QEX

ARRL Experimenter's Exchange

$1.75

December 1996

Analyzing Arbitrary Transmission Lines

QEX: The ARRL
Experimenter's Exchange
American Radio Relay League
225 Main Street
Newington, CT USA 06111

QEX (ISSN: 0886-8093) is published monthly by the American Radio Relay League, 225 Main Street, Newington CT 06111-1494. Subscription rate for 12 issues to ARRL members is $12; nonmembers $24. Other rates are listed below. Periodicals postage paid at Hartford CT and at additional mailing offices.

POSTMASTER: Form 3579 requested. Send address changes to: QEX, 225 Main St, Newington CT, 06111-1494

Issue No. 179

David Sumner, K1ZZ
Publisher

Jon Bloom, KE3Z
Editor

Lori Weinberg
Assistant Editor

Zack Lau, W1VT
Contributing Editor

Production Department

Mark J. Wilson, K1RO
Publications Manager

Michelle Bloom, WB1ENT
Production Supervisor

Sue Fagan
Graphic Design Supervisor

Joseph Costa
Technical Illustrator

Joe Shea
Production Assistant

Advertising Information Contact:

Brad Thomas, KC1EX, *Advertising Manager*
American Radio Relay League
860-667-2494 direct
860-594-0200 ARRL
860-594-0259 fax

Circulation Department

Debra Jahnke, *Manager*
Kathy Capodicasa, N1GZO, *Deputy Manager*
Cathy Stepina, *QEX Circulation*

Offices

225 Main St, Newington, CT 06111-1494 USA
Telephone: 860-594-0200
Telex: 650215-5052 MCI
Fax: 860-594-0259 (24 hour direct line)
Electronic Mail: MCIMAILID: 215-5052
Internet:qex@arrl.org

Subscription rate for 12 issues:

In the US: ARRL Member $15, nonmember $27;

US, Canada and Mexico by First Class Mail: ARRL Member $28, nonmember $40;

Elsewhere by Surface Mail (4-8 week delivery): ARRL Member $20, nonmember $32;

Elsewhere by Airmail: ARRL Member $48, nonmember $60.

Members are asked to include their membership control number or a label from their *QST* wrapper when applying.

In order to insure prompt delivery, we ask that you periodically check the address information on your mailing label. If you find any inaccuracies, please contact the Circulation Department immediately. Thank you for your assistance.

About the Cover

These feed lines are easy to analyze, but what of more complex shapes? G8WRB provides an answer.

Features

December 1996 QEX Advertising Index

American Radio Relay League: Cover IV
Communications Specialists Inc: 10
HAL Communications Corp: Cov III
PC Electronics: 17
Sescom, Inc: Cov III
Tucson Amateur Packet Radio Corp: 22
Z Domain Technologies, Inc: 10

Empirically Speaking

The North American Digital Systems Directory

In the early 1980s, some odd entries began appearing in the annual editions of *The ARRL Repeater Directory*. These stations were listed with the identification: PACKET. In subsequent years, packet network station listings got larger and, eventually, occupied their own section of the *Repeater Directory*, along with a sprinkling of other "special modes." Now, for the 1997-1998 edition, the *Repeater Directory* will no longer carry listings of packet stations.

Some might view this as a negative. Why should the ARRL stop printing packet listings? Is ARRL no longer interested in digital modes? Is ARRL abandoning the packet network? Hardly. Rather, this decision simply reflects the realities of amateur packet as it is practiced and of communication methods of the 1990s.

Listing packet-network stations in a publication that is updated only once per year has always been something of a least-objectionable solution. Many packet networks are much more dynamic than voice repeaters—the primary topic covered by the *Repeater Directory*. With changing frequencies and node names, and ever-expanding networks, a year-old listing of packet stations could only be at best incomplete. Because of this, some packet networks didn't bother to provide listings for the *Repeater Directory* at all, making the listings even more incomplete.

In the last half of the 1990s, trying to make use of a printed snapshot of this kind of fluid information makes little sense. A much better way of maintaining and delivering this kind of information is now available: the World Wide Web.

Recognizing this, Tucson Amateur Packet Radio (TAPR) has taken the initiative to do something about it. The result is the North American Digital Systems Directory (NADSD). The directory (available at **http://www.tapr.org/directory/**) is maintained under the auspices of TAPR, and with the appreciative support of the ARRL, by a number of participating regional groups. As described by TAPR: "The purpose of the Digital Systems Directory is not to manage, coordinate, or regulate the usage of digital systems, but to provide the most up to date and accurate listing of digital systems that can be provided. Neither is it a formal organization, but a mechanism to allow regional groups to provide and share information regarding digital systems."

What's appealing about having this information available via the Web is, of course, the ability to update it on an as-needed basis, not just when a publication deadline nears. And intelligent presentation and searching of the data is possible too, along with more "high-tech" applications. In the latter realm, the NADSD will include a *javAPRS* capability that will allow a Java-equipped Web browser to view an interactive map of packet stations. All in all, a far cry from a static, paper directory.

This Month in *QEX*

Most any basic text on antennas and transmission lines will give you the formulas to calculate the impedance of the commonly used transmission line geometries: twin-lead, coax and the like. But what if you have to deal with something *un*common? In that case, you can either spend a great deal of time trying to derive a formula for the impedance—and good luck if the dimensions vary along the line—or fall back on a numerical solution such as that provided by Dave Kirkby, G8WRB, in "Finding the Characteristics of Arbitrary Transmission Lines."

Seems there's no end to the interest in the question: Which HF digital mode works best? There's probably no definitive answer to that question, either, but some solid empirical data gets us closer. "On-Air Measurements of HF Throughput Over NVIS Paths Using KA9Q TCP/IP," by Ken Wickwire, KB1JY; Mike Bernock, KB1PZ; and Bob Levreault, W1IMM, provide just that.

A microcontroller is a great way of implementing control circuitry that would be too complex for digital logic and analog circuits to handle. A simple means of developing code for a microcontroller, such as "A Development System for the 8051 Microcontroller," by Anthony L. Marchese, N2YM, should get you started.—*KE3Z, email:* ***jbloom@arrl.org***.

Finding the Characteristics of Arbitrary Transmission Lines

A transmission-line analysis isn't always possible using the "book" formulas. Here's a way of handling that situation.

By Dave Kirkby, G8WRB

Transmission lines, in various forms, are used in great numbers in radio frequency (RF) equipment. Fig 1 shows five common transmission lines, as well as a totally arbitrary transmission line. Although one may believe that the important properties of transmission lines, such as the characteristic impedance are easily obtainable from readily available formulas, it is shown that this is not always the case. This article describes my efforts to overcome this when designing a high-power 144- to 146-MHz valve amplifier, although the technique has much wider use in the design of microwave circuits, RF relays, directional couplers, etc.

Stokes Hall Lodge
Burnham Road
Althorne, Essex CM3 6DT, England
e-mail **davek@medphys.ucl.ac.uk**

Transmission lines such as coaxial cable, twin-wire feeder, microstrip and stripline have distributed inductance along the conductors and distributed capacitance between the conductors, so a section of line can be represented by Fig 2.

The values of the distributed capacitance and inductance determine such properties as the characteristic impedance Z_0, which is calculated using Eq 1.

$$Z_0 = \sqrt{\frac{L}{C}}\ \Omega \qquad \text{Eq 1}$$

For example, a transmission line having a capacitance of 100 pF/m and an inductance of 250 nH/m would have a characteristic impedance of 50 Ω. They also determine the velocity, v, at which a radio wave propagates in the transmission line, according to Eq 2:

$$v = \frac{1}{\sqrt{LC}}\ \text{m/s} \qquad \text{Eq 2}$$

Therefore, a transmission line with C=100 pF/m and L=250 nH/m would propagate a radio wave at 2×10^8 m/s. Since radio waves propagate at 3×10^8 m/s in free space, the velocity factor of the transmission line would be $2 \times 10^8 / 3 \times 10^8 = 0.66$. These are common values for coaxial cables.

Finding the Properties of Common Types of Transmission Lines

Given knowledge of the physical dimensions of coaxial cable and the permittivity of the dielectric, its properties can easily be found with a few simple formulas. For example, the characteristic impedance Z_0 is given by Eq 3:

$$Z_0 = \frac{60}{\sqrt{\varepsilon_0 \varepsilon_r}} \log_e\left(\frac{D}{d}\right) \qquad \text{Eq 3}$$

where ε_0 is the permittivity of free space (ε_0=8.854x10^{-12} F/m), ε_r is the relative permittivity of the dielectric (ε_r=1.0 for air, 2.1 for PTFE, 2.3 for polyethylene etc), D is the inner diameter of the outer conductor and d is the outer diameter of the inner conductor. This formula is exact and easy to use with a scientific calculator.

Microstrip line is more complex to analyze, but an approximate formula, good enough for engineering purposes, can be found in the amateur press.[1, 2] An exact formula does exist for microstrip, but it is too complex for general use. An especially convenient calculation method is a small, free, Microsoft Windows program called *Txline* written by AWR which calculates the properties of microstrip and stripline, as well as three less widely used transmission line types. (Applied Wave Research Inc, USA, email **pekarek@appwave.com**. Web page: **http://www.appwave.com/**.)

Finding the Characteristics of an Arbitrary Transmission Line

Now consider a transmission line such as that on the right side of Fig 1. If you wish to find the characteristics of a transmission line such as this, there will definitely be no formula in a book! While such a transmission line is not likely to be encountered in practice, on occasion one does require characteristics of transmission lines that can not easily (if at all) be found in the literature. This happened to me when trying to determine the characteristic impedance of an air-spaced microstrip to be used as an anode resonator in a 144- to 146-MHz grounded-grid valve amplifier using an Eimac 3CX5000A7 triode. (The amplifier has not been completed yet, nor will it be for at least a year. The details may be published at a later date, assuming it works well.) Using transmission lines as resonators is a common practice in VHF valve amplifiers,[3,4] although contrary to popular belief, L-C tuned circuits can be successfully used.[5] When using transmission lines, theoretically any impedance line can be made to resonate with the valve's output capacitance if cut to the correct length,[6] but there is an optimum value for Z_0 for maximum amplifier bandwidth.[7] My aim was to design a transmission line with the optimal Z_0, which for my particular choice of valve (3CX5000A7) and resonator configuration (half-wave line with the tube at the center), was a line impedance of at least 81 Ω. Unfortunately, the presence of the metal amplifier case around the microstrip, essential for safety, could not be ignored. Making the case sufficiently large, so it was well away from the microstrip and could therefore be ignored, was not practical—I did not want the amplifier to fill half the shack! Hence my amplifier's enclosure was to become an integral part of an unsymmetrical shielded stripline transmission line, as shown in Fig 3, with a metal stripline of width w and thickness t, placed centrally on the horizontal axis at a height h above the bottom of a metal case of internal width W and internal height H. (The distinction between microstrip and shielded stripline is rather vague here. Any microstrip line enclosed in a metal box really becomes a shielded stripline. If, however, the enclosure is large compared with the microstrip, which is usually the case with low-power circuits, then for all practical purposes, the transmission line remains microstrip. Although my amplifier was intended to have an air-spaced microstrip resonator, the relative size of the enclosure to the anode line made the enclosure part of a shielded stripline resonator.) The term *unsymmetrical* is used to indicate that the center conductor is not in the center of the lower and upper conductors. To the best of my knowledge, following a computer search of the Science Citation Index, listing virtually all professional science and engineering publications since 1981, there is no analytical expression for the impedance of the transmission line in Fig 3. Robrish has, however, found one for an unsymmetrical stripline that is not shielded at the sides (equivalent to W being infinite).[8]

With a real need to solve a problem, but with no formula available to me, a *finite difference* computer program was written to solve numerically the problem of the shielded stripline in Fig 3. This was based on a method in a book by Dworsky where the interested reader can find full proofs of all the mathematics, which is quoted here without proof.[9] The program calculates the characteristic impedance (in Ω), the capacitance per meter (in pF/m) and the inductance per meter (in nH/m) of the transmission line. In its most basic form, the complete code is only 66 lines long and is reproduced in full in Appendix 1. The program expects the five parameters W, H, w, h and t to follow the program name on the command line. Once you understand how the program functions, modifying it to handle a weird transmission line like that on the right of Fig 1 is not difficult.

The program can also be used in

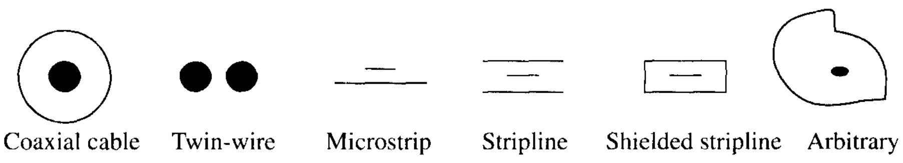


Fig 1—Diagram showing a number of common transmission lines. All, except the twin-wire, are just special cases of the last one and can be analysed by making minor modifications to the program described here.

