapper as a .DLL in an older version of C++ and avoid many of these issues.

*USB Device Driver and Initialization*

USB is a plug-and-play interface in *Windows*. To associate a device driver to a device, *Windows* needs to know the ID of the USB device that is installed and which device driver to connect to it. When the VNA is plugged into a computer for the first time, the *new-hardware-detected wizard* is called by *Windows*. It causes the VNA to enumerate without any code loaded into it. The Cypress EZUSB will answer this enumeration request with the VID (Vendor ID) and PID (Product ID) code of the EZUSB processor. Then *Windows* will pop up a dialog box and ask the user to choose between supplying a disk file or letting *Windows* search for the file. In the VNA package, an .inf file (driver information file) supplied by Cypress tells windows to use the EZUSB.DRV driver file when the VID_PID device is connected. The wizard makes an entry in the registry associating the VID_PID with the device driver so that the wizard and user dialog box don't have to be displayed again. One needs to copy the EZUSB driver into the driver directory in *Windows*, which is different for different versions of *Windows*, to use it. (In *WinXP*, it's *Windows*\System32\Drivers.) Having done that the first time, *Windows* can find and load the driver anytime one plugs in that USB device.

The USB interface to *Windows* is fairly complex, and writing device drivers is not a trivial task. So the driver supplied by Cypress is used for the VNA. It's a very low-level driver, but we've had good results with it so far. It does have a few bugs. Those that are known have been avoided in the wrapper module code.

Other than the wrapper problem, very few issues were encountered in constructing and debugging the NET code. The 2003 version of the C++ .NET debugger is a lot more stable than the 2002 version.

*HTML Help*

A help package was generated for the VNA using the *Microsoft HTML Help Compiler 1.4*, available for free from the Microsoft Web site. There's also a good tutorial available on the Web[7] about how to use this tool to build help and context-sensitive help. Each help topic is actually an HTML page; the compiler links all the pages into a single compressed binary file using the .chm suffix along with the table of contents, index, and search tags. The tags can point within the file or across the Internet, but we decided to leave all of them within the compiled file so that an Internet connection is unneeded to use the help file — or the VNA! The computer does require *IE 4.0* or later to view the help topics since they are HTML. This type of help interface should look familiar to most *Windows* users.

Available source code can be found at **www.arrl.org/qexfiles**.

**Calibration and Error Correction**

All s-parameter measurements made by the VNA need to be calibrated. The actual measurement occurs on the printed circuit board itself, which is removed from the device under test (DUT). The connecting leads and components on the circuit board, connectors, interconnecting cables to the DUT and measurement imperfections of the components transform the measurement values. Additionally, the VNA does not have an absolute amplitude or phase reference, but only a relative reference. In practice, error correction and calibration are combined into a single compensation to the measurement. Different cable lengths may be needed for different measurements

and the cable length significantly affects the measurement. Thus, calibration is usually required for each different test-fixture setup. The host program has a mode to take a calibration set and store it with a descriptive file name to be recalled as needed.

Calibration involves compensating the phase delay of the instrument and the interconnecting cables. It also compensates for amplitude variation with frequency. Because the VNA is really only half a network analyzer (forward direction only), the TRL technique cannot be used. The TRL technique can avoid the need for precision standards. Instead, calibrations for $s_{21}$ and $s_{11}$ are derived independently using accurate RF loads—well, we hope they're accurate.

*Transmission Calibration ($s_{21}$)*

The transmission calibration ($s_{21}$) is easier to understand. There are two interconnecting cables—one from the TX connector to the DUT, and another from the DUT to the RX connector. In the through-calibration mode, these two cables are disconnected from the DUT, and directly connected together with a *very* short connector called a "bullet." The software makes the assumption that the bullet has no length and is a perfect impedance match. That would be a poor assumption at microwave frequencies, but at 120 MHz, it's not too bad. The transmit signal is then swept across the frequency range of the instrument and the received magnitude and phase recorded for 1024 different discrete frequencies. The phase delay and amplitude received are computed and stored in a table by frequency. Obviously, longer cables would have greater phase delay and perhaps more attenuation at higher frequencies. The RX port is terminated on the circuit board with a fairly accurate 50-ohm resistive load. Similarly, the TX port is sourced from a fairly accurate 50-ohm resistive pad at the output of the TX buffer amplifier, resulting in a good source match. To apply the transmission calibration against the measured DUT data, the measurement of $s_{21}$ of the DUT is divided by the recorded calibration value at the same frequency. Since both these data are complex numbers, this involves using complex division.

$$s_{21}^{\text{actual}} = \frac{s_{21}^{\text{measured}}}{\text{Cal}_{21}} \qquad \text{(Eq 1)}$$

If the measured value of $s_{21}$ were exactly the same as the stored calibration constant, then $s_{21}^{\text{actual}}$ would be $1.0+j0$, signifying that the DUT had no gain, loss, or phase shift. Since the DUT does have $s_{21}$ imperfections, Eq 1 just derives the difference between the DUT measurement and the calibration measurement, thus removing most instrument and interconnection cable errors.

