QEX

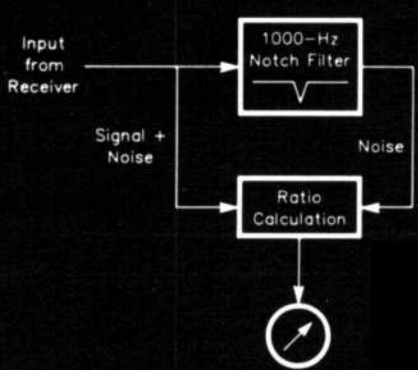
$1.75

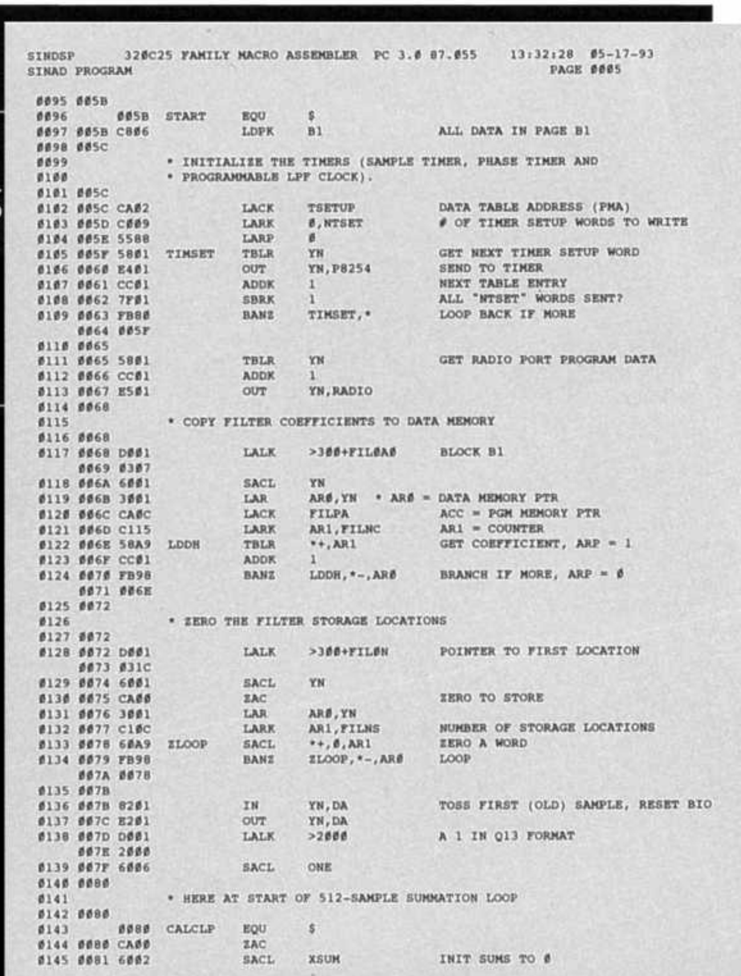
ARRL Experimenter's Exchange

June 1993

A Tool for Measuring Receiver SINAD?