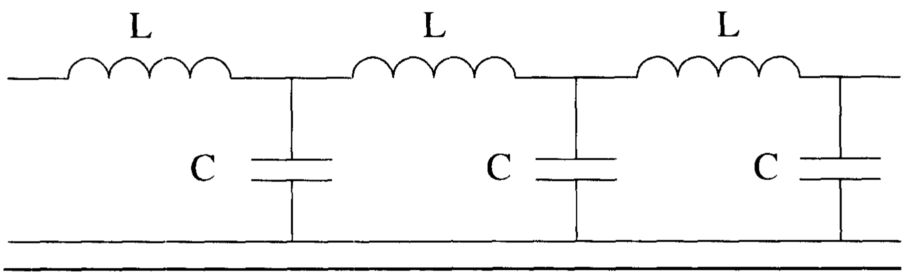


Fig 2—A transmission line has a distributed shunt capacitance and distributed series inductance. Loss terms of series resistance and shunt resistance are ignored.

[1]Notes appear on page 8.

designing directional couplers, RF relays or any other item that contains a transmission line.[6] By being able to calculate the impedance for any position of a center conductor relative to the outer, you can usually design a transmission line for exactly 50-Ω using whatever sizes of inner and outer conductors are readily available. There is no need to machine the inner to have the right size for a 50-Ω line—you simply place the inner conductor the correct amount off-center.

Theory

The complete theory of how the program works is quite complex. Fortunately, neither using the program nor modifying it to handle virtually any transmission line requires a detailed understanding. Here's a simplified description that assumes only one dielectric, although we later extend this to more than one. The transmission line is assumed to have constant dimensions along its length. We assume the outer conductor is earthed, with 0 V on it. We set the inner conductor to a dc voltage of V_0 V. V_0 will be set equal to 1 V here, although it does not matter what voltage is chosen as long as it is nonzero, since its value gets cancelled. Theoretically, if we knew the voltage at every single location over the cross section of the line, we could determine the capacitance per meter of the transmission line. However, there are an infinite number of different locations, and as no computer has an infinite amount of memory, that approach is impossible.

If, however, we cover the transmission line's cross section with an imaginary square grid, there is now a finite number of nodes (corners of the squares). This is shown in Fig 4. We can now store the voltage at every node in a computer. If the grid size is sufficiently small, the voltage will not change much from one node to the next, and the error in not knowing the voltage everywhere will be small. The voltages will be stored in a two-dimensional matrix $V_{i,j}$, where i ranges from 0 to $I_{\max}$ and j ranges from 0 to $J_{\max}$. The transmission line's boundary must lie on the grid, so odd shaped conductors will have to be approximated. This is shown in Fig 5 (for the strange transmission line to the right of Fig 1).

We already know the voltage at some points on the grid since the outer is at 0 V dc and the inner at V_0, which is 1-V dc, but the other voltages can be found easily if we accept Laplace's equation.[10] We need not bother ourselves with the details of Laplace's equation, but must accept that to satisfy a discrete version of it, and so calculate the voltage at every unknown node, we just apply Eq 4 over every node, except on the transmission line conductors, where the voltages are fixed at either 0 or 1 V.

$$V_{i,j}=\frac{V_{i+1,j}+V_{i-1,j}+V_{i,j+1}+V_{i,j-1}}{4} \qquad \text{Eq 4}$$

We do this once at every node, then repeat the process again and again. We keep doing this since each time we get a new voltage for the point *i,j*, it is closer to the true value, but will probably never exactly get there. Hence if we have 1,000 nodes, we might typically apply Eq 4 100,000 times—100 times per node. Eq 4 ensures that the voltage at a node is the average of the voltages at the nodes around it, which ensures that Laplace's equation is satisfied.

If you inspect the program you will note that the two lines that update voltage are equivalent to:

$$V_{i,j}(\text{new})=r\left[\frac{V_{i+1,j}+V_{i-1,j}+V_{i,j+1}+V_{i,j-1}}{4}\right]+(1-r)V_{i,j}(\text{old}) \qquad \text{Eq 5}$$

where r=1.5. The reason for using Eq 5 instead of Eq 4 is that the latter speeds the program's convergence to the correct value of voltage at every point—it does not affect the ultimate result.

Having found the voltage $V_{i,j}$ at every node, we can find the capacitance per meter of the line at dc. The theoretical basis of this is not trivial, so it is best to accept that the capacitance per meter is related to the voltage summed over the rectangle enclosing the transmissions line's cross section.

$$C_0=\frac{\varepsilon_0}{2Vo^2}\sum_{i=0}^{I_{\max}-1}\sum_{j=0}^{J_{\max}-1}(V_{i,j}-V_{i+1,j+1})^2+(V_{i+1,j}-V_{i,j+1})^2 \text{ F/m} \qquad \text{Eq 6}$$

This looks complex, but in fact takes only 4 lines of code to complete.

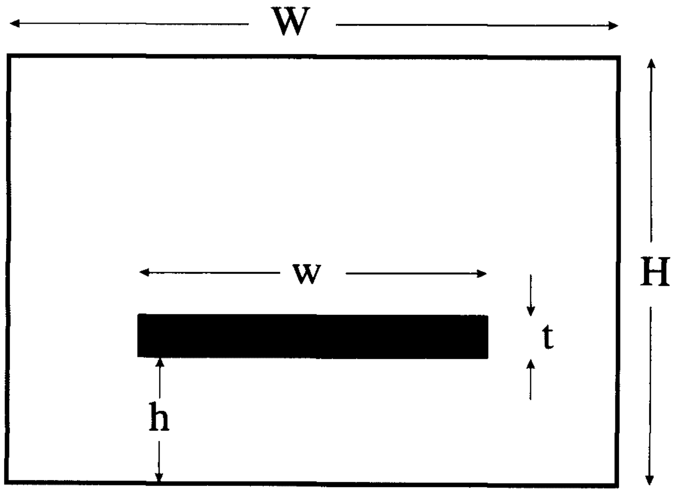


Fig 3—The amplifier whose stripline characteristics were wanted. Unfortunately, the presence of the amplifier case can not be ignored when determining the stripline impedance.

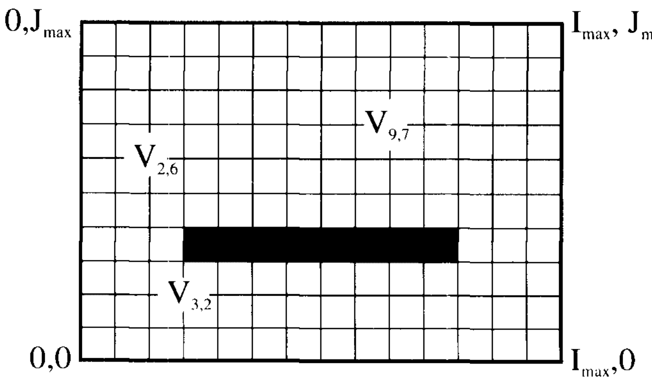


Fig 4—The transmission line cross section is covered with a grid $I_{\max}$ by $J_{\max}$ (in this case 12×10), and the voltage V_{ij} is evaluated at every node.

Having found the capacitance per meter for an air-spaced line C_0, the inductance per meter L is given by Eq 7:

$$L = \frac{\mu_0 \varepsilon_0}{C_0} \text{ H/m} \qquad \text{Eq 7}$$

where μ_0 is the permeability of free space = 12.57×10^{-7} H/m. If the line is not air-spaced, but is filled with a single dielectric with relative permittivity ε_r (where $\varepsilon_r > 1$), the true capacitance per meter of the line is:

$$C = \varepsilon_r C_0 \text{ F/m} \qquad \text{Eq 8}$$

Having found the distributed inductance and capacitance, the characteristic impedance Z_0 is simply found from Eq 1. If the line is not air-spaced, the velocity of propagation can be found from Eq 2.

A Simple Computer Program

Appendix 1 lists a *finite difference* computer program, written in C, which I called *ATLC—Arbitrary Transmission Line Calculator*. The program as listed solves the problem of Fig 3 for arbitrary values of W, H, w, h and t. It could, however, be modified to solve for any transmission line, such as that in Fig 5, by altering a few lines that set different parts of the transmission line to 0 or 1 V. The geometry of the transmission line would need to be defined in the source code. The program compiles with no problem on Microsoft's *Quick C* version 2.0 for a PC, using a compact memory model, *gcc* version 2.7.2 for a Sun Ultra 1 workstation and *gcc* 2.7.2 for Linux. Since the program uses no special functions, it should compile and run without any hassle on any modern computer. The program's source code (ATLC.C), along with a precompiled executable (ATLC.EXE) is available by anonymous FTP from **ftp.arrl.org/pub/qex**.

Where possible, the program was written in such a way that programmers unfamiliar with the C programming language should be able to convert it to any language they like—FORTRAN, BASIC, Pascal, etc. There are just a few bits that might puzzle such a programmer. First, a C array of dimensions declared as *float arraynname[x][y]*, has locations [0..x–1][0..y–1], which is different from say FORTRAN, where they wound be (1..x)(1..y). The function *atoi()*, converts a string of characters to an integer, *atof()* a string of characters to a floating point number and the function *fabs()* finds the absolute value of a floating point number. The *main()* function essentially reads the values of W, H, w, h and t from the command line, does a few quick checks, then passes them to the calculation routine, which is something I often find convenient for numerical programs. However, you could read W, H, w, h and t in from the keyboard or disk if you prefer.

The program expects the input dimensions to be in units of the width (or height) of a grid square—not millimeters or inches. You must decide the size of grid to use. Generally, if the largest dimension in your problem is y, then you want to allocate 40 or more grid squares to y. So for example, if a stripline is enclosed in a box 400-mm wide × 200-mm high, use 10 mm = 1 grid square, and hence the problem uses 40 grid squares (W=40) by 20 grid squares (H=20). Then recheck the results at double the resolution (5 mm = 1 grid square in this case) and ensure the results are not significantly different. The examples and results later demonstrate this. If you wish to use in excess of a 127 by 127 grid, the array in the program in Appendix 1 will need to be enlarged.

Mixed Dielectric

If the transmission line contains two or more different dielectrics with different relative permittivities, such as a microstrip on a printed circuit board as shown in Fig 6, then the problem becomes more complex. First, any calculation of C performed with a program like this is the dc value. While this does not vary with frequency when there is only one dielectric, it does when there are more than one. Hence even if we find C, L and Z_0 for the dc case, these will be frequency dependant. However, if the frequency is not too high (and this is difficult to quantify), the approximations will be acceptable.

The procedure for finding the characteristic impedance of a multiple-dielectric microstrip line and a FORTRAN IV program to perform this are contained in Dworsky's book. No program will be given, but the basic method is outlined here. The method is:

1) Calculate the voltage distribution across the transmission line, assuming it's air-spaced, as before.

2) Calculate the capacitance per meter C_0, assuming the transmission line has just an air-dielectric, as described before.

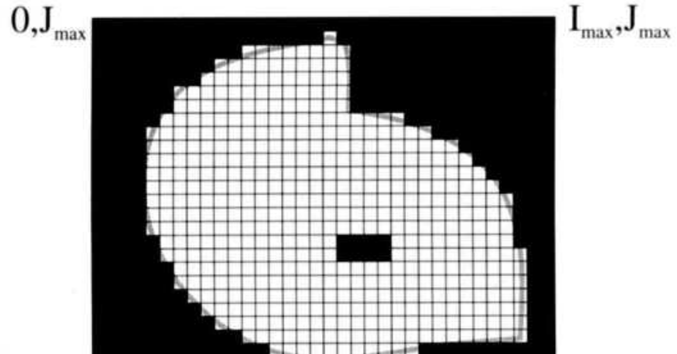


Fig 5—Diagram showing how the odd shaped transmission line on the right of Fig 1 would be approximated on the grid.

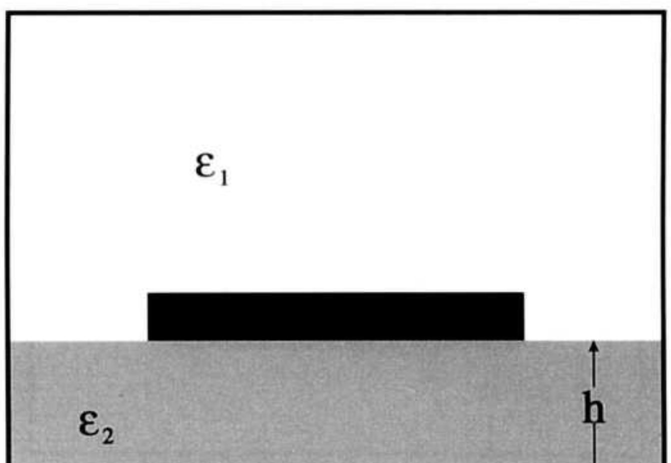


Fig 6—Shielded microstrip with a mixed dielectric. Microstrip line etched on a PC board and enclosed in a case would resemble this.