*Reflection Calibration ($s_{11}$)*

The reflection calibration ($s_{11}$) is a bit more complex. Three calibration runs are made using different loads attached to the end of the TX cable that has been removed from the DUT: 1) with a 50-ohm load, 2) with a shorted load, and 3) with an open load. The accuracy of these calibration loads directly affects the correction accuracy of subsequent measurements. Table 1 shows the value of the reflection coefficient $\Gamma$ for each load type *directly at the load itself,* assuming that they are perfect loads, which they are not. The VNA instrument measures a value for $\Gamma$ that is rotated in phase and attenuated in magnitude by the interconnecting cables, PC-board traces, and components in the cable and instrument itself. The calibration computation involves de-rotating the measured value by the calibration phase delay, and adjusting the measured magnitude by subtracting the reflection calibration amplitude component. This calibration computation must be done at each frequency.

The three load components lie along the center horizontal straight line on a Smith chart, a short being at the left, 50-ohm in the center, and an open on the right. See Fig 8. The calibration measurement will indicate instead three points on a line that is rotated from the horizontal position by the amount of phase delay in the transmission cable plus the delay internal to the instrument itself. The amount of rotation is frequency-dependent.

After de-rotation to the horizontal position, the three measurements will not lie exactly on the ideal positions on the Smith chart listed in the table, but will be offset by the remaining measurement errors. Before the de-rotation, the amplitude measurement of the DUT is scaled differently on the left and righthand side of center (low and high resistive components, respectively) using the short and open cali-

**Table 1**
**Reflection coefficient for terminated, shorted, and open loads in polar and rectangular coordinates.**

| $\Gamma$ | *Polar Coordinates* | | *Rectangular Coordinates* | |
|---|---|---|---|---|
| *Load* | *Magnitude* | *Phase* | *Real* | *Imaginary* |
| Short | 1 | ±180° | –1 | 0 |
| 50 Ω | 0 | arbitrary | 0 | 0 |
| Open | 1 | 0° | +1 | 0 |

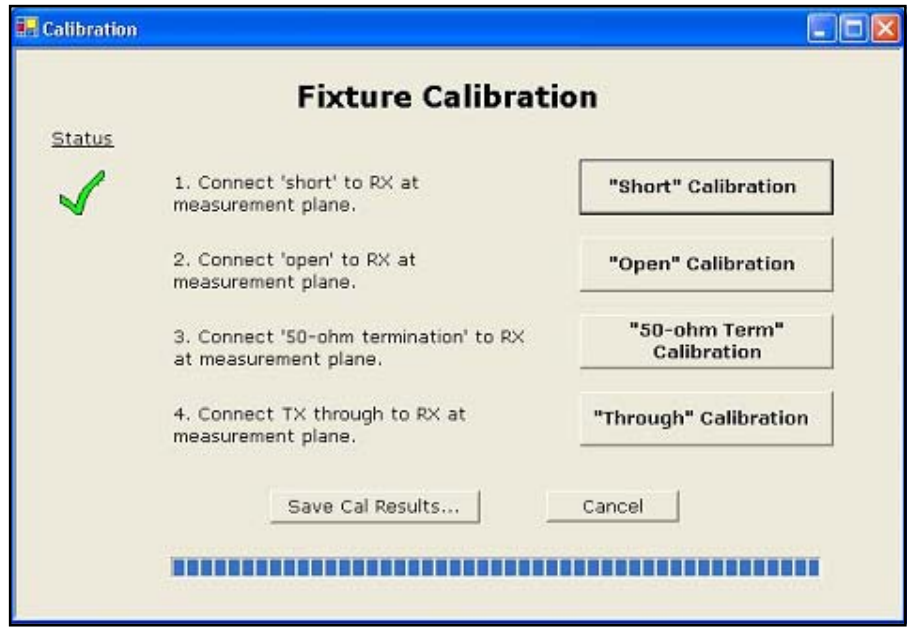


**Figure 7—Fixture Calibration Menu.**

bration values as full-scale (left and right, respectively), and the 50-ohm value as a zero scale constant. This is because some measurement errors are dependent on the impedance seen by the directional coupler itself. After this adjustment, the value is de-rotated by the calibration reflection phase delay value, which results in the final value for Γ. Fig 7 shows the fixture calibration menu, with the short calibration completed.

We can place reflection measurement errors into three types:

- $E_d$ Directivity error—this error is caused by leakage in the directional coupler and impedance mismatches between the VNA and the DUT reflecting additional energy.
- $E_s$ Source mismatch error—this error is caused by impedance mismatch between the signal source and the directional coupler. In the VNA, an attenuator pad is used between the buffer and the directional coupler to provide a reasonably good match, but it's not perfect.
- $E_t$ Tracking errors—these are errors in the measuring circuits that provide the magnitude and phase values and include test-cable artifacts.

In his book on microwaves, Pozar describes how to analyze and derive these reflection measurement error terms for the one-port calibration.[8] The technique models the three errors as a virtual s-parameter error matrix inserted between the VNA and the DUT.

Refer to Fig 9 for s-parameter definitions. In an s-parameter matrix, the forward input voltage, $a_1$, reverse input voltage, $a_2$, forward output voltage, $b_2$, and reverse output voltage, $b_1$ are related by the s-parameters:

$$b_1 = a_1 s_{11} + a_2 s_{12}$$
$$b_2 = a_1 s_{21} + a_2 s_{22} \quad \text{(Eq 2)}$$

Thus the voltage emerging from the network lefthand side, $b_1$ is the sum of the input reflection property of the network times the input voltage $a_1$, plus the reverse isolation of the network times the voltage supplied into the network from the right-hand side, $a_2$. We substitute the error terms into the s-parameters of the virtual error matrix:

$s_{11}$ is the source directivity error, $E_d$.