DSP!





```
SINDSP      320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                     PAGE 0005

 0095 005B
 0096      005B  START    EQU    $
 0097 005B C806           LDPK   B1               ALL DATA IN PAGE B1
 0098 005C
 0099                  * INITIALIZE THE TIMERS (SAMPLE TIMER, PHASE TIMER AND
 0100                  * PROGRAMMABLE LPF CLOCK).
 0101 005C
 0102 005C CA02           LACK   TSETUP           DATA TABLE ADDRESS (PMA)
 0103 005D C009           LARK   0,NTSET          # OF TIMER SETUP WORDS TO WRITE
 0104 005E 5588           LARP   0
 0105 005F 5801  TIMSET   TBLR   YN               GET NEXT TIMER SETUP WORD
 0106 0060 E401           OUT    YN,P8254         SEND TO TIMER
 0107 0061 CC01           ADDK   1                NEXT TABLE ENTRY
 0108 0062 7F01           SBRK   1                ALL "NTSET" WORDS SENT?
 0109 0063 FB80           BANZ   TIMSET,*         LOOP BACK IF MORE
      0064 005F
 0110 0065
 0111 0065 5801           TBLR   YN               GET RADIO PORT PROGRAM DATA
 0112 0066 CC01           ADDK   1
 0113 0067 E501           OUT    YN,RADIO
 0114 0068
 0115                  * COPY FILTER COEFFICIENTS TO DATA MEMORY
 0116 0068
 0117 0068 D001           LALK   >300+FIL0A0      BLOCK B1
      0069 0307
 0118 006A 6001           SACL   YN
 0119 006B 3001           LAR    AR0,YN  * AR0 = DATA MEMORY PTR
 0120 006C CA0C           LACK   FILPA            ACC = PGM MEMORY PTR
 0121 006D C115           LARK   AR1,FILNC        AR1 = COUNTER
 0122 006E 58A9  LDDH     TBLR   *+,AR1           GET COEFFICIENT, ARP = 1
 0123 006F CC01           ADDK   1
 0124 0070 FB98           BANZ   LDDH,*-,AR0      BRANCH IF MORE, ARP = 0
      0071 006E
 0125 0072
 0126                  * ZERO THE FILTER STORAGE LOCATIONS
 0127 0072
 0128 0072 D001           LALK   >300+FIL0N       POINTER TO FIRST LOCATION
      0073 031C
 0129 0074 6001           SACL   YN
 0130 0075 CA00           ZAC                     ZERO TO STORE
 0131 0076 3001           LAR    AR0,YN
 0132 0077 C10C           LARK   AR1,FILNS        NUMBER OF STORAGE LOCATIONS
 0133 0078 60A9  ZLOOP    SACL   *+,0,AR1         ZERO A WORD
 0134 0079 FB98           BANZ   ZLOOP,*-,AR0     LOOP
      007A 0078
 0135 007B
 0136 007B 8201           IN     YN,DA            TOSS FIRST (OLD) SAMPLE, RESET BIO
 0137 007C E201           OUT    YN,DA
 0138 007D D001           LALK   >2000            A 1 IN Q13 FORMAT
      007E 2000
 0139 007F 6006           SACL   ONE
 0140 0080
 0141                  * HERE AT START OF 512-SAMPLE SUMMATION LOOP
 0142 0080
 0143      0080  CALCLP   EQU    $
 0144 0080 CA00           ZAC
 0145 0081 6002           SACL   XSUM             INIT SUMS TO 0
```


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

Non-Profit Org.
US Postage
PAID
Hartford, CT
Permit No. 2929

QEX (ISSN: 0886-8093) is published monthly by the American Radio Relay League, Newington, CT USA.

David Sumner, K1ZZ
Publisher

Jon Bloom, KE3Z
Editor

Lori Weinberg
Assistant Editor

Harold Price, NK6K
Zack Lau, KH6CP
Contributing Editors

Production Department

Mark J. Wilson, AA2Z
Publications Manager

Michelle Bloom, WB1ENT
Production Supervisor

Sue Fagan
Graphic Design Supervisor

Dianna Roy
Senior Production Assistant

Circulation Department

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

Offices

225 Main St, Newington, CT 06111 USA
Telephone: 203-666-1541
Telex: 650215-5052 MCI
FAX: 203-665-7531 (24 hour direct line)
Electronic Mail: MCIMAILID: 215-5052
Internet:qex@arrl.org

Subscription rate for 12 issues:

In the US by Third Class Mail: ARRL Member $12, nonmember $24;

US, Canada and Mexico by First Class Mail:
ARRLMember $25, nonmember $37;

Elsewhere by Airmail:
ARRLMember $48 nonmember $60.

QEX subscription orders, changes of address, and reports of missing or damaged copies may be marked: *QEX* Circulation.