3) Find the inductance per meter, by using C_0 in Eq 7.

4) Recalculate the voltage distribution across the transmission line using Eq 4, *except* when you are at the interface between the two dielectrics. When at the interface, in order to satisfy Laplace's equation, we must use Eq 9 instead:

$$V_{i,j} = \frac{\varepsilon_1 V_{i+1,j} + \varepsilon_2 V_{i-1,j} + \frac{(\varepsilon_1 + \varepsilon_2)}{2}(V_{i,j+1} + V_{i,j-1})}{2(\varepsilon_1 + \varepsilon_2)} \quad \text{(when j = h)}$$

Eq 9

where ε_1 is the relative permittivity of the dielectric above the interface—this is probably air, so ε_1=1.0. ε_2 is the relative permittivity of the dielectric below the interface—usually a PC board material.

5) Recalculate the true line capacitance per meter, C, using the fact that C is:

$$C = \frac{\varepsilon_0}{2Vo^2} \sum_{i=0}^{I_{max}-1} \sum_{j=0}^{J_{max}-1} \varepsilon_j \left[(V_{i,j} - V_{i+1,j+1})^2 + (V_{i+1,j} - V_{i,j+1})^2 \right] \text{ F/m}$$

Eq 10

where ε_j is the permittivity of the dielectric immediately below row j—that is ε_2 where $j < h$ and ε_1 where $j > h$.

6) Calculate Z_0, using L calculated in step 3 and C calculated in step 5.

Since the voltage and capacitance distribution are calculated twice, the program will take twice as long as a single dielectric case.

Effects of Boundaries

This finite difference method fully encloses the inner conductor in a shield, which need not be rectangular. However, it must exist. If the physical problem being modeled is not like this, there is a small problem. For example, to model an air-spaced microstrip (see Fig 1), with no surrounding metal, we have a problem. One way to solve this is to put the boundary at a large distance. Try it at a distance of x, then repeat the calculation at a distance of $2x$. If the results are significantly different, x was not large enough.

Testing the Program

Any numerical program requires very thorough testing as, unlike many programs such as games and word-processors, data can look believable even though it is total rubbish. Some people may be skeptical since the theoretical basis of the program has not been rigorously justified here—although it is in Dworsky's book. To test the program, a number of transmission lines were analysed under conditions where the results could be checked by other means. For example:

Test 1) The characteristic impedance Z_0 of a thin (t=0), air-spaced (ε_r=1) microstrip of width 5 mm and height 5 mm was calculated using an empirical formula (Note 1) and found to be 126.15 Ω. The finite difference program was then set to find the impedance of a shielded stripline, using 1 grid point = 0.5 mm. Therefore the line was 5/0.5=10 grid points wide (w=10) and 5/0.5=10 grid points high (h=10). This was enclosed in a shield 63-mm wide (therefore W= 63/0.5=126) × 63-mm high (therefore H=63/0.5=126). The shield being so large compared to the stripline, its effect should be minimal and so the results should be similar. The program terminated after performing 1140 iterations in 954 seconds on a 25-MHz 486 PC and returned a Z_0 of 123.2 Ω, which is acceptably close, with a difference of −2.3%. Of course, the shield, even at this distance, may affect Z_0 by a few percent. To check this for certain, the program was run with a 0.25 mm = 1-grid-point resolution (therefore w=20, h=20, t=0) and a screen 300-mm wide (W=300/0.25=1200) by 300-mm high (H=300/0.25=1200) on a fast Sun Ultra 1 computer. Z_0 was then 124.8 Ω, a difference of just −1.1% from the expected value. This could be due to errors in the empirical formula, which is not 100% accurate.

Test 2) The characteristic impedance Z_0 of an air-spaced shielded stripline with a width of 19 grid points (w=19), 1 grid point thick (t=1) placed centrally along both the x and y axes, in a shield of 99 (W=99) × 49 (H=49) grid points was calculated. Since it was placed centrally, h=24. According to Dworsky, this has a theoretical impedance of 109 Ω. The program took 104 seconds on the 25-MHz 486 to perform 400 iterations and return Z_0 =108.7 Ω, an error of only −0.28%.

Results

For the amplifier being designed, the anode line needed to be at least 90 mm above the ground for practical reasons—to clear the valve chimney. According to the *Txline* program mentioned earlier, choosing a stripline 160-mm wide and 1-mm thick, and assuming no surrounding metal case, the line impedance would be 94.5 Ω. This would have been acceptable as it exceeded the 81 Ω minimum needed. However, this ignored the presence of the metal case, which was 200-mm wide by 290-mm high.

For a finite difference simulation of the problem, a scale of 5 mm = 1 grid point was used, as this gave convenient numbers while also executing quickly. Hence the case width (W) was set at 290/5=58 grid points, the case height (H) to 200/5=40 grid points, the stripline width (w) to 160/5=32 and the stripline height (h) to 90/5=18. The stripline thickness should have been 1/5 = 0.2 grid points, but this is clearly impossible, so this was rounded to the nearest integer: 0. Below is the output of the computer simulation. Note that Z_0 slowly converges upwards, as the program makes better and better estimates of the true voltage distribution across the transmission lines cross section.

```
C:\2D_ATLC>atlc 58 40 32 18 0
 10 c=77.91pF/m l=142.85nH/m Zo=42.820003 Ohms
 20 c=60.06pF/m l=185.30nH/m Zo=55.544252 Ohms
 30 c=53.14pF/m l=209.45nH/m Zo=62.782017 Ohms
 40 c=50.13pF/m l=222.03nH/m Zo=66.553549 Ohms
 50 c=48.81pF/m l=228.01nH/m Zo=68.347540 Ohms
 60 c=48.23pF/m l=230.76nH/m Zo=69.170590 Ohms
 70 c=47.97pF/m l=232.01nH/m Zo=69.544214 Ohms
 80 c=47.85pF/m l=232.57nH/m Zo=69.713328 Ohms
 90 c=47.80pF/m l=232.82nH/m Zo=69.789547 Ohms
100 c=47.78pF/m l=232.94nH/m Zo=69.823501 Ohms
110 c=47.77pF/m l=232.99nH/m Zo=69.838244 Ohms
120 c=47.76pF/m l=233.01nH/m Zo=69.844314 Ohms
130 c=47.76pF/m l=233.01nH/m Zo=69.846544 Ohms
140 c=47.76pF/m l=233.02nH/m Zo=69.847136 Ohms
```

The 140 iterations took 17 seconds on the 25-MHz 486 PC. To check that the data was reasonable, the program was rerun using double the resolution (2.5 mm between grid points). This took 272 seconds, but the result, Z_0 = 70.1 Ω, differed by less than 0.4% from the previous, much faster run. Hence there was no appreciable difference except a 16-fold increase in execution time. A much more accurate calculation, using a 280 × 200 grid, where the thickness of the stripline could be taken into account properly (t did not have to be set to zero), showed the correct result to be nearer 68.6 Ω but took 9111 s (just over 2.5

hours). Hence a calculation performed in 17 seconds has an error of probably less than 2%, which should be of adequate accuracy.

The results show that the amplifier case reduces the impedance of the stripline from 94.5 Ω to 68.6 Ω, below the minimum acceptable value of 81 Ω. Since the line impedance is altered by the amplifier case, the resonant length of a line is also altered. The program allows many "what if?" calculations—what if the line was narrower, the case bigger, the line thicker, etc.

Computational Problems

The program is computationally intensive and will therefore be slow if the highest accuracy is desired, although such accuracy is usually not necessary. The program's execution time increases with the square of the array area used for calculation, so doubling the resolution means a 16 times increase in execution time. A computer with some built-in floating point hardware is essential for this—a fast 386 with a 387 math chip, or better still, a 486 or Pentium machine. This is not the application to demonstrate that your Commodore 64 or Sinclair Spectrum was worth hanging on to! The amount of memory used by the program is determined by the array size used for storing the voltages. A 127x127 array of single-precision floating point numbers (4 bytes each) uses just under 64 kbytes, which is the limit on some PC compilers for a single array. Using a decent compiler will allow much bigger arrays, but the execution time will slow. However, 127x127 seems adequate for most applications. The way to be sure you are using sufficiently fine resolution is to try doubling it to see if the results differ significantly. If you have the memory, it would be wise to use double-precision numbers for the array to reduce rounding errors.

Discussion

Itoh has compared a number of different numerical methods for analysing passive microwave structures and finds the finite difference method to be the slowest and use the most memory![11] However, it does have two distinct advantages over other methods. First, it is very general, and second, the problem requires no pre-processing. These advantages far outweigh its disadvantages for amateur use.

If the program was written as a Windows application, it would be possible to define the transmission line's outline with a mouse, which would be easier than the method used here for complex shapes. Different colors could be used for different dielectrics.

A three-dimensional (3-D) version of this program could be written, which would make analysing transmission lines with discontinuities along their length possible. However, this is probably not a viable option for current home computers—but it will be in a few years. The memory requirements for a 3-D model would not be excessive even by current standards—a 100×100×100 array of double-precision numbers needing only about 8 Mbytes of RAM—but would probably take about a week to execute on a fast 200-MHz Pentium processor.

Notes:

[1]Weiner, K. DJ9HO, "The UHF Compendium," Parts 1 and 2, Published by Verlag Rudolf Schmidt, Germany, pp 35-40. Note: the formula in this reference has a typographical error—it shows Z_0 being proportional to $1/\varepsilon$, when it should be proportional to $1/\sqrt{\varepsilon}$.

[2]Fisk, J. R., W1HR, "Microstrip Transmission Line," *Ham Radio*, pp 28-37, January 1978.

[3]Meade, E. L., K1AGB, "A 2-kW PEP Amplifier for 144 MHz," Part 1, *QST*, pp 34-38, Dec 1973. Part 2, *QST*, pp 26-33, Jan 1974.

[4]McMullen, T. F. and Tilton E. P., "New Ideas for the 2-Meter Kilowatt," *QST*, pp 24-30, Feb 1971.

[5]Mandelkern, M., "A Luxury Linear," *QEX*, pp 3-12, May 1996.

[6]Jessop, G. R., G6JP, *VHF UHF Manual*, 4th Edition, Radio Society of Great Britain, 1983.

[7]Sutherland, R. I., W6UOV, "Design Data for a Two-Kilowatt VHF Linear," *Ham Radio*, pp 6-13, March 1969.

[8]Robrish, P., "An Analytic Algorithm for Unbalanced Stripline Impedance," *IEEE transactions on microwave theory and techniques*, pp 1011-1016, 1990.

[9]Dworsky, L. N., *Modern Transmission Line Theory and Applications*, Chapter 9, John-Wiley, 1979.

[10]Ramo, S., Whinnery, J. R. and Van Duzer, T., *Fields and Waves in Communication Electronics*, 2nd edition, Wiley, 1984.

[11]Itoh, T., *Numerical Techniques for Microwave and Millimetre-Wave Passive Structures*, Wiley, 1989.