The product $s_{21}s_{12}$ is the reflection tracking error, $E_t$, which we can allocate between the two terms as we like—in this case, setting $s_{21}$ to unity, and $s_{12}$ to $E_t$ (see Fig 10).

$s_{22}$ is the source matching error, $E_s$.

If we place the DUT that has an actual reflection $s_{11}^{\text{actual}}$ (Γ actual) on the output of this virtual S-parameter error network, then from the input of this error network we measure $s_{11}^{\text{measured}}$ (Γ measured). It can be shown that the actual measured value of the DUT looking through the error matrix is:

$$s_{11}^{a} = \frac{s_{11}^{m} - E_d}{E_s(s_{11}^{m} - E_d) + E_t} \quad \text{(Eq 3)}$$

The derivation requires a bit of algebra and Pozar covers this in his book. We have three unknown terms and three measurements, so it is possible to solve for the three error terms. The solution to the first term is:

$$E_d = s_{11,\text{load}}^{m} \quad \text{(Eq 4)}$$

This is because the reflection from a perfectly terminated load is zero. Thus $a_2$ of the virtual error matrix is zero, and the $E_s$ and $E_t$ terms fall out of the $b_t$ voltage term (reflected signal). Similarly:

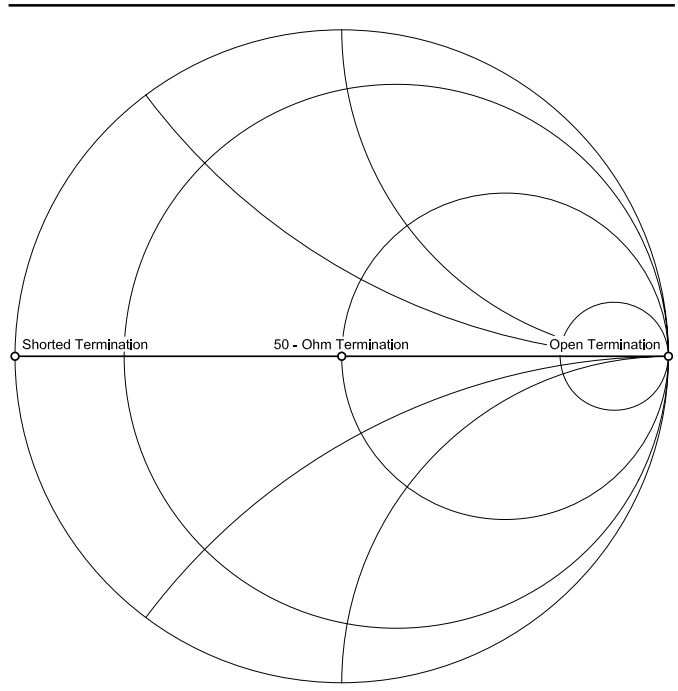


**Figure 8—Smith Chart plot of ideal calibration load points after computationally removing the test cable transmission line length.**

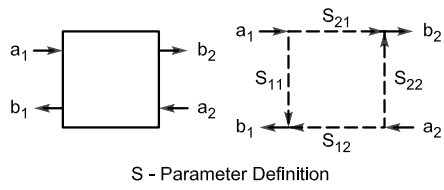


**Figure 9—S-parameter definition.**

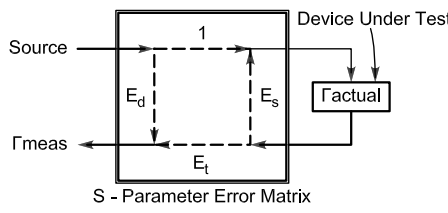


**Figure 10—S-parameter error matrix.**

$$E_s = \frac{2s_{11,\text{load}}^{m} - \left(s_{11,\text{short}}^{m} + s_{11,\text{open}}^{m}\right)}{s_{11,\text{short}}^{m} - s_{11,\text{open}}^{m}} \quad \text{(Eq 5)}$$

and

$$E_t = \frac{2\left(s_{11,\text{open}}^{m} + s_{11,\text{load}}^{m}\right)\left(s_{11,\text{short}}^{m} + s_{11,\text{load}}^{m}\right)}{s_{11,\text{short}}^{m} - s_{11,\text{open}}^{m}} \quad \text{(Eq 6)}$$

$E_t$ includes the phase delay and attenuation of the interconnecting cable between the VNA and the device under test. Thus, different calibration files need to be collected for different cabling setups, since $E_t$ will be different in each case.

D. Pozar's book on microwave engineering covers the

topic of VNA calibration and compensation. In the host program, the calibration routine guides the user through four test setups, and makes four measurements—short, open, load, and through. All are swept over the complete frequency range of the analyzer. It then derives the values of $E_d$, $E_s$, and $E_t$ over frequency, and also derives the transmission calibration term discussed previously. The three reflection error terms, as well as the raw reflection calibration data, can be plotted on the polar chart to give an idea of what they look like. The calibration routine stores a table of the three derived error terms, along with the $s_{21}$ transmission calibration value, each at 1024 frequencies, in a file on disk. The name of the file can be selected at the time the calibration file is saved.