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

About the Cover:

Digital signal processing applications are growing by leaps and bounds. Here's one close to the hearts of amateurs: measuring receiver performance. **136**

Features

Columns

June 1993 QEX Advertising Index

THE AMERICAN RADIO RELAY LEAGUE

The American Radio Relay League, Inc, is a noncommercial association of radio amateurs, organized for the promotion of interests in Amateur Radio communication and experimentation, for the establishment of networks to provide communications in the event of disasters or other emergencies, for the advancement of radio art and of the public welfare, for the representation of the radio amateur in legislative matters, and for the maintenance of fraternalism and a high standard of conduct.

ARRL is an incorporated association without capital stock chartered under the laws of the state of Connecticut, and is an exempt organization under Section 501(c)(3) of the Internal Revenue Code of 1986. Its affairs are governed by a Board of Directors, whose voting members are elected every two years by the general membership. The officers are elected or appointed by the Directors. The League is noncommercial, and no one who could gain financially from the shaping of its affairs is eligible for membership on its Board.

"Of, by, and for the radio amateur, "ARRL numbers within its ranks the vast majority of active amateurs in the nation and has a proud history of achievement as the standard-bearer in *amateur affairs.*

A bona fide interest in Amateur Radio is the only essential qualification of membership; an Amateur Radio license is not a prerequisite, although full voting membership is granted only to licensed amateurs in the US.

Membership inquiries and general correspondence should be addressed to the administrative headquarters at 225 Main Street, Newington, CT 06111 USA.

Telephone: 203-666-1541 Telex: 650215-5052 MCI.
MCIMAIL (electronic mail system) ID: 215-5052
FAX: 203-665-7531 (24-hour direct line)

Canadian membership inquiries and correspondence should be directed to CRRL Headquarters, Box 56, Arva, Ontario N0M 1CO, tel 519-660-1200.

Officers

President: GEORGE S. WILSON III, W4OYI
1649 Griffith Ave, Owensboro, KY 42301

Executive Vice President: DAVID SUMNER, K1ZZ

Purpose of *QEX*:

1) provide a medium for the exchange of ideas and information between Amateur Radio experimenters

2) document advanced technical work in the Amateur Radio field

3) support efforts to advance the state of the Amateur Radio art

All correspondence concerning *QEX* should be addressed to the American Radio Relay League, 225 Main Street, Newington, CT 06111 USA. Envelopes containing manuscripts and correspondence for publication in *QEX* should be marked: Editor, *QEX.*

Both theoretical and practical technical articles are welcomed. Manuscripts should be typed and doubled spaced. Please use the standard ARRL abbreviations found in recent editions of *The ARRL Handbook.* Photos should be glossy, black and white positive prints of good definition and contrast, and should be the same size or larger than the size that is to appear in *QEX.*

Empirically Speaking

Our New Look

From time to time it's useful to step back and take a critical view at the practices you've been using. As part of the ongoing review of *QEX* that we discussed in this space in the August 1992 issue, we looked at the typography and graphic design of *QEX*—and weren't happy with what we saw. So, we've changed our design.

The new layout of *QEX* is intended to make it more readable, allowing us to present information in a more organized fashion. Our new three-column page layout gives us more flexibility in combining text and graphics in a way that keeps one from intruding on the other. We've selected new fonts that we think will make the text easier to read, too.

We hope these changes will serve our readers better, and we encourage you to suggest any alterations to *QEX* that you think would help. In the end, though, the mechanics of the magazine are secondary—it's the *content* that is most important. We think we are headed in the right direction, providing useful and interesting information to the amateur technical community. Which is not to say we don't have plenty left to do to upgrade *QEX*. The most frequent complaint about *QEX* is that it is too thin. And we agree. Now that we have the mechanical issues dealt with, we are prepared to make *QEX* thicker. But there is one catch: We can't make it bigger unless we get quality material to publish.

That, of course, is where you come in. Whether we can bring you more pages of experimenters' goodies each month is directly dependent on whether readers are willing to do their part by contributing to the process, thereby helping to make *QEX* the effective *experimenter's exchange* we want it to be. Articles need not be long, and they need not contain complete projects with all the *i*'s dotted and the *t*'s crossed. What they do need to contain are ideas, preferably practical ones that can assist other experimenters.