Appendix 1—Source Code for ATLC.C

Note: Some compilers may need the line '#include <stdlib.h>'—others may not.

```
#include <stdio.h> /* ATLC - Arbitrary Transmission Line Calculator. ver 1.0 */

#include <math.h>        /* By D. Kirkby G8WRB.  Compiles okay with          */

#include <stdlib.h>      /* Microsoft  Quick C, version 2.0 and GNU C.       */

#define Imax 126         /* Voltage array size will be 0..Imax               */

#define Jmax 126         /* ie v[0..Imax][[0..Jmax]                          */

float v[Imax+1][Jmax+1]; /* Declare an array to hold the voltages            */

void arbitrary_transmission_line(int W, int H, int w, int h, int t);


void main(int argc, char **argv) /* Read parameters from command line here   */

{

    int W, H, w, h, t; /* integers for number of grid squares to use. */

    if((argc!=6) ) /* Check the number of command line arguments are correct */

    {

            printf("Usage: %s W(shield) H(shield) width height thickness\n", argv[0]);
            exit(1); /* Exit - program called with wrong number of arguments */
    }
    W=atoi(argv[1]); /* Read shield width (in grid points) from command line.    */
    H=atoi(argv[2]); /* Read shield height (in grid points) from command line.   */
    w=atoi(argv[3]); /* Read strip width (in grid points) from command line.     */
    h=atoi(argv[4]); /* Read strip height (in grid points) from command line.    */
    t=atoi(argv[5]); /* Read strip thickness (in grid points) from command line. */
    if((W>Imax)||(H>Jmax)||(h+t>H-1)||(w>W-2)||(h<0)||(t<0)||(W<0)||(H<0)) /* Basic checks */
    {
            printf("Sorry - one of the arguments is silly - too big, too small ?\n");
            exit(2);
    }
    arbitrary_transmission_line(W,H,w,h,t); /* Calculate L, C and Zo     */
}

void arbitrary_transmission_line(int W, int H, int w, int h, int t)
{
    double Eo=8.854e-12, Er=1.0, mu=12.57e-7, c, l, Zo, vnew,r=1.5, c_old;
    int i, j, k=0, done=0;
    for(i=0;i<=W;i=i+1)     /* Zero the voltage array. Its essential that the  */
            for(j=0;j<=H;j=j+1) /* outer is at 0V, but desirable for everywhere to */
                    v[i][j]=0.0;      /* start at 0 V. */
    for(i=(W-w)/2;i<=(W-w)/2+w;i=i+1) /* Put stripline in centre of x axis, */
            for(j=h;j<=h+t;j=j+1)           /* and between h and h+t on the y axis,*/
                    v[i][j]=1.0;                /* then set stripline there to 1 V */
    do{ /* Set up a relaxation loop, to find the voltage at every point */
            k=k+1; /* increment the counter used to count the iterations */
      for(i=1;i<=W-1;i=i+1) /* Data at i=0 must stay fixed at v=0 */
                    for(j=1;j<=H-1;j=j+1) /* as this is a 'boundary condition' */
                            if(v[i][j]!=1.0) /*ie. don't do this where the stripline is */
                            {
                      vnew=r*(v[i+1][j]+v[i-1][j]+v[i][j+1]+v[i][j-1])/4+(1-r)*v[i][j];
                                    v[i][j]=vnew; /* New voltage is calculated */
                            }
            if(k%10==0) /* Now we have v distribution we find C every 10 iterations */
            {
                    c_old=c; c=0.0;
                    for(i=0;i<=W-1;i=i+1) /* Sum v over cross-section to get C, which  */
                         for(j=0;j<=H-1;j=j+1) /* is easy for a rectangular cross section */
                             c=c+pow(v[i][j]-v[i+1][j+1],2.0)+pow(v[i+1][j]-v[i][j+1],2.0);
                    c=c*Eo/2.0; /* Find capacitance - only correct if air-spaced */
                    l=mu*Eo/c;  /* Calculate the line inductance - always correct */
                    c=c*Er;            /* Correct the capacitance if line has a dielectric */
```

```
                Zo=sqrt(1/c);     /* Calculate the characteristic impedance */
                printf("%5d c=%.2lfpF/m l=%.2lfnH/m Zo=%lf Ohms\n",k,c*1e12,l*1e9,Zo);
                if(fabs(c_old-c)/c < 0.00001) /* Until they differ by < 0.001 % */
                        done=1;  /* Little change in calculated value of C - so we finish*/
                else
                        done=0; /* Large change in calculated value of C - lets continue */
          }
    }while(done==0); /* Repeat for until the capacitance has converged */
} /* End line of program - line 66 */
```

□□

DSP

Without Tears®

Learn DSP and put your knowledge to work
IMMEDIATELY!

To receive an informative brochure on this popular seminar, call Z Domain Technologies, Inc. at 1-800-967-5034 or 770-587-4812. Hours: 9-5 EST. Or E-mail: dsp@zdt.com

Also, a 3-day ADVANCED class in DSP is available on alimited basis — call for more details.

By taking this 3-day course, you will really learn DSP. *Guaranteed!*

Coming to a city near you:
Richardson, TX • Long Beach, CA
Rosemont, IL • San Jose, CA

Now Available on CD-ROM!

http://www.zdt.com/~dsp

Surface Mount Chip Component Prototyping Kits— Only 49^{95}

CC-1 Capacitor Kit contains 365 pieces, 5 ea. of every 10% value from 1pf to 33µf. CR-1 Resistor Kit contains 1540 pieces, 10 ea. of every 5% value from 10Ω to 10 megΩ. Sizes are 0805 and 1206. Each kit is ONLY $49.95 and available for Immediate One Day Delivery!

Order by toll-free phone, FAX, or mail. We accept VISA, MC, COD, or Pre-paid orders. Company PO's accepted with approved credit. Call for free detailed brochure.

COMMUNICATIONS SPECIALISTS, INC.
426 West Taft Ave. • Orange, CA 92665-4296
Local (714) 998-3021 • FAX (714) 974-3420
Entire USA 1-800-854-0547

On-Air Measurements of HF Throughput Over NVIS Paths Using KA9Q TCP/IP

More empirical data measuring HF digital systems in "the real world."

By Ken Wickwire, KB1JY; Mike Bernock, KB1PZ; and Bob Levreault, W1IMM

This article is one of a series treating on-air measurement of throughput of various HF data-transmission protocols available to amateurs (see the references to our other reports at the end). In it we describe an extensive set of measurements of throughput for text files sent over near-vertical-incidence-skywave (NVIS) paths using the file transfer protocol (FTP) implemented in versions of the KA9Q TCP/IP package. The measured throughput data in our experiments were analyzed using software specially written to compute throughput statistics for our FTP file transfers. Since NVIS paths are among the worst available for HF communication our throughput assessment is conservative.

FTP embodies, through its use of the transmission control protocol (TCP), relatively sophisticated flow control and an automatic repeat request (ARQ) scheme. The TCP/FTP protocol pair can send both text and binary data. To some extent, the FTP/TCP combination offsets the well known shortcomings of the standard TNC modems many hams currently use for HF packet file transfer: although usually slower than AX.25 packet, FTP/TCP is more reliable and can transfer a wider variety of file types.

Those who have tried HF file transfers over difficult (eg, short) links using the KA9Q package will probably guess that throughputs close to those achieved with protocols like GTOR and CLOVER are rarely obtained using the package's underlying link-layer protocol (AX.25) running on TNCs.[1] Although we too expected this to be the case, we decided that a TCP/IP measurement campaign was still worth carrying out to increase our understanding of four areas:

1. How the FTP protocol works in bad HF channel conditions;
2. The TCP and AX.25 parameter settings that lead to relatively high throughput;
3. The parameter-adapting strategies that lead to relatively high throughput; and

[1]Notes appear on page 17.

232 North Road #17
Bedford, MA 01730

22 Red Field Circle
Derry, NH 03038

3 Ferndale Avenue
Norfolk, MA 02056

4. How FTP's throughput compares with that of protocols like AMTOR and GTOR.

Increased knowledge in these areas should allow hams to make further progress in the development of robust, adaptive HF data transmission systems. Adaptation to changing channel conditions, combined with ARQ and modems that employ forward error correction, interleavers and possibly channel equalizers, is the key to effective HF data communication.

The NVIS paths we have studied, which are from 25 to 200 miles long, frequently display strong multipath, high local and propagated noise, D-layer absorption at midday and occasionally strong interference from other stations operating in both voice and digital modes. (Horizontal antenna polarization at all stations allowed us to be fairly certain we were using NVIS rather than surfacewave propagation, and this was confirmed by the fading we observed most of the time.)

All the stations in our tests used Kantronics KAMs running in the KISS mode for TCP/IP operations. In the KISS mode, a TNC acts as a simple packet assembler/deassembler and the majority of whatever protocol operations send data frames through the TNC run on an attached computer. Most of our measurements were made using transmitter output of around 100 W, and all stations used sloping longwires or dipoles. These setups can be viewed as reflecting average station capabilities. All files sent are ASCII text files with lengths between about 500 and 11,000 characters. To keep testing time down, we usually restricted the file size to around 1000 characters.

The majority of measurements were at 3.6155-MHz LSB, with some at 1.815-MHz LSB. They were made over all daylight hours and a few were made in the evening while the maximum usable frequency (MUF) was still above the operating frequency. The tests covered the eleven-month period from January to December 1996. The average sunspot number during this period was about ten, so MUFs were near their cyclical minima.

The rest of the paper describes the relevant parts of the KA9Q TCP/IP package and file-transfer operation, the statistical analysis software, the paths between stations and layout of antennas, a statistical summary of the data, a discussion of the statistical results and concluding remarks.

The KA9Q TCP/IP Software Package

The software used by most hams to communicate with TCP/IP over AX.25 radio links is based on a modification for half-duplex operation of the UNIX TCP/IP/X.25 protocol suite. The UNIX suite was originally developed for the Department of Defense and now forms the basis of Internet communications. The ham version, written about ten years ago by Phil Karn, KA9Q, has been much refined and augmented and now exists in many versions that run on almost every kind of personal computer. Subsequent versions of Phil's software came to be called the Net Operating System (NOS) and we will use NOS as a generic name for the recent packages we use. These versions all perform file transfer (and most other TCP/IP-suite capabilities) in the same way.

Because NOS uses TNCs running in the KISS mode, it brings many of the protocols that usually run on modem-firmware during AX.25 packet operations back into the computer.[2] These protocols run as executable images of publicly available source code. The availability of this source code gives NOS developers, of which there are now many besides KA9Q, complete control over the protocols they run. This is not the case with most TNCs running pure packet.

The KA9Q spin-offs include most of the capabilities of their commercial cousins, such as PING (check paths to remote stations), FINGER (get info on remote stations) ARP (poll for the address of a remote station), TTYLINK (keyboard chat), TELNET (BBS operation), TCP (reliable transport-layer services), UDP (transport services less reliable than TCP's, used for piecewise path checks, etc), IP (network-layer services), SMTP (mail transfer), POP (automatic mail delivery), NNTP (news-bulletin delivery) and FTP (file transfer). For definitions and descriptions of these capabilities, most of which do not concern us here, see the fourth reference. Although several of these subpro-tocols can be used to send text files, we will be concerned mainly with FTP, which is the most versatile and efficient of the file-transfer protocols of the TCP/IP suite that do not employ compression.

File Transfer with FTP

The KA9Q FTP package can send both text and binary (also known as image) data, and connected stations can transfer files at the same time, although that slows down one-directional throughput. In the ASCII text mode, each line is processed separately, so throughput may be slightly lower than in the binary mode, which is suitable for graphics files, compressed files, executable images, etc. FTP file transfers use the transport control protocol (TCP) to achieve reliable data transmission.

TCP is a protocol that works at the *transport layer* of the open systems interconnection (OSI) model of modern data transmission systems. As a transport-layer protocol, TCP takes care of flow control and reliability of file transfer, among other things. Flow control is necessary to make sure that transferred file contents do not overflow the receiver's buffers, and that the receiver does not stand idle for too long while file data still needs to be sent. TCP passes data frames to the Internet protocol (IP), which provides OSI network-layer services such as message routing. By definition, IP and AX.25 (OSI link layer) services are generally tailored to particular networks and communications hardware and are not required to be highly reliable: data packets can get lost, delivered out of order or corrupted under those services with nothing done about it. Reliability is taken care of by TCP. The *applications* or *user processes* (like FTP) at OSI layers above the transport layer make only rudimentary reliability checks and work under the assumption that the transport layer will deliver reliable data to them.

TCP also contains sophisticated algorithms for dealing with channel congestion. One of these algorithms makes running estimates of the packet roundtrip time and resends packets after waiting for some multiple of the current estimate when *congestion* causes a packet to be lost. If, through this process, the time between repeats grows as large as a settable parameter called (in some implementations) the maximum roundtrip time (MRTT), then the waiting packet is resent. Unfortunately, on radio channels such algorithms sometimes confuse congestion with changes in SNR and become less aggressive in their repeats (in response to perceived congestion) when they should repeat more aggressively in response to a drop in SNR. This effect can be offset to some extent by listening to the channel (for possible real congestion) and lowering the MRTT when a drop in SNR is suspected and there is not a large number of outstanding packets (in the virtual circuit mode; see below).

Under the TCP protocol, raw data packets are first imbedded in *TCP-frames*, which have 20-byte headers (sender and receiver *port numbers*, sequence number, ACK number, checksum, etc). The TCP frames (called *datagrams*) are then encapsulated in Internet protocol (IP) frames, which also have 20-byte headers (IP addresses, frame length, checksum, etc). Thus, all TCP/IP data frames have 40 bytes of header data, which is one of the reasons for the modest TCP/IP throughput with 300-baud TNCs. In NOS, the IP frames are finally imbedded in AX.25 frames for transmission by a TNC. AX.25 frames also contain overhead bytes for call signs (including digipeaters), so data sent using FTP via TCP can have up to 20 + 20 + 72 = 112 bytes of overhead. In our point-to-point tests, the AX.25 frames have about 16 bytes of sender/receiver call sign data for a grand total of about 56 bytes of overhead in each sent AX.25 packet. Fig 1 shows what a TCP data frame looks like after encapsulation in an AX.25 frame.

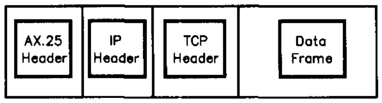


Fig 1

The size of the data frames (the data and the 20-byte TCP header are sometimes called a TCP *segment*) sent to the initial (TCP) layer for processing is set by a TCP parameter called the maximum segment size (MSS), which is normally no greater than about 200 bytes in HF operation. In fact, when using conventional TNCs, one should keep the MSS smaller than 216 (= 256 – TCP and IP header sizes) if AX.25-level *fragmentation* is to be avoided. Fragmentation happens when AX.25 packets are divided into subpackets by the TNC. This requires additional ACKs and usually lowers throughput.