The calibration function also allows loading a named calibration file instead of having to repeat a previous calibration run, which saves a lot of time. This makes it easier to have different calibration files for each test setup. The raw data is also saved to the file allowing rerunning the calibration and changing only one measurement type (for example "through"), and then saving the cal data set. The raw data will be overwritten by any new measurements while preserving raw data not re-collected. Then the cal file can be saved again. This updates the cal data set with only the new data type. This saved a lot of time during the debug cycle.

Fig 11 shows the measurement of a shorted cable without fixture calibration applied. Without calibration, the short describes a circle near the edge of the polar chart, illustrating that the connecting cable between the short and the VNA rotates the phase angle of the short. After the fixture calibration is applied (Fig 12), the cable phase rotation has been removed and the short shows up as point on the left side of the polar chart, with all frequencies being measured falling on the same point (0 ohms).

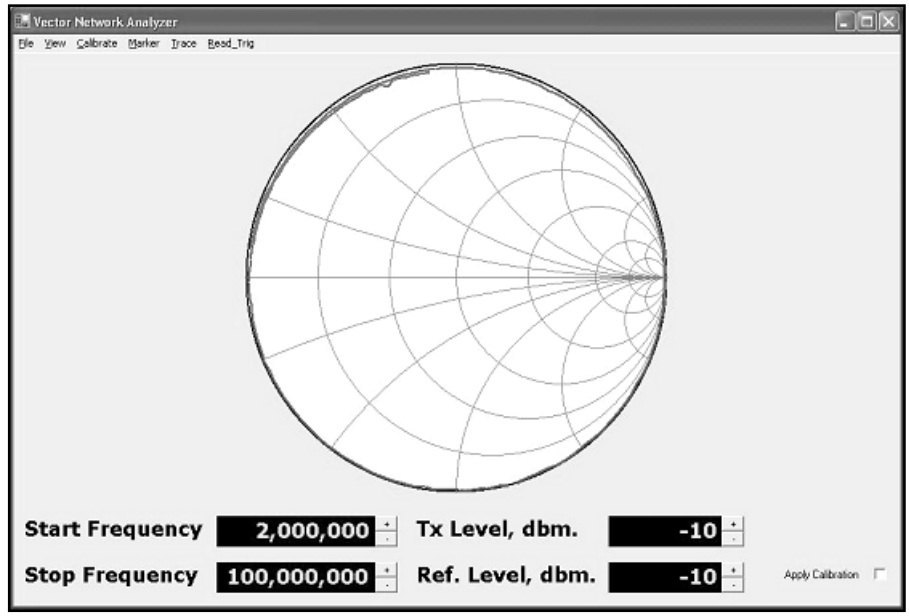


**Figure 11—Shorted cable without fixture calibration.**

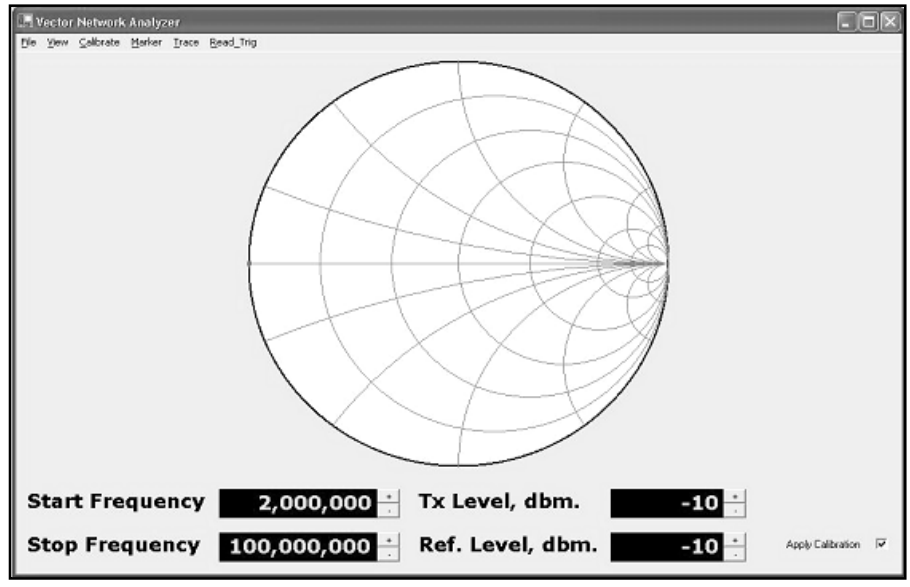


**Figure 12—Shorted cable with fixture calibration applied.**

*Phase Measurements & Detector Calibration*

At first, a simple technique was used to translate the phase detector analog measurements measured by the AD8302 from voltage to phase, in degrees, with respect to the reference signal. The arctangent of the I-measurement divided by the Q-measurement, with appropriate correction for the measurement quadrant, was utilized. Unfortunately, this simple technique suffers from a significant error because of phase-detector saturation in the AD8302. Particularly near 0 V output, the phase-detection gain drops non-linearly. The range near positive output (+1.8 V) is less saturated, but the actual positive voltage peak value varied a lot from device to device. In one case, it significantly exceeded the value of $V_{ref}$ from the specific device being measured.