If you are working on something original, please consider writing about it for *QEX*. Don't worry about making your English and drawings publication-quality, we'll take care of that. But we do need your publishable ideas. Whether *QEX* expands to bring you more of what you want each month depends on our getting them.

This Month in *QEX*

Wes Hayward, W7ZOI, presents a detailed discussion of a deceptively simple circuit in, "Variations on a Single-Tuned Circuit." While the LC tank itself isn't that obscure, proper coupling of the input and output signals isn't trivial. Wes provides a *MathCad* application and a complete explanation to make the process quite straightforward.

David Leeson, W6QHS, author of the ARRL book, *Yagi Antenna Design*, explains in "Joint Design for Yagi Booms" how you can keep your boom from twisting apart at the joints between sections.

Your editor contributes "Measuring SINAD With DSP," a TMS320C25 DSP application for equipment testing that can be easily ported to the DSP system of your choice.

In this month's installment of "Digital Communications," columnist Harold Price, NK6K, turns the keyboard over to Phil Karn, KA9Q, for a detailed description of Phil's FEC experiments that we mentioned here briefly in the April issue. Harold also follows up on some of his previous discussions of software and radio design issues.—*KE3Z*
email: jbloom@arrl.org (Internet)

Variations on a Single-Tuned Circuit

The single-tuned circuit is useful—and easy to design using MathCad, as W7ZOI shows us.

Wes Hayward, W7ZOI

The simplest LC bandpass consists of one resonator, the single-tuned circuit (STC). We use one when we're looking for simplicity, or when an impedance transformation is needed as part of the frequency selective circuit. Although simple, not all versions of the single-resonator filter are identical.

This note examines one particular form of the STC. This form evolves from the fundamental forms shown in Fig 1. The form in Fig 1A is a parallel-tuned circuit consisting of ideal (lossless) reactances, L and C, loaded with a parallel resistance. This resistance represents the losses in the circuit. The circuit of Fig 1B uses a series resistance to model the losses. The two forms are substantially identical at resonance.

If all of the losses in the STC are intrinsic to the resonator L and C, the resulting Q is an *unloaded* value. However, if they are partly the result of external loading, a *loaded* Q related to resistance-to-reactance ratios is the result. Fig 1C represents a variation with part of the loss related to a series resistance while the other part is attributed to parallel loading. Extending this concept further results in the filter circuit shown in Fig 2. The parallel loading is obtained by transforming a low resistance load of value R_L to appear as a higher resistance, R_p, that parallels the tuned circuit. The transformation is done by C_s and C_p, which transfer R_L to cause a resistance R_p to appear across the tuned circuit. This additional capacitance also adds to the resonator C in parallel.

A capacitor, C_e, in parallel with the R_L

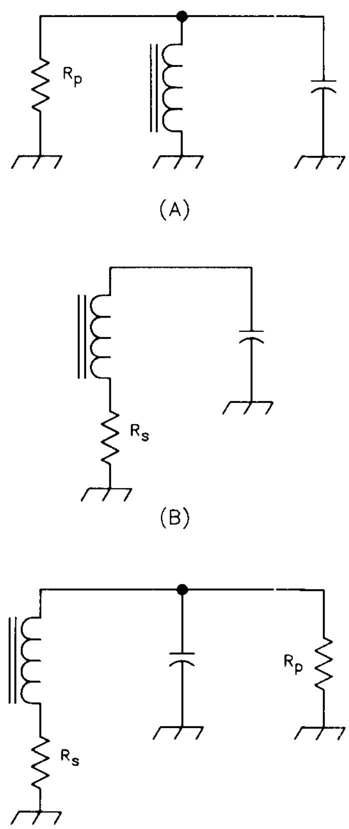


Fig 1

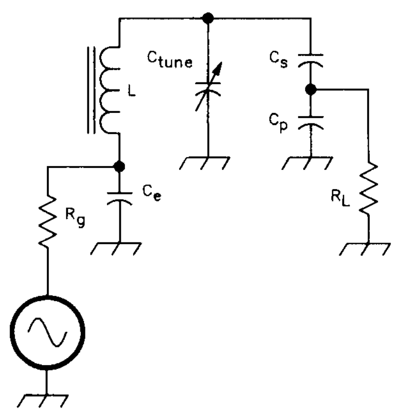


Fig 2

[1]Notes appear on page 5.