Another parameter, called the TCP window (WIN), determines the number of bytes in the *sliding window* used for transport-layer flow control. This is the maximum buffer size in bytes the station can make available for incoming data. When stations link for FTP and most other types of communication sessions, they tell (*advertise* to) each other their WIN settings. The other station's WIN setting then gives each station the largest number of bytes it can have outstanding (ie, on the way to the other station) at one time. The largest segment a station can send is given in NOS by the minimum of the receiving station's advertised MSS and WIN values.[3]

TCP runs in either of two modes: *datagram* or *virtual circuit*. In datagram mode, each twice-encapsulated data frame is included in an AX.25 UI frame, and this UI frame is sent by the TNC to the radio for transmission. Acknowledgments and repeats are handled only at the TCP level and are accomplished by sending the number of the most recent correct byte (*octet*) rather than the number of a frame. A TCP receiver sends only acknowledgments; if the sender gets no acknowledgment of a data packet by a specific time-out period, it resends the packet. By setting its WIN value to an integer multiple of its MSS (while keeping its WIN *smaller* than the other station's advertised WIN), a sending station can send more than one TCP data segment (frame) at a time (namely a number equal to the multiple). This technique can lower ACK overhead and raise efficiency in some conditions. When that happens, throughput goes up significantly.

In the virtual circuit mode, each twice-encapsulated file data segment is included in an AX.25 I frame (rather than a UI frame) and this I frame is sent by the TNC to the radio for transmission. In this mode, however, acknowledgments and repeats are handled at both the TCP (transport) level *and* the AX.25 (link) level. The TCP and IP layers see the AX.25 layer in this mode as a quasi-reliable, dedicated hardware connection; ie, as a *virtual circuit*. This generally leads to higher reliability (lost or bad packets are immediately resent by the AX.25 level) at the expense of throughput due to the additional time spent handling ACKs and retries.

Since flow control is occurring in this mode at two levels, it is generally agreed that an accumulation of TCP, IP or AX.25 frames at any level must be avoided in the VC mode to prevent even bigger additions to ACK and repeat overhead. To prevent accumulation of TCP frames, one should keep the window size WIN equal to the segment size MSS. To prevent accumulation of AX.25 frames at the link level (where accumulation has the worst effect), one should make the AX.25 PACLEN big enough to comprise an MSS-sized data frame plus the 40 bits of TCP and IP header information. That is, one should make the parameter called AX PACLEN greater than MSS + 40 (remembering that this value should not exceed the TNC packet-size limit of 256). It is further recommended that the AX MAXFRAMES parameter be kept at one in the VC mode.

Our Measurement Strategy

In our experiments we first send a few PING packets to make sure that the HF channel is open and looks reliable in terms of SNR, tone quality and PING round-trip time.[4] If the channel looks reliable, the file-sending station establishes an FTP session with the *ftp call-sign* command. The FTPUSERS files at our receiving stations are set up beforehand to give the sender FTP write privileges. Various TCP parameters (see below) are also set beforehand at the receiving sites to give the sender wide leeway in adjusting his own TCP parameters (MSS, WIN, PACLEN and MRTT) for maximum throughput. All stations set their AX.25 and TCP RETRIES parameters fairly large to limit time-outs during difficult conditions, and backoff TIMERTYPES are set to LINEAR (rather than EXPONENTIAL) in the PC versions, since we use channels with little or no congestion (QRM).

After the FTP session is established, the sender decides whether he wants to transfer files in ASCII or binary mode and sends the appropriate command (*type a* or *type i*) to his software, which then sends it to the receiving station. After the receiver has confirmed the sender's login and the file type, the sender decides whether he wants to transfer in datagram or virtual circuit mode and sends the appropriate command (*mode ax0 dg* or *mode ax0 vc*) to his software. (Strictly speaking, we use what might be called the *half-VC* mode since receiving stations are always in the datagram mode.) Finally, the sender issues the *put filename* command, where *filename* is the name of the file to be sent. The sender's TCP/IP software then negotiates the transfer mode with the receiving station and proceeds to transfer the file using the FTP protocol.

If the transfer is in datagram mode, the sender now simply waits until the transfer is over, which TCP/IP announces with "File received OK" along with the transfer time in seconds, which we enter by hand into a data archive file. As the transfer proceeds, both sender and receiver may monitor progress by issuing the *tcp status* command, which gives the number of bytes transferred or received, among other parameters.

Shown below is an excerpt from a

TCP/IP *trace file* showing a few sent and received frames monitored at a file-sending station during an FTP text-file transfer from KB1JY to KB1PZ in the *datagram mode*. (This is one of several trace formats and is not necessarily the form the data take as they leave the TNC to go out over the air.) The trace data for each protocol layer are given on a separate line. The "len" entries in the IP lines give the size of data frames on the 0000 lines plus TCP and IP header sizes. The "Wnd" entries give the size of the raw-data frames.

```
Dec 10 20:53:53 ax0 sent:
AX25: KB1JY->KB1PZ UI pid=IP
IP: len 89 44.56.4.124->44.56.4.57 ihl 20 ttl 15 prot TCP
TCP: 1026->20 Seq x4b88b351 Ack xbc9da001 PSH ACK Wnd 50
0000 In the summer of 1994 measurements of the cod, fl

Dec 10 20:54:24 ax0 sent:
AX25: KB1JY->KB1PZ UI pid=IP
IP: len 89 44.56.4.124->44.56.4.57 ihl 20 ttl 15 prot TCP
TCP: 1026->20 Seq x4b88b351 Ack xbc9da001 PSH ACK Wnd 50
0000  In the summer of 1994 measurements of the cod, fl

Dec 10 20:54:31 ax0 recv:
AX25: KB1PZ->KB1JY UI pid=IP
IP: len 40 44.56.4.57->44.56.4.124 ihl 20 ttl 15 prot TCP
TCP: 20->1026 Seq xbc9da001 Ack x4b88b382 ACK Wnd 200

Dec 10 20:54:31 ax0 sent:
AX25: KB1JY->KB1PZ UI pid=IP
IP: len 89 44.56.4.124->44.56.4.57 ihl 20 ttl 15 prot TCP
TCP: 1026->20 Seq x4b88b382 Ack xbc9da001 PSH ACK Wnd 50
0000  ounder and haddock stocks over the Georges Bank a
```

Note that the AX.25 lines show that raw data are being sent as AX.25 UI frames. In this excerpt, the first line of text characters ("In the summer...") had to be sent twice before it got a TCP-level acknowledgment. The total overhead during this transfer is 20 + 20 + 5 + 5 = 50 bytes per frame, which is half the total frame content (= Wnd + overhead).

If the transfer is in virtual-circuit mode, the sender usually monitors the transfer status in three ways: he checks the transfer progress with the *tcp status* command. At the same time, he checks the timer status, which tells him the current estimate of the round-trip time (called the smoothed round-trip time, or SRTT) and whether it exceeds the maximum round-trip time he has set by the MRTT parameter. If it does exceed it, he may want to increase the value of MRTT to prevent a possible build-up of outstanding (unacknowledged) AX.25 frames, which can lower VC-mode efficiency catastrophically. To see if there is such a build-up, the sender issues the *ax status* command, which tells him the number of AX.25 packets in his sending queue. We have found that if this number is consistently about 3 or less, and the SRTT stays well below the MRTT, then the transfer is probably going well. If not, unacknowledged link-layer (AX.25) packets are probably building up and the MRTT needs to be increased.[5]

Here's a trace of some sent and received frames during a transfer in the VC mode:

```
Dec 10 21:35:18 ax0 recv:
AX25: KB1PZ->KB1JY RR(F) NR=4

Dec 10 21:35:19 ax0 sent:
AX25: KB1JY->KB1PZ I NR=0 NS=4 pid=IP
IP: len 59 44.56.4.124->44.56.4.57 ihl 20 ttl 15 prot TCP
TCP: 1030->20 Seq x54857667 Ack x15400001 PSH ACK Wnd 20
0000  the cod, flounder a

Dec 10 21:35:27 ax0 sent:
AX25: KB1JY->KB1PZ I(P) NR=0 NS=4 pid=IP
IP: len 59 44.56.4.124->44.56.4.57 ihl 20 ttl 15 prot TCP
TCP: 1030->20 Seq x54857667 Ack x15400001 PSH ACK Wnd 20
0000  the cod, flounder a

Dec 10 21:35:32 ax0 recv:
AX25: KB1PZ->KB1JY REJ(F) NR=5

Dec 10 21:35:34 ax0 sent:
AX25: KB1JY->KB1PZ I NR=0 NS=5 pid=IP
IP: len 59 44.56.4.124->44.56.4.57 ihl 20 ttl 15 prot TCP
TCP: 1030->20 Seq x54857667 Ack x15400001 PSH ACK Wnd 20
0000  the cod, flounder a

Dec 10 21:35:40 ax0 recv:
AX25: KB1PZ->KB1JY UI pid=IP
IP: len 40 44.56.4.57->44.56.4.124 ihl 20 ttl 15 prot TCP
TCP: 20->1030 Seq x15400001 Ack x5485767a ACK Wnd 200

Dec 10 21:35:41 ax0 recv:
AX25: KB1PZ->KB1JY RR NR=6

Dec 10 21:35:43 ax0 sent:
AX25: KB1JY->KB1PZ I NR=0 NS=6 pid=IP
IP: len 59 44.56.4.124->44.56.4.57 ihl 20 ttl 15 prot TCP
TCP: 1030->20 Seq x5485767a Ack x15400001 PSH ACK Wnd 20
0000  nd haddock stocks o
```

Note the more detailed information on the AX.25 lines, including the fact that the sender's AX.25 frames are now I frames. Also observe that the "...cod, flounder..." text had to be repeated at the AX.25 level before getting an AX.25 ACK. The total overhead during this transfer is again 50 bytes/frame, which is now about 70% of the total frame content. This not-unusual overhead percentage for VC transfers over this link points out the need for a more robust modem (with forward error correction, etc) that can reliably deliver longer frames and thus reduce the ratio of overhead to data in files transferred by TCP/IP.

It's pretty clear that trying to maximize throughput in the VC mode can keep a sender very busy, which is why we haven't done as many transfers in VC mode as in DG mode. If such transfers were ever to become a practical matter using 300-baud TNCs, the TCP/IP package would have to be modified to perform this monitoring and parameter-adapting automatically. This would be an interesting project, but whether it is worth doing depends on the near-term availability of more robust modems and link-layer protocols that can handle TCP data frames. A combination of automatic adjustment of MSS, WIN, MRTT, etc, with a more robust transmission system like GTOR, Clover or PacTOR II would be a formidable offering to those who want to do TCP/IP over HF.

The data archive file into which the results of each transfer are manually entered contains the date-time group, TCP mode (dg or vc), values of MSS and WIN, the TNC baud rate, the file size with and without line-processing overhead, the transfer time in seconds, the hand-calculated throughput in char/s (which doesn't have to be accurate since it's calculated later by the analysis software) and the file format (a or i). Here's an excerpt from the data file for tests with 1000-byte files run in November 1996:

```
29.11.96 16:33:00 vc 30 30 300 1000 1000 1096 0.91 [i]
29.11.96 17:07:00 dg 50 150 300 1000 1000 428 2.34 [i]
29.11.96 17:51:00 vc 20 20 300 1000 1000 1403 0.71 [i]
29.11.96 18:22:00 vc 10 10 300 1000 1000 1688 0.59 [i]
29.11.96 19:59:00 vc 20 20 300 1000 1000 1005 0.99 [i]
29.11.96 20:16:00 dg 100 100 300 1000 1000 222 4.50 [i]
29.11.96 20:21:00 dg 100 200 300 1000 1000 172 5.81 [i]
```

This file is opened and analyzed by a data-analysis program described in the next section.

The Data-analysis Software

The results in the data archive are analyzed off-line by a program called *summary_tcp.c*. This program reads the archive file line-by-line looking for various strings. As it moves through the file to the end-of-file character, the program keeps running totals of throughput and other data corresponding to the strings, from which it calculates statistics such as the average and standard deviation of the throughput. The statistics are printed to a summary file after the pass through the archive file. Switches in the summary code are set before each run to pick out specific data (corresponding to various string combinations) for analysis. (For example, we select the "vc" and "[a]" strings for a run to calculate the throughput for files sent in the virtual circuit mode as ASCII text. Since the summary program was written to analyze archive files of fixed format but arbitrary length, summaries of the data collected so far can be made at any time.