The nature of the error causes a somewhat sinusoidal error in the phase reading— the phase varies both above and below the actual value. This error was particularly significant in group-delay measurements, since the group-delay routine differentiates the phase slope at adjacent frequency sample points. Fig 11 shows the

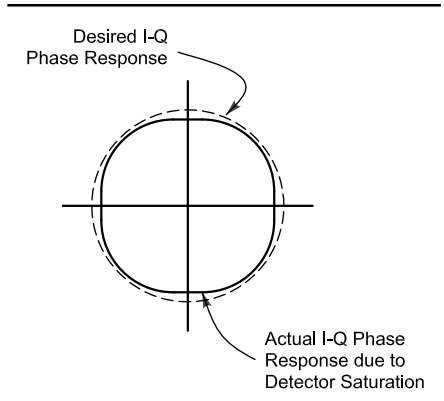


**Figure 13—Error of phase reading over 360° caused by detector saturation near zero and +2.0 V.**

slightly oblong shape of the phase response caused by detector saturation.

A new phase calibration routine was devised that uses a length of coaxial cable as a reference element. This calibration result is used to compensate the converted phase values. A fixed length of cable has a constant time delay, and thus has a linear phase change with frequency. The detector calibration routine measures a small fixed length of cable and compares it with the actual phase-detector measurement. The software then stores a table containing the difference, in degrees, between the measured phase and the phase calculated by linearly interpolating the cable phase. That is, it finds two points in frequency that differ in phase by 360° because of the calibration cable, then derives the phase at any frequency in between using a linear fit to the two endpoints. This phase difference is stored in a table with one-degree resolution.

A separate table is generated for the transmit and receive AD8302 phase detectors. This same calibration routine also finds the midpoint of each phase detector's output range by finding the detected output voltage— approximately +0.9 V, but slightly different from device to device—wherein the reference cable produces 0°, 180° and 360° phase differences at the same output voltage. The process converges in about three iterations. The algorithm thus fully characterizes each AD8302 by its individual performance.

This phase calibration process is done separately for the TX and RX phase detectors, since they are different physical devices and have different errors. The detector constants and correction tables are stored in a small detector calibration file in the program startup directory—the directory where the VNA program starts execution. The detector calibration does not change with the test setup; thus it only needs to be measured once, then the values are relatively permanent. The software automatically looks for and loads this detector calibration file on startup and provides a warning message to run the one-time detector calibration routine if the file cannot be found. A menu option to run the detector calibration allows the initial generation and storage of the constants in this file using a couple of short test cables. Running the detector calibration routine overwrites any existing detector cal file.

A small residual error remains after this improved calibration technique, which can be seen most easily when measuring group delay. Fig 14 shows the phase response of an approximately 3 m long cable with and without this error correction.

Fig 15 shows the transmission amplitude response, phase and group delay of a 3 m long cable. The group delay scale is 10 ns per division, thus this test cable has