7700 SW Danielle Ave
Beaverton, OR 97005

Input variables:

$f := 7$ (MHz) $BW := 0.1$ (MHz) $R_0 := 50$ (ohms) $L_u := 2.5$ (μH) $Q_u := 200$

The equations:

$$Q_f := \frac{f}{BW} \qquad \omega := 2 \cdot \pi \cdot 10^6 \cdot f \qquad L := L_u \cdot 10^{-6} \qquad C_0 := \frac{1}{(L \cdot \omega^2)} \text{ (nodal capacitance)}$$

First we compute the parameters associated with the tapped-C end.

$$Q_e := \left(\frac{1}{Q_f} - \frac{1}{Q_u}\right)^{-1} \qquad R_p := 2 \cdot Q_e \cdot \omega \cdot L \quad \text{(External Q, parallel load at C-tap end.)}$$

$$C_{smin} := \frac{1}{\omega \cdot \sqrt{R_p \cdot R_0 - R_0^2}} \quad \text{(The minimum series C that will work.)}$$

$$C_{sminp} := C_{smin} \cdot 10^{12}$$

$C_{sminp} = 20.9$ (in pF)

Pick a value for Csp (must be > Csminp).

$$C_{sp} := 27$$

$$C_s := C_{sp} \cdot 10^{-12}$$

$$G := \frac{1}{R_0} \qquad C_p := \frac{\sqrt{R_p \cdot \omega^2 \cdot G \cdot C_s^2 - G^2}}{\omega} - C_s \quad \text{(Cp is the parallel capacitor at the tapped-C end.)}$$

$$C_{eq} := \frac{G^2 \cdot C_s + \omega^2 \cdot C_s \cdot C_p \cdot (C_s + C_p)}{G^2 + \omega^2 \cdot (C_s + C_p)^2}$$

This concludes the calculations related to the capacitor-tap end "loading."
Now we calculate the parameters for the other (shunt-C) end.

$$R_s := \frac{\omega \cdot L}{2 \cdot Q_e} \quad \text{(The required resistance for the specified Q.)}$$

The actual termination is transformed to Rs by the shunt capacitor:

$$C_e := \sqrt{\frac{R_0 - R_s}{R_s \cdot \omega^2 \cdot R_0^2}} \qquad C_x := \frac{C_e^2 \cdot \omega^2 \cdot R_0^2 + 1}{C_e \cdot \omega^2 \cdot R_0^2} \qquad C_{ep} := C_e \cdot 10^{12}$$

Cx is the equivalent capacitance that is in series with the small resistance, Rs.
Cep is commonly quite large.

Now determine the parallel capacitance to tune the resonator:

$$C_{tune} := \frac{C_{eq} \cdot C_0 + C_x \cdot C_0 - C_x \cdot C_{eq}}{C_x - C_0} \qquad C_{pp} := C_p \cdot 10^{12} \qquad C_{tp} := C_{tune} \cdot 10^{12}$$

Here MathCAD performs the actual calculations for the capacitances (in pF).
L is displayed in μH, R in ohms:

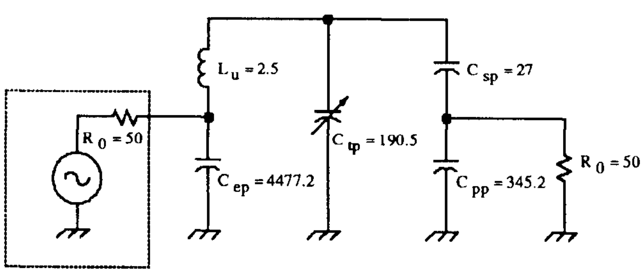


Fig 3—This *MathCad* application calculates the component values for the circuit of Fig 2. The generator and load resistances are assumed to be the same. The generator resistance is transformed by the parallel capacitance at the input, and the load resistance is transformed by the tapped capacitance at the output. The result is the needed parallel loading of the tank circuit to achieve the correct loaded *Q* for the specified bandwidth of the circuit. The calculated tuning capacitance takes the equivalent capacitance at the input and output into account.

resistance effects a similar impedance transformation. The shunt C causes a resistance R_s to appear in series with the inductor, L. The capacitor also adds reactance in series with L that will affect the resonator tunng.

A set of equations for this filter is shown in Fig 3. This is an output from *MathCad*, Version 3.1 from MathSoft (for Microsoft *Windows* on the IBM-PC and clones).[1] Note that numeric values for a specific design, in this case, a 7-MHz filter, appear at the top of the page. The equations that follow state the relationships needed to describe the filter. If you have *MathCad*, you can key the equations in as shown, and then edit the values shown in the top row to fit your own requirements. [You can also download the STC.MCD file from ARRL's telephone BBS at 203-666-0578.—Ed.] Calculated results will appear further in the page. You must supply the program with a value for the series capacitor, C_s. The listing then calculates the rest of the circuit values.

Of course, it is not necessary for you to have *MathCad* to use the equation set. It can be used with a scientific calculator for direct calculation. It can also be used as the basis for a program for a programmable calculator or a computer. Note the difference in the *MathCad* listing between the equal sign and the one preceded by a colon. The ":=" is a definition of a variable or relationship supplied by the user while the simpler "=" is a request for a numeric evaluation. The *MathCad* format is handy for documentation, for it tests an equation set for completeness. If data values or equations are left out of the description, the ending values will not be calculated.

The equation set can be extended to design similar, related filters. For example, the circuit of Fig 4 uses identical "series" terminations at each end. This filter has good attenuation in the stopband above the filter passband. Another useful variation is that shown in Fig 5 where parallel loading is used at each end. The exact equations for these filters are not included here, but it should not be difficult to extend the algebra of Fig 3 to these circuits. The response of the 7-MHz filter used as the example (in Fig 3) is shown in Fig 6. Also plotted in that figure is the response of the filter of Fig 4 using an identical bandwidth and inductor.

A word of warning about these simple series-tuned circuits. They are very useful tools in the experimenter's list of "standard circuits." But, these circuits