Shown below is the output of the summary program for all the NVIS tests run up to December 1996. For this output we set the software switches to compute throughput statistics for files sent as ASCII text in the datagram mode.

```
Statistical Summary of TCP/IP Throughput Tests:
10.12.96 19:18:19

NUMBER OF TCP TRANSFERS IN SAMPLE = 135
E(MSS) = 93.78 bytes, sd(MSS) = 44.29 bytes
E(WIN) = 135.23 bytes, sd(WIN) = 82.62 bytes
E(BAUD RATE) = 227.41 bps, sd(BAUD RATE) = 46.41 bps
E(CHARS TRANSFERRED) = 1459.3, sd(CHARS TRANSFERRED) = 803.6
E(TRANSFER TIME) = 571.7 s, sd(TRANSFER TIME) = 462.9 s
E(THRUPUT)  =  3.39  cps,  sd(THRUPUT)  =  1.87  cps,
sd(mean_THRUPUT) = 0.161 cps
max_THRUPUT = 9.80 cps, E(THRUPUT/Hz) = 0.007 cps/Hz
```

The output shows that the average throughput in this case for 135 transfers was about 3.4 characters per second (cps) and that the largest observed throughput in this mode was about 10 cps. The sd(THRUPUT) reflects the spread of the throughput measurements about their average. Roughly speaking, about two-thirds of a set of measurements will be within one standard deviation of their mean and over 90% will be within two standard deviations of their mean.

We also calculate the *standard deviation of the mean* [sd(mean_ THRUPUT)] of the throughput (in characters per second) and the average throughput per Hertz of signaling bandwidth. The standard deviation of the mean (equal to the standard deviation of the throughput divided by the square root of the sample size) is an assessment of the variability of the mean itself (which has its own statistical variability). The sd(mean) above suggests that our sample size in this case is big enough to give us pretty high confidence that if we collected many more throughput measurements under roughly the same conditions, we would not get an average throughput that differed from the one above by more than about two-tenths a character per second.

To calculate the average throughput per Hertz [E(THRUPUT/Hz)], we divide the average throughput by the average signaling bandwidth. We calculated the latter using the formula for "necessary telegraphy bandwidth" (from the 1992 Department. of Commerce *RF Management Handbook*) BW = *baud rate* + 1.2 × *shift*, where *shift* for most of our tests was 200 Hz. For our tests we used either 200 or 300 baud, so we took 250 baud as the baud rate in the formula. (As the output above shows, this is usually 10 or 20 baud off, but this has little effect on the results.) The resulting average bandwidth is 490 Hz.

Layout of Paths and Discussion of Antennas and NOS Software Versions

The stations used for the NVIS tests are in Bedford, Massachusetts (KB1JY); Norfolk, Massachusetts (W1IMM); Derry, New Hampshire (KB1PZ); and Augusta, Maine (KB1JY). Bedford used an 80-m dipole up 30 feet for most tests and occasionally a terminated, bottom-fed 125-ft longwire pointing southwest (not, of course, ideal for these tests). Norfolk used an 80-m dipole up 40 feet. Derry used an off-center-fed 80-m dipole up 30 feet. Augusta used an unterminated, top-fed, 125-ft sloping longwire running northeast to southwest. The links (followed by lengths and rough estimates of the percentage of data collected over each link) are:

Bedford-Norfolk (35 miles, 35%)
Bedford-Derry (25 miles, 35%)
Augusta-Norfolk (200 miles, 15%)
Augusta-Derry (150 miles, 15%).

All of these links run more or less north-south.

KB1JY, the sender in most of the tests, ran NET/Mac2.3.57. The receiving stations at KB1PZ and W1IMM ran JNOS1.10M and MFNOS1.13. NET/Mac lacks some of the capabilities of the NOS (PC) versions, such as exponential backoffs in the presence of congestion, but it has all the capabilities we needed for our tests.

Statistical Summary of results

The rest of the results of our NVIS tests (as of December 1996) are given below as output from three more runs of the *summary_tcp* program. These are for the combinations of transfer mode and file type *DG* and *i*, *VC* and *a* and *VC* and *i*. The meanings of the entries are explained above.

```
Statistical Summary of TCP/IP Throughput Tests:
11.12.96 15:35:23

DG/i

NUMBER OF TCP TRANSFERS IN SAMPLE = 157
E(MSS) = 72.39 bytes, sd(MSS) = 50.73 bytes
E(WIN) = 135.54 bytes, sd(WIN) = 82.88 bytes
E(BAUD RATE) = 285.35 bps, sd(BAUD RATE) = 35.47 bps
E(CHARS TRANSFERRED) - 1467.5, sd(CHARS TRANSFERRED) = 1321.8
E(TRANSFER TIME) = 425.3 s, sd(TRANSFER TIME) = 310.3 s
E(THRUPUT)  =  4.58  cps,  sd(THRUPUT)  =  3.11  cps,
sd(mean_THRUPUT) = 0.248 cps
max_THRUPUT = 13.57 cps, E(THRUPUT/Hz) = 0.009 cps/Hz

VC/a
```

```
NUMBER OF TCP TRANSFERS IN SAMPLE = 43
E(MSS) = 87.79 bytes, sd(MSS) = 37.04 bytes
E(WIN) = 87.79 bytes, sd(WIN) = 37.04 bytes
E(BAUD RATE) = 258.14 bps, sd(BAUD RATE) = 49.92 bps
E(CHARS TRANSFERRED) = 1441.9, sd(CHARS TRANSFERRED) = 628.8
E(TRANSFER TIME) = 810.1 s, sd(TRANSFER TIME) = 440.9 s
E(THRUPUT)   =   2.41   cps,   sd(THRUPUT)   =   1.68   cps,
sd(mean_THRUPUT) = 0.257 cps
max_THRUPUT = 8.23 cps, E(THRUPUT/Hz) = 0.005 cps/Hz

VC/i

NUMBER OF TCP TRANSFERS IN SAMPLE = 70
E(MSS) = 70.71 bytes, sd(MSS) = 47.44 bytes
E(WIN) = 70.71 bytes, sd(WIN) = 47.44 bytes
E(BAUD RATE) = 295.71 bps, sd(BAUD RATE) = 20.40 bps
E(CHARS TRANSFERRED) = 1206.4, sd(CHARS TRANSFERRED) = 549.4
E(TRANSFER TIME) = 825.6 s, sd(TRANSFER TIME) = 547.1 s
E(THRUPUT)   =   2.05   cps,   sd(THRUPUT)   =   1.50   cps,
sd(mean_THRUPUT) = 0.179 cps
max_THRUPUT = 6.58 cps, E(THRUPUT/Hz) = 0.004 cps/Hz
```

The data in these outputs are shown in Table 1. The first column gives the average throughput and its standard deviation, the average throughput per Hertz, the standard deviation of the mean throughput and the maximum observed throughput. The second column gives the number of transfers in each case. The third column gives the mean and standard deviation of MSS, WIN and the TNC baud rate. The fourth and fifth columns give the means and standard deviations of the transfer time and the number of transferred characters.

Discussion of Results

Like packet radio (only more so), TCP/IP over NVIS paths with a TNC is very much a daytime occupation, at least for those who want halfway tolerable throughput. At night, especially with the sunspot cycle near minimum, there is usually so much multipath, noise and interference on the small band of frequencies that lie below the MUF that FTP transfers are either impossible, or the sport of only those willing to go without sleep trying transfers at *very* low baud rates. (An automatic link establishment [ALE] system, such as prescribed in MIL-STD-188-141A, could probably have found a useful frequency even at night if given enough choices and an antenna suitable for both surface waves and NVIS.)

As expected, when propagation is good (usually about 10 AM to 4 PM local time), the highest throughput is achieved in the datagram mode when multiple TCP frames can be sent in a frame window, cutting down greatly on ACK overhead. We found that MSS/WIN combinations like 50/200 or even 100/200 often led to the steady, satisfying screech of datagrams being sent and acknowledged without retries during all or most of a transfer at such times. Outside these times—but still during the day— smaller frame sizes and single-frame windows often work, but throughput is nothing to crow about.

In contrast, PacTOR (with Huffman compression on) and GTOR achieve average throughputs of about 18 and 24 cps on NVIS links, and CLOVER II over 40 cps in its LZW compressed mode.^{6} The wide variability of throughput in both DG and VC modes reflects to some extent the amount of manual parameter adapting we performed, especially during virtual circuit transfers.

Sending (text) files in binary mode raises throughput by 10 to 30% because the extra overhead used to process individual lines is avoided. (That this did not occur in VC mode is due to the fact that most of the VC transfers in binary mode were done over paths starting in Augusta, where there is sometimes power-line noise and a resistively terminated antenna that keeps output at about 60-70 W.)

Virtual-circuit-mode transfers do show a persistence not always seen in the datagram mode, but they yield about half the throughput of DG transfers. This is probably due simply to the additional overhead needed to handle the link-layer (AX.25) ACKing and retrying.

It's conceivable that wild and crazy experimentation with multiple TCP frames (WIN = N*MSS for N > 1) or even multiple AX.25 frames (MAXFRAMES > 1) in this mode may have yielded higher throughput on occasion, but it would have required such feverish (and confusing) performance monitoring and parameter adjusting that we let our courage fail us. This is an area of study for those equipped with faster and more reliable waveforms and robust error-control schemes.

Like HF packet, TCP/IP over HF is a pursuit ripe for application of an automatic adaptive protocol, perhaps one that adjusts MSS, WIN, MRTT, etc, along with link-level parameters like baud rate, interleaver depth and FEC performance that are normally connected with functions carried out in a modern modem.

Concluding Remarks

We hope that our data will aid discussions of TCP/IP file transfer over HF and perhaps serve as a useful introduction to this fascinating topic. Although the data confirm that throughput is modest over NVIS paths (but not much lower in the datagram mode than AMTOR [5 cps] and packet [6 cps]), it may surprise some that TCP/IP works at all over HF using TNCs running at no more than 300 baud. Government and other programs are now busy developing and testing systems that send TCP frames using much more effective modems (such as the MIL-STD-188-110A serial-tone type that can run at 2400 bps). The first application of this work is HF e-mail, but other TCP/IP applications will no doubt follow.

Acknowledgment

We are grateful to Gary Grebus, K8LT, for useful discussions of the datagram and virtual circuit connection modes.

About the Authors

Mike Bernock, Bob Levreault and Ken Wickwire work for the MITRE Corporation near Boston, where they have been involved for several years in digital and voice communications in the HF and VHF bands. Mike has worked for over ten years on SINCGARS, the Army's tactical VHF combat net radio. Bob and Ken have also been involved in the development and use of the military and federal standards for HF automatic link establishment (ALE), advanced HF data modems and robust HF data-transmission protocols. They participate regularly in SHARES preparedness exercises and in the recently established worldwide experimental ALE network run by the DoD. All are regular users of the club station at MITRE (W1ON), which runs digital gateways for packet radio, NET/ROM, TCP/IP and the APRS tracking system. They long ago accepted the fact that it is impossible to keep radio out of one's private life, whether it belongs there or not.

References

Phil Karn, KA9Q, *NET User Reference Manual* (NOS Version), 1991.

Gary Ford, N6GF, *Beginner's Guide to TCP/IP on the Amateur Packet Radio Network Using the KA9Q Internet Software*; Versions 1 and 2, 1990 and 1992.

John Ackerman, AG9V, *Getting Started with TCP/IP on Packet Radio*, 1992.

John Spragins, et al, *Telecommunications Protocols and Design*, Addison-Wesley.

Stan Horzepa, WA1LOU, "HF Throughput Truths," in Packet Perspectives, *QST*, Feb 1996.

Ken Wickwire, KB1JY, et al, "On-air Measurements of HF TOR and Packet Throughput, Part I: Near-Vertical-Incidence-Skywave Paths," *Digital Journal*, March 1996.

Ken Wickwire, KB1JY, "On-air Measurements of HF TOR and Packet Throughput, Part II: One-hop Skywave Paths," *QEX*, June 1996.

Ken Wickwire, KB1JY, "On-air Measurements of HF Data Throughput: Results and Reflections," *Proceedings of the 15th ARRL/TAPR Digital Communications Conference*, ARRL, 1996.

Notes

[1]This is not a shortcoming of the KA9Q code, but rather of the non-error-correcting modems used to send TCP/IP frames over the HF channel.

[2]This is where some modern, robust HF data-transmission systems, like FED-STD-1052, do a lot of their processing above the link level, leaving only FEC, interleaving and equalizing to their HF modems.

[3]To allow for at most one outstanding packet (the maximum often recommended for poor channels), one should set the WIN value equal to twice the MSS.

[4]For HF point-to-point work we put "route add default axN" into our autoexec.nos/net setup files to route all traffic directly to called stations through the HF port (N is the number of that port in each case).