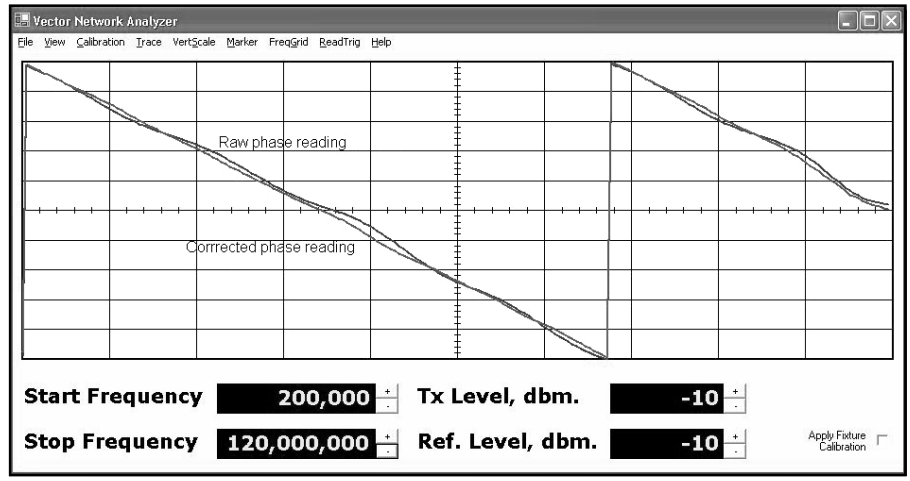


**Figure 14—Phase response before and after detector calibration.**

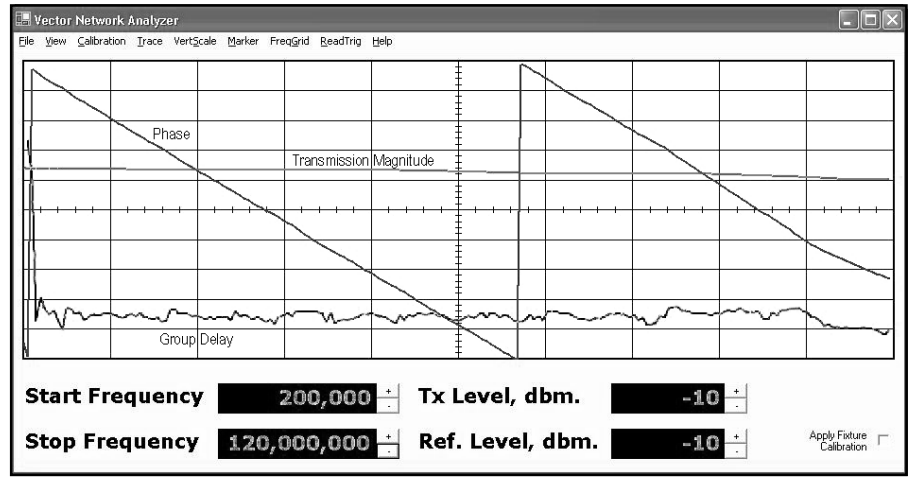


**Figure 15—Transmission Magnitude, Phase and Group Delay of a 3 m test cable.**

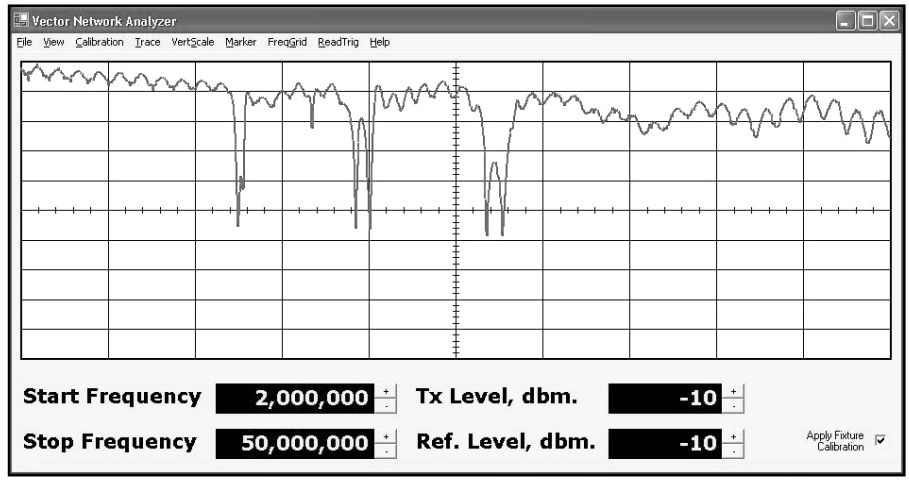


**Figure 16—Return loss of KT34XA antenna through 300 feet of hardline. Vertical scale is 5 dB/div.**

about 14 ns of delay. A perfect result would be a straight horizontal line for group delay. This measurement was made by differentiating the phase at each incremental frequency sample, thus it produces the noisiest measurement. When the delay differentiation is spread across several samples (called the *frequency aperture* of the group-delay measurement), the errors are significantly filtered. The result shown in the figure we consider to be quite good. Seven different detector-calibration algorithms were tried to get to this point. The drop in delay above 100 MHz is caused by reference-signal degradation, in turn caused by the DDS's reconstruction low-pass filter cutoff limitation described earlier.

## Software Source Code

VNA source files for both the target processor and the host processor have been made available in open format on the Web for amateur and non-commercial use. It's hoped that readers will add useful and interesting functionality to the VNA and make any changes similarly available for amateur and non-commercial use.

### *Software Tool Sources*

The development software can be purchased inexpensively. The target is written in C and 8051 assembly. Cypress provides a limited-capability free copy of the Keil compiler in its development package. Another compiler package, Reads-51, which is free for non-commercial use can be downloaded from the Web,[9] but the target software has not been ported to that package yet. The host software is written in C++. The Microsoft compiler can be purchased as part of the Microsoft C++ .NET 2003 Step-by-Step Deluxe Learning Edition, which contains both the excellent Step-by-Step book and the C++ standard-edition compiler, bundled together.[10] The set can be found at local bookstores, or at a discount from several Web-based bookstores. We found it commercially available on the Web for less than $80. The *HTML Help compiler v1.4* is free from the Microsoft Web site, and a good tutorial is available by Char James-Tanny, as was mentioned earlier. The installer was generated using the Jordan Russell INNO installer program, which is free for any use (commercial or not) and is very simple to use.[11]

## Applications

There are a large number of applications for a VNA. We'll examine a couple of them here.

### *Antenna Return Loss (SWR)*

One of the most common measurements made is the standing wave ratio of an antenna. A low SWR means that the antenna input impedance is close to that of the measuring reference impedance. *Return loss* is the common term for an equivalent measurement, that being the ratio of the reflected voltage to the incident voltage, usually expressed in dB. To convert from return loss to SWR, the following formulas are used (S = SWR, ρ= reflection coefficient, RL = return loss in dB):

$$\rho = 10^{-\left(\frac{RL}{20}\right)} \qquad \text{(Eq 7)}$$

$$S = \frac{1+\rho}{1-\rho} \qquad \text{(Eq 8)}$$

So, for example, a return loss of 20 dB is a reflection coefficient of 0.1, and an SWR of 1.22. A return loss of 10 dB is a reflection coefficient of 0.316, and an SWR of 1.92.