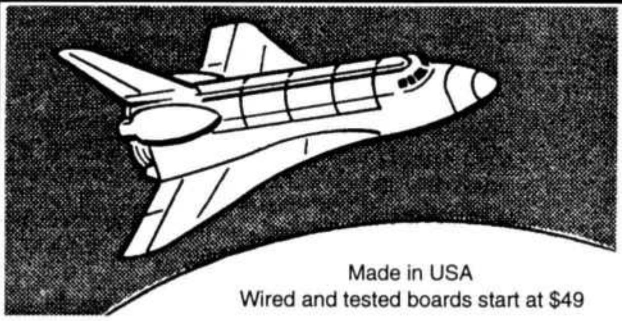
[5]Occasionally, increasing the MRTT happens too late to save a sender who hasn't set MSS and WIN small enough for current channel conditions. Then the only sensible remedy is to abort the transfer or even reset (break off unilaterally) the session.

[6]Some NOS versions can also compress files using the Lempel-Ziv-Welch (LZW) algorithm, but we didn't try that. Compression might have at least doubled average throughput. □

Table 1
Statistical Summary of TCP/IP Throughput Data

Mode	*E(thruput)*	*No. Xfers*	*E(MSS)*	*E(xfer_tm)*	*E(No_char)*
	sd(thruput)		*E(WIN)*	*sd(xfer_tm)*	*sd(No_chr)*
	E(tput/Hz)		*E(BAUD R.)*		
	sd_mn(tput)				
	max_tput				
Datagram	3.39 cps	135	94	572 s	1459
ASCII	1.87 cps		135	463 s	804
	0.007 cps/Hz		227 bps		
	0.16 cps				
	9.80 cps				
Datagram	4.58 cps	157	72	425 s	1468
Binary	3.11 cps		136	310 s	1322
	0.009 cps/Hz		285 bps		
	0.25 cps				
	13.57 cps				
V.Circuit	2.41 cps	43	88	810 s	1442
ASCII	1.68 cps		88	441 s	629
	0.005 cps/Hz		258 bps		
	0.26 cps				
	8.23 cps				
V. Circuit	2.05 cps	70	71	826 s	1206
Binary	1.50 cps		71	547 s	549
	0.004 cps/Hz		296 bps		
	0.18 cps				
	6.58 cps				

AMATEUR TELEVISION
Web site: www.hamtv.com



Made in USA
Wired and tested boards start at $49

SEE THE SPACE SHUTTLE VIDEO

Many ATV repeaters and individuals are retransmitting Space Shuttle Video & Audio from their TVRO's tuned to Spacenet 2 transponder 9. Others may be retransmitting weather radar during significant storms or home camcorder video. If it is being done in your area on 420 MHz - check page 554 in the 96-97 ARRL Repeater Directory or call us, ATV repeaters are springing up all over - all you need is one of the TVC-4G ATV 420-450 MHz downconverters, add any TV set to ch 2, 3 or 4 and a 70 CM antenna (you can use your same 435 Oscar beam). We also have downconverters and antennas for the 902-928 & 1240-1300 MHz bands. In fact we are your one stop for all your ATV needs and info - antennas, transceivers, amps, etc.
Hams, call for our complete 10 page ATV catalogue!

Low Cost Start

Model TVC-4G
ATV Downconverter
tunes 420-450 MHz to ch 3
only $89
TVC-9G 900 MHz - $99
TVC-12G 1200 MHz - $109

Get The ATV Bug

Companion TX70-1b
1.5W ATV
TRANSMITTER
only $279
Buy both save $19
Full color & sound
Plug in your camcorder, antenna & 13.8 Vdc @ 1A

CALL (818) 447-4565 M-Th 8AM - 5:30 PM PST. VISA, MC, UPS COD
P. C. ELECTRONICS
2522 S. PAXSON Lane ARCADIA CA 91007
Email: tomsmb@aol.com
24 hr FAX (818) 447-0489
Tom (W6ORG) & MaryAnn (WB6YSS)

A Development System for the 8051 Microcontroller

A microcontroller can replace a lot of control circuitry. Here's a tool to get you started.

By Anthony L. Marchese, N2YM

My Grandpa always told me to use the right tool for the job. Grandpa was talking about woodworking tools but the same rule applies to computers. Let me put this in perspective; when we say the word computer, most envision glittering, lightning-fast systems with gigabyte hard drives, CD-ROMs and high-resolution graphics. Now think about using this same, expensive, high-tech, electronic masterpiece to control the temperature of your coffee pot. Not on your life.

The right tool for the job is a special class of microprocessor, known as a microcontroller, which is intended for applications where simplicity, space, weight, power and cost are primary concerns. The microcontroller fills this niche by combining a variety of features such as memory, input/output (I/O), counter/timer functions, serial communications and A/D conversion with a central processing unit (CPU), arithmetic logic unit (ALU) and special-purpose registers on a single IC.

The 8051 8-bit microcontroller is used for advanced, sequential applications and provides an inexpensive introduction to the world of embedded control. However, the development of any computer-based application without a means to analyze the hardware and software interaction, or lack thereof, can be frustrating, to say the least. The development system described here provides a convenient platform for hardware and software debugging of 8051 microcontroller-based applications.

This project uses the 80C31, which is the ROM-less, low-power version of the 8051. The device provides five vectored interrupts with two priority levels, two 16-bit event counter/interval timers, 32 I/O lines, 128 bytes of internal RAM and a full-duplex, programmable serial port all in a single package.

The 8051 supports a 111-element instruction set. The development board utilizes a 6-MHz clock, which allows 49 of the instructions to execute in 2 mS. The multiply and divide commands are the slowest of all instructions, requiring 8 mS to execute. The instruction set takes advantage of the controller's extensive Boolean processing (single bit) capabilities. The AND, OR, set/clear and complement instructions allow manipulation of

35 Shannon Crescent
Spencerport, NY 14559-9758

individual bits in the accumulator, I/O registers and in an area of internal RAM. Conditional branches and relative jumps, based upon the results of either byte-wide register comparisons or the status of individual bits, simplify event-based decisions. The ability to directly address and manipulate single bits is of particular importance in real-time situations, considering that functions must often execute within the time constraints of the given application. The 8051's Boolean processing capabilities minimize program code, thereby reducing the execution time required for the management of the extensive binary input and output inherent to sequential control applications.

Software

The brains of the development board is "ELMER," an *E*mbedded *L*ow-level *M*onitor for *E*xperiments and *R*eal-time debugging. A debugger is simply a group of software routines that allow the user to examine and manipulate data during application development. ELMER contains a variety of routines for this purpose, shown in Table 1.

The command format is short to simplify entry and reduce typing. A carriage return <CR> is required after each line but has been left off for clarity. ELMER is somewhat finicky, as any attempt to correct a command line results in an error message. Further, ELMER insists upon capital letters, hex addresses and hex (in one case binary) data. Spaces are not required but may be used to separate the command from the data/address.

ELMER occupies addresses $0000 through $07FF in the U2 EPROM. Since the 8051 interrupt vectors are located within this address space ($0000 to $002B), ELMER assigns the vectors to external RAM locations:

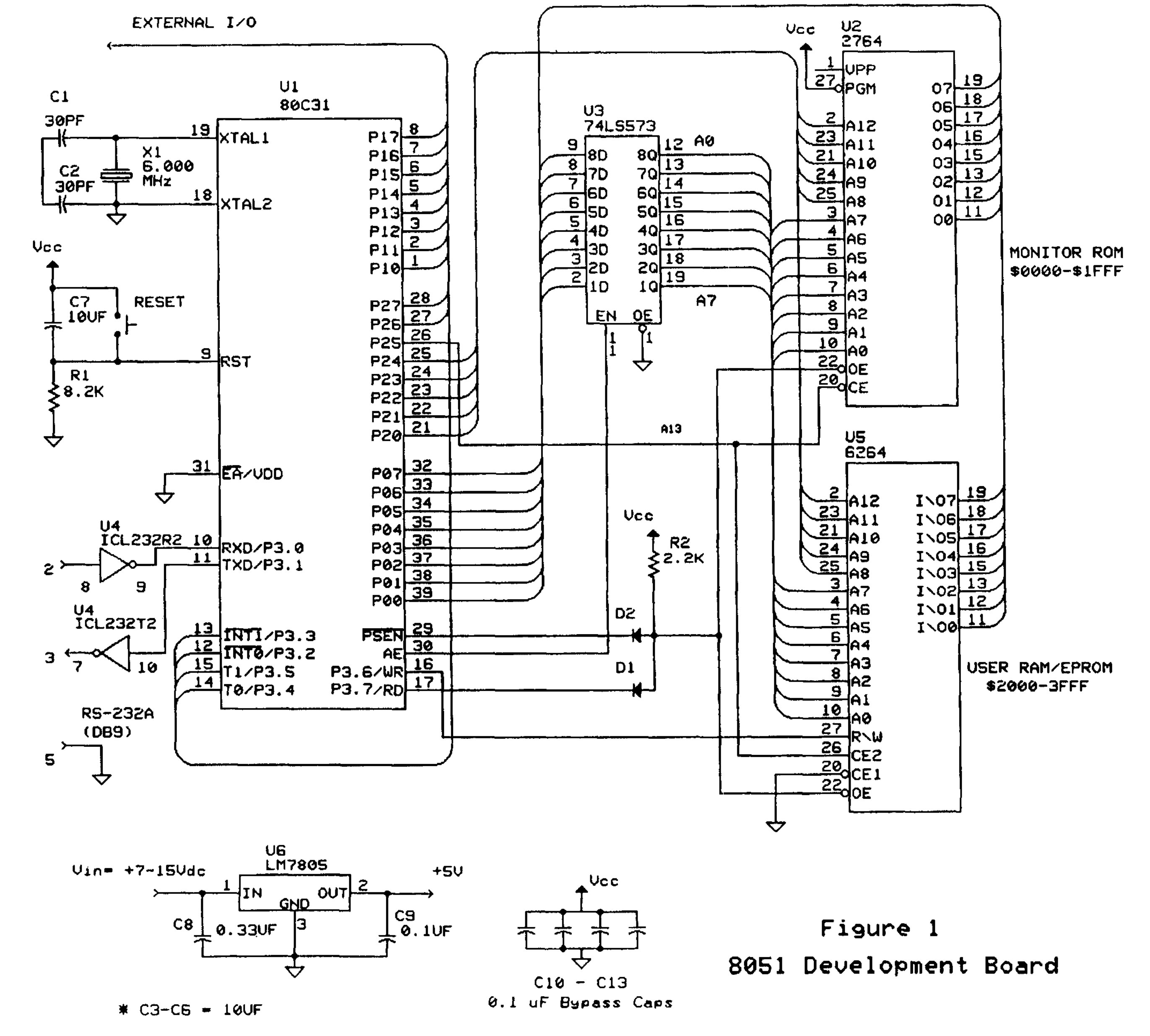


Figure 1
8051 Development Board

Fig 1

IE0	external interrupt 0	$3A00
TF0	timer0 overflow	$3B00
IE1	external interrupt 1	$3C00
TF1	timer1 overflow	$3D00
RI/TI	serial RX/TX INT	$3E00

The monitor program initializes the stack to internal RAM location $30, which provides the user with 47 bytes of unused space before bumping into the start of ELMER's scratch pad. The 80C31's four register banks, as well as bit addressable portions of internal memory, are located below the stack. However, avoid the use of Register Bank 0 as the monitor routines use R0-R7 for data manipulation. ELMER's external RAM usage is limited to locations $3FF0 to $3FFF. Monitor routines temporarily store program segments in these locations to facilitate access to internal memory locations $80 to $FF, which are only available with direct addressing.

Hardware

The development board compensates for the 80C31's lack of internal ROM by strapping the controller's External Access (EA) line to ground. Ports 0 and 2 are used for the multiplex low-address/data and high-address busses in this configuration, which prevents their use as general-purpose I/O. Port 3 lines 6 and 7 also assume alternate functions as the external data memory write and read strobe in this mode. The system boots from U2, an external 2764, 8k × 8 EPROM, which contains the monitor routines and occupies addresses $0000 through $1FFF. U5, a 6264, 8k × 8 RAM, occupies addresses $2000 through $3FFF. The 8051 logically separates program and data memory and can address up to 64k of each. The development board combines the controller's PSEN and RD signals with a diode AND gate to allow all external memory to be addressed as either program or data. This minimizes reprogramming of EPROMs, as code can be downloaded, executed and debugged in RAM.

An LM7805 regulator converts an external dc power supply to the +5 V required by the development board. The supply output at 80 mA should be greater than +7 V to maintain regulation but less than +15 V as higher levels cause considerable warming of the LM7805.

Table 1—ELMER Debugger Commands

Command	*Description*
B	Display current breakpoint and register data.
B*aaaa*	Set breakpoint at hex address *aaaa*. ELMER disables all interrupts when the breakpoint is reached, then displays the data in the registers defined in the **R** command. Breakpoints can only be set in RAM, as ELMER replaces the original program code with a small subroutine.
B–	Clear the breakpoint and restore the original program code.
C	Clear external RAM locations $2000-$3FFF.
D*aaaa*	Display (16) external memory locations beginning at location *aaaa.*
G*aaaa*	Execute subroutine at external memory location *aaaa*. The RET instruction must be present at the end of the user's program to pass control back to ELMER.
I*aa*	Display internal RAM location *aa*. The user may modify the location by entering the appropriate hex characters after the original data is displayed. Entering a <CR> without preceding hex characters causes the program to return to the monitor prompt without modifying the location.
L	Upload Intel hex file from serial port to RAM. Type **L** to initiate the transfer process. Please note, the upload routine verifies file structure and checksums but does not check address boundaries. This allows the system to be expanded if additional memory is required. The use of .ORG or similar statements in the program will ensure that the uploaded code resides in the external RAM space ($2000-$3FFF).
M*aaaa*	Modify external RAM location *aaaa*. This routine displays the memory address followed by the original data. The user may enter new hex data to modify the location. The next address and data is then displayed to allow the user to modify multiple locations without having to repeatedly enter the command line. A <CR> without preceding data causes returns the user to the command prompt.
P*nb*	Display a whole port *n* or a single bit *nb*. The user may modify the location by entering the appropriate hex characters for the whole port or a **1** or **0** for the single-bit modify after the original data is displayed. Entering a <CR> without preceding data causes the program to return to the monitor prompt without modifying the port.
R	Display special function registers including the data pointer (DPTR), program status word (PSW), stack pointer (SP) and the A & B accumulators.
?	Displays a menu of available commands, including the correct format and a brief description of the function.

System Setup

Program development is typically performed on a host system such as a personal computer. Programs are written as text files, then translated into an encoded object file for eventual transfer to the development system. Two additional pieces of software are required for these tasks:

1. An 8051 assembler converts a text file containing mnemonics into an object file for transfer to the development board. A compiler, such as the C-8051 cross compiler offered by Archimedes Software, may be used in place of the assembler. The compiler provides a method for those who prefer to code in higher-level languages such as C to convert programs into 8051-based object files. The development board contains a routine to upload Intel hex format object files into external RAM from a PC attached to the serial port.

2. A communication program, such as Microsoft's Window *Terminal* or *Procomm* from Datastorm Technologies, Inc, will also be required for the host system to communicate with the development board. A standard serial cable can be used to connect the TxD, RxD and signal ground lines from the development board's full-duplex RS-232 port to a host system such as a personal computer. The serial port is con-

figured for 2400 baud, 8 data bits, no parity and 1 stop bit. None of the RS-232 handshake lines or any form of software flow control are required for data transfer.

Construction

The schematic for the development board is provided in Fig 1. The artwork and layout for a double-sided PC board are shown in Figs 2, 3 and 4. I prefer to use the PC board for the units I build, but the board may be also be constructed using either wire-wrap or point-to-point methods. You can construct the unit in one evening if you use the printed circuit board. Two or three evenings may be required with the other methods. It is not necessary, but you may wish to socket U1-U5. You will also need access to a programmer for the 2764 EPROM. The assembled list and object files for the monitor software are available.[1]

Conclusion

The development board provides a platform for the design and testing of 8051-based application circuits. The ability to develop applications without having to reinvent the wheel has saved me countless hours of work and frustration. The ultimate goal is to be able to debug the application, then to reduce the hardware by replacing the development board with a single IC: the 8751, which is an 8051 with 4k of internal EPROM. However, a special programmer is required for this. I'll bet that I can use the 8051 development board to construct an 8751 programmer, but as Grandpa used to say, "That's another story."

Notes

[1]The files can be found at **http://www.arrl.org/qexfiles/mar8051.zip**.

Table 2
Parts Sources

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002-4100
800-831-4242 (tel order line)
415-592-8097 (tel)
415-592-2503 (fax)

Prime Electronics
150 West Industry Court
Deer Park, NY 11729
516-254-0101

TASM - 8051 assembler:
Speech Technology Inc
Software Division
837 Front Street South
Issaquah, WA 98027

Archimedes Software
2159 Union Street
San Francisco, CA 94123
415-567-4010

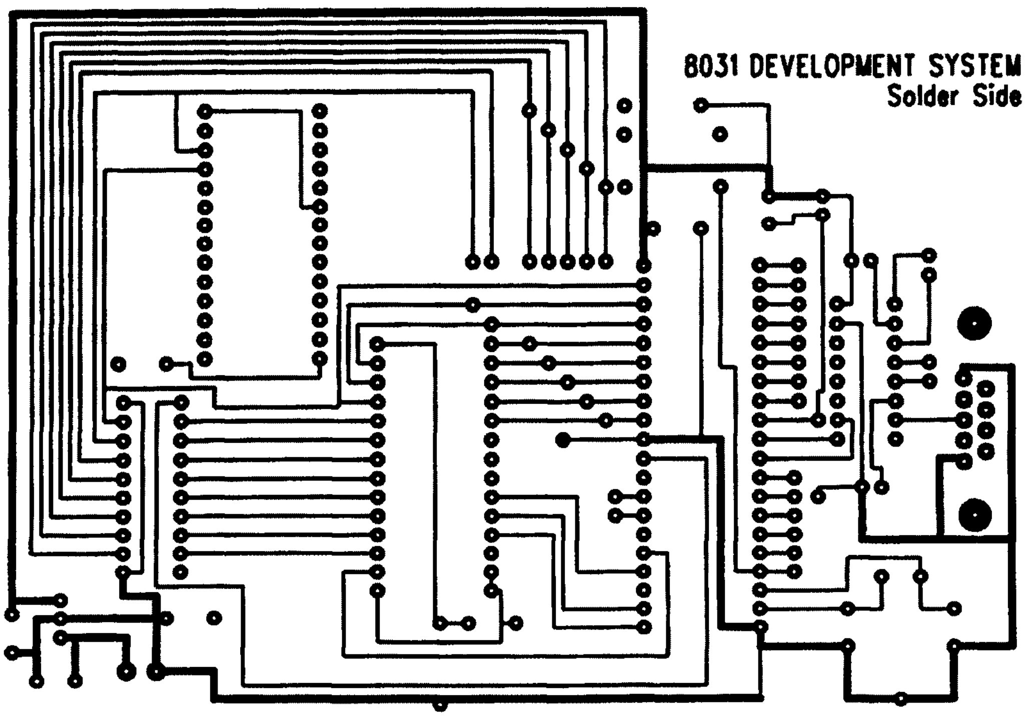


Fig 2

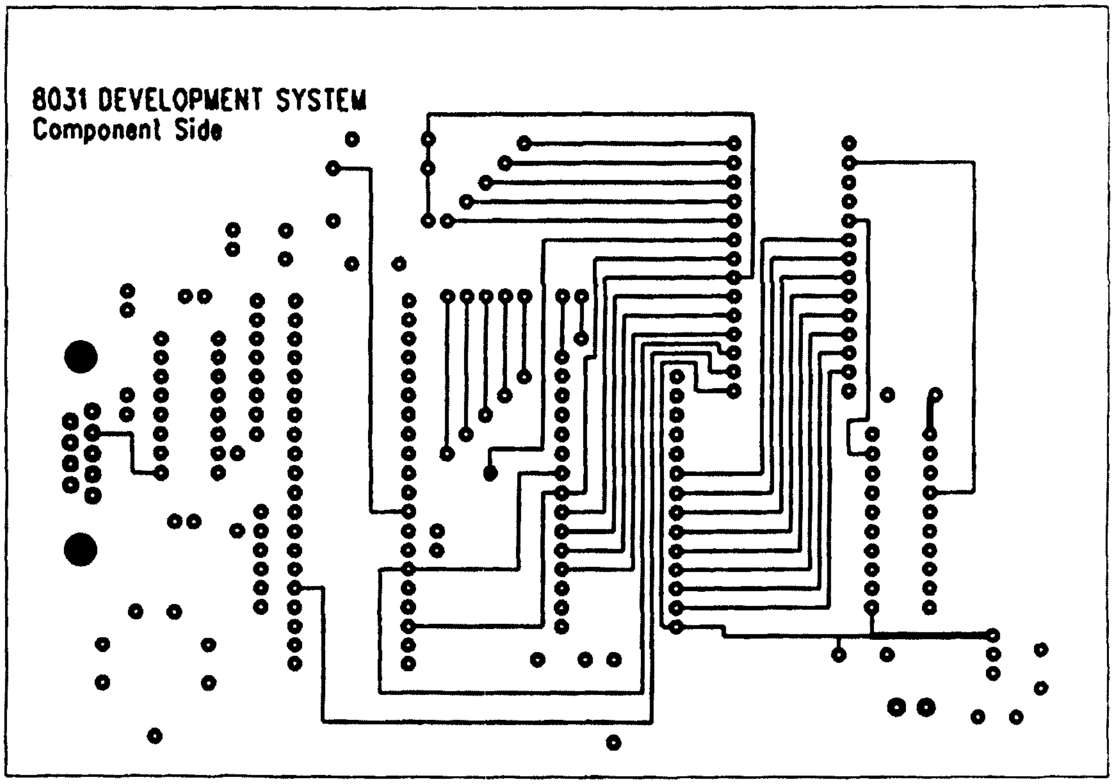


Fig 3

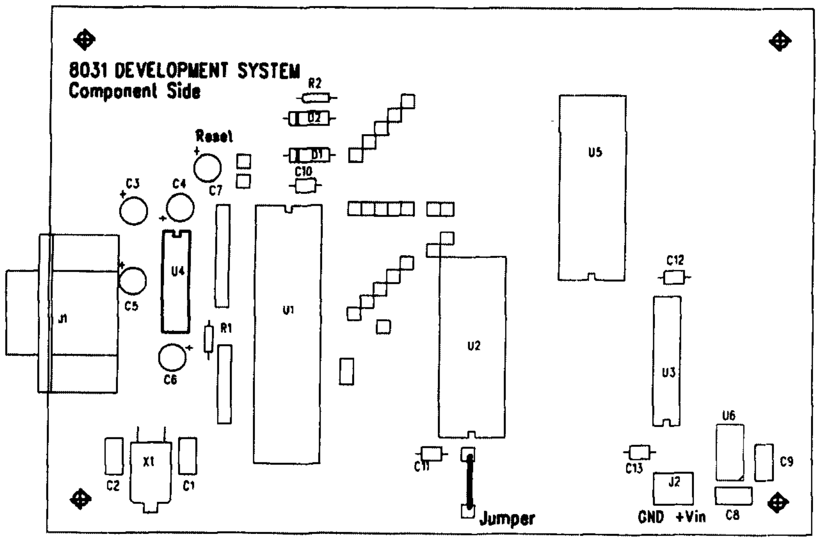


Fig 4

TAPR **Books**

NOSIntro by Ian Wade, G3NRW	\$ 23.00
TAPR 1994 Meeting Papers	\$ 6.00
TAPR Packet Radio General Info	\$ 6.00
Past TAPR 4 yr PSR sets (each)	\$ 20.00

ARRL Digital Conference Proceedings

1st - 4th	(1981, '83, '84, '85)	\$ 18.00
5th	(1986) Orlando, FL	\$ 10.00
6th	(1987) Redondo Beach, CA	\$ 10.00
7th	(1988) Columbia, MD	\$ 12.00
8th	(1989) Colorado Springs, CO	\$ 12.00
9th	(1990) London, ONT Can	\$ 15.00
10th	(1991) San Jose, CA	\$ 10.00
11th	(1992) Teaneck, NJ	\$ 10.00
Entire Set (1981 - 1992)		***\$ 80.00***

TAPR Members receive 10% discount on all kits and publications. See classified for *special* membership offer. Contact TAPR for details and a brochure on all our kits and services.

TAPR • 8987-309 E Tanque Verde Rd #337
Tucson, Az • 85749-9399 • (817) 383-0000
TAPR is a Non-Profit R & D Corporation

1996 Index

January 1996 to December 1996 QEX
(issues 167 through 178)
All dates are 1996 unless otherwise indicated.

Feedback

I have found (or have had pointed out to me) the following errors/ommissions in my article, "A Practical Approach to Computerized Control of Stepper Motors and Relays," October 1996 *QEX*. I also want thank everyone who's taken the time to write and express their comments.—*Ron Pierce, NI0L*

1) Fig 1 is not captioned. It should be: Manual test board schematic.

2) Fig 3. The caption should say: CD 4514 4-bit latch/4 to 16-line decoder.

3) Fig 4. The relay contacts should have been labeled RY1,RY3,RY5,RY7, etc.

4) Fig 5. There is a wiring error. The jumper from the top of L4 should go to the top contact of RY3, not to the bottom contact of RY4.

5) Fig 7 is not captioned. It should read: A typical hookup for a 5-wire motor."

6) The text in the middle of the right hand column of page 7 should say: See circuit details in Fig 6.