The following measurements show the magnitude of the return loss versus frequency for a KT34XA antenna at the end of 300 feet of hard-line cable. The resonance points are clearly visible. The phase part of the impedance is that at the ham-shack end of the cable, not at the antenna. Fig 16 shows the return loss at 5 dB/div of the antenna swept from 1 MHz to 50 MHz. The 20-m, 15-m and 10-m

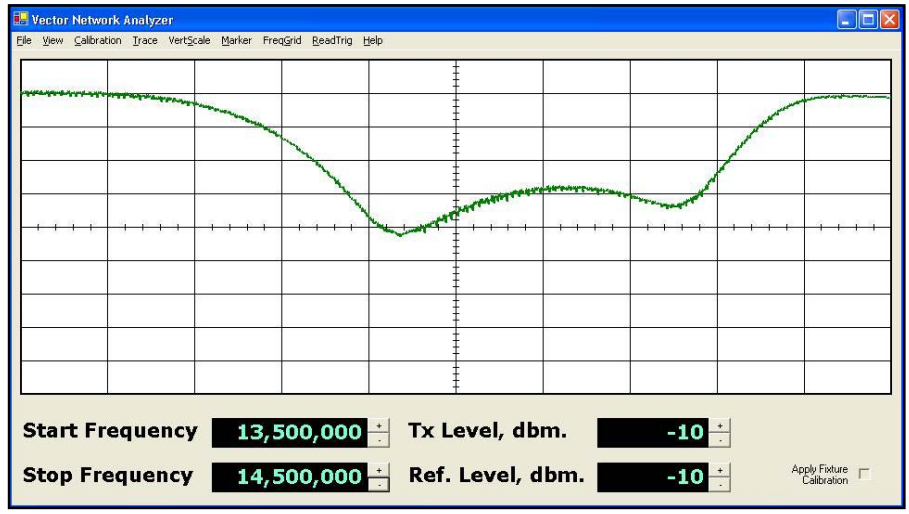


**Figure 17—Return loss of Fig 16 from 13.5 MHz to 14.5 MHz. 26 dB return loss (best case at 13.94 MHz) is an SWR of 1.105 (at the ham-shack end of the feed line). Vertical scale is 5 dB/div.**

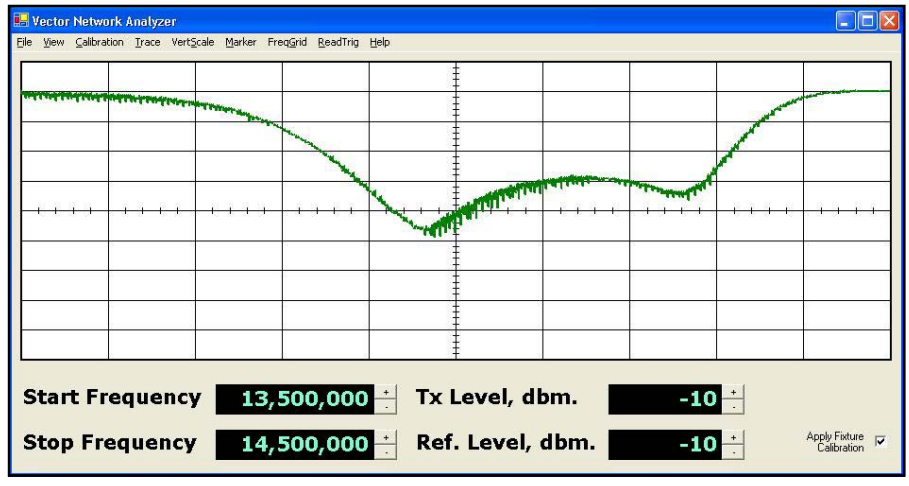


**Figure 18—Return loss of Fig 17 with fixture calibration applied (the fixture is the instrument itself and one m of cable). Note the best-case return loss is now 28 dB at a frequency of 13.96 MHz (an SWR of 1.083). Vertical scale is 5 dB/div.**

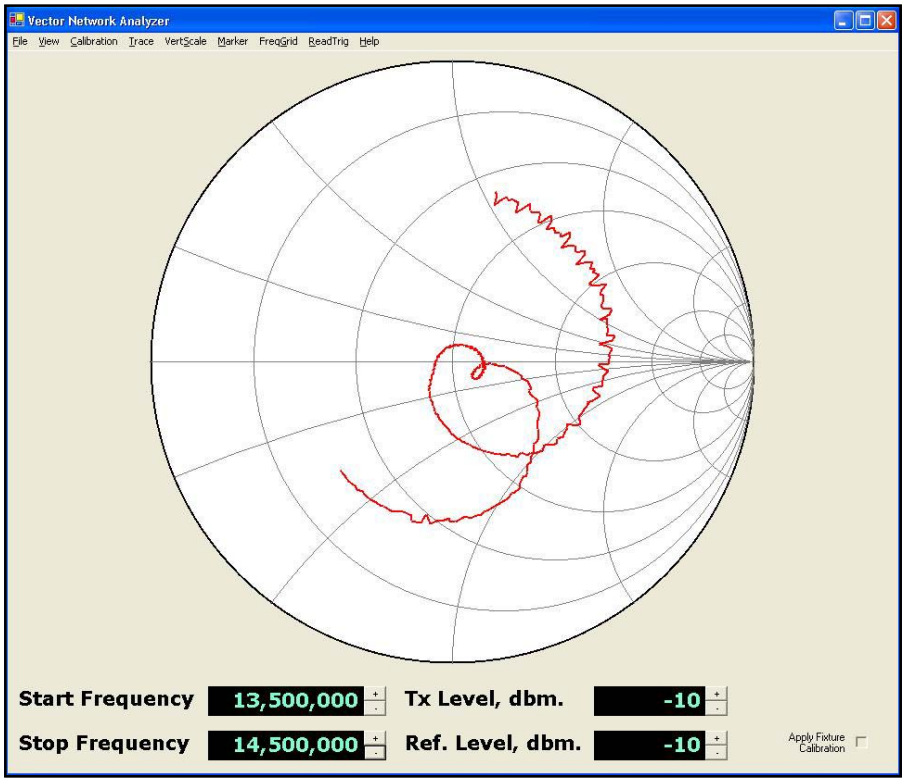


**Figure 19—Return loss of Fig 18 on a polar scale.**

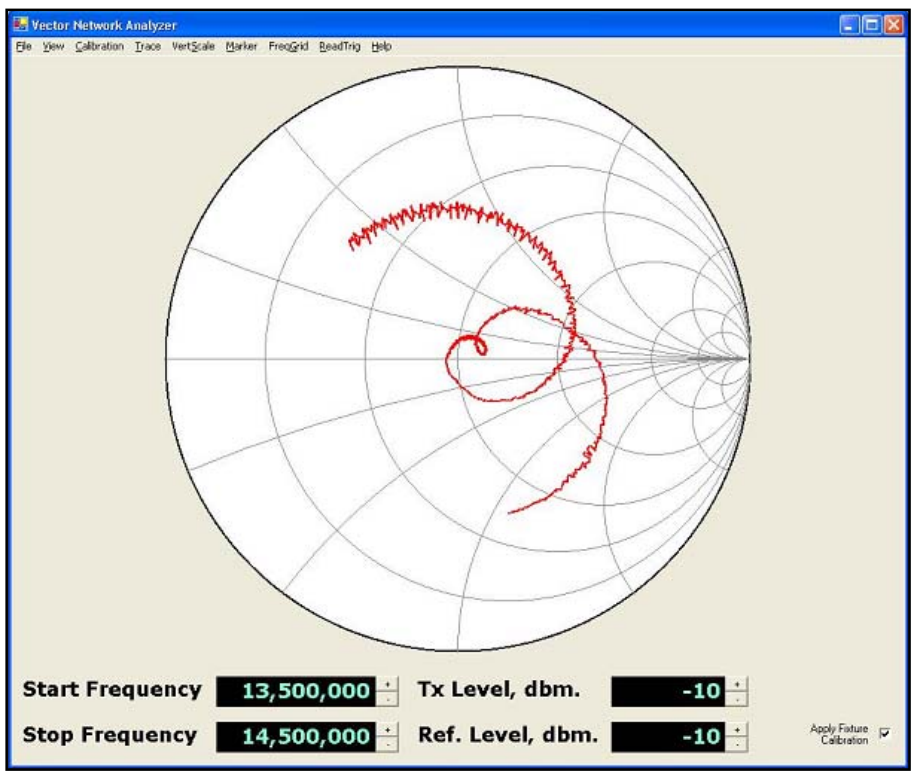


**Figure 20—Return loss of Fig 19 with calibration applied. Note the significant amount of phase rotation that is removed with instrument calibration and just 1 meter of cable correction.**

band resonances are easily seen. Fig 17 shows a closeup of the return loss from 13.5 to 14.5 MHz; Fig 18 shows the same reading after the instrument has been calibrated with a 1-m length of cable. This short cable is used to connect to the analyzer to the hard-line cable and thus represents the load that would be seen by a transmitter connected to the cable. Note that the apparent return loss is improved at some frequencies because imperfections in the instrument and directional coupler have been subtracted out of the measurement. Fig 19 shows this same closeup on a polar plot. Fig 20 is the same polar plot but with fixture calibration enabled. Note the rotation of the polar plot.

The instrument limitation for return loss measurements is about 30 dB, and this degrades by a few dB at frequencies above about 50 MHz. The apparent return loss of the antenna looks better than it really is at higher frequencies because of the increasing loss of the 300 feet of hard line with frequency.

**Notes**

[1]Universal Serial Bus Specification, Revision 1.1, Compaq, Intel, Microsoft, NEC, September 23, 1998, available at **www.usb.org/developers/docs/**.

[2]AN2135 Microprocessor Data sheet and Cypress EZUSB Development kit are available from Cypress Semiconductor, Inc., **www.cypress.com**, look under **USB Full-Speed Peripherals**.

[3]Analog Devices, Inc, AD8302 data sheet, Rev A., **www.analog.com**.

[4]Analog Devices, Inc., AD9854 DDS data sheet, Rev B, **www.analog.com**

[5]Minicircuits Inc., directional coupler PDC-20-3, datasheet available at **www.minicircuits.com/dg03-192.pdf**.

[6] J. Axelson, *USB Complete*, 2nd edition, Lakeview Research, ISBN 096508195-8, **www.lvr.com**.

[7]C. James-Tanny, *Creating HTML Help with Microsoft's HHW*, 2003, JTF Associates, Inc,**www.mvps.org/htmlhelpcenter/htmlhelp/hhtutorials.html**.

[8]D. Pozar, *Microwave Engineering,* Addison Wesley, 1998, ISBN 0-471-17096-8.

[9]Rigel Corporation Inc, Reads-51 package, PO Box 90040, Gainesville, FL 32607, **www.rigelcorp.com**.

[10]J. Templeman, A. Olsen, *Microsoft Visual C++ .NET Step by Step, Version 2003*, Microsoft Press, ISBN 0-7356-1907-7 is just the book; ISBN 0-7356-1908-5 is the Deluxe Learning Edition that includes the C++ standard edition compiler package.

[11]Jordan Russell's software, INNO installer program, **www.jrsoftware.org/isinfo.php**. □□