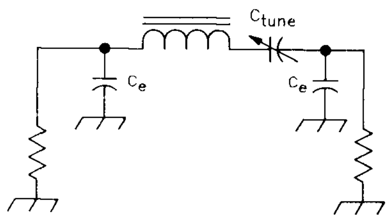


Fig 4

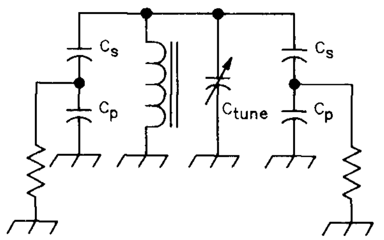


Fig 5

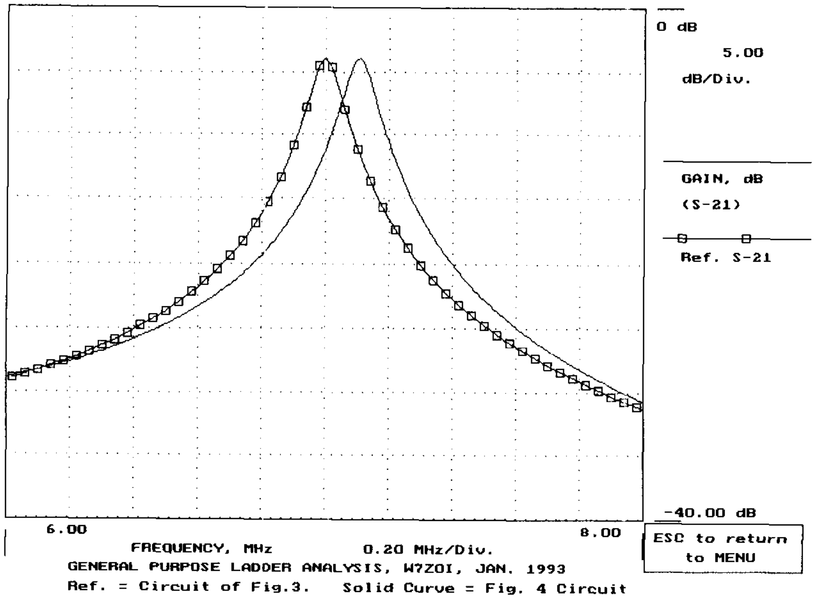


Fig 6

should be *designed*, not empirically "grown" on the bench. Many of the component values are not in line with intuition, making it easy to end up with filters with excessive insertion loss. The calculated designs can, however, be built with slightly different component values with little change in response.

An interesting variation replaces C_e in Fig 2 with an inductor having an identical reactance at the filter center frequency. This topology is especially useful for VHF filters. A recent article presented some double-tuned circuits with this form of loading.[2]

The filter shown in Fig 2 is useful over a wide frequency range, especially in low impedance environments. I've used the topology at audio and also in monolithic form at microwave frequencies. In both extremes, performance is limited only by the available unloaded inductor Q.

References

[1]Those interested in *MathCad* should write to the MathCad Upgrade Center, PO Box 120, Buffalo, NY 14207-0120.

[2]Hayward, "The Double-Tuned Circuit: An Experimenter's Tutorial," *QST*, December 1991.

Joint Design for Yagi Booms

Maybe your boom won't bend, but will it twist apart? Here's how to ensure it won't!

David B. Leeson, W6QHS

Boom Joint Considerations

In the construction of a Yagi boom, the question arises of the required strength of joints between sections of boom tubing. Making a joint that is as strong in *bending* as the adjacent tubing is relatively straightforward; the joint must have at least the same section modulus as that required for the tubing itself. The required section modulus, and hence the dimensions of the joint, can be calculated from wind- and ice-loading considerations.[1]

A previously unanswered question is how to be sure that the *torsional* strength of a boom joint is strong enough. Generally, boom joints fail by elongation of the holes containing the fasteners because the torsional (twisting) motion of the elements in the wind results in forces that exceed the bearing capability of the tubing material. Shear failures in the fasteners themselves are far less common, especially if steel fasteners are used. It is generally accepted that the shear force capability of the fasteners and the bearing force capability of the joint material is not augmented by any additional strength benefit from clamping force and friction.

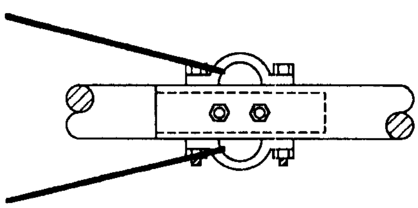


Fig 1—Bolted boom joint with guying eyes.

This failure mode, elongation and ultimate destruction of the holes in the thinnest boom section tubing, is perhaps the most common failure in 40- and 20-meter Yagis, given that adequate boom strength is achieved for bending considerations.

Even without a detailed calculation of the forces arising from resonant torsional motion of the elements, we can see that the polar moment of the elements ought to be minimized by design. This means keeping away from mass that is located far from the boom—toward the element tips. The tips of the elements ought to have the minimum weight (wall thickness and diameter) that will meet other requirements, and reinforcing should be limited to element sections near the center. Also, the torsional requirements for the boom will tend to increase from the boom ends, where only a single element is creating torsional force, to the center, which must resist the torsional forces of several elements in the typical case. Fortunately, this requirement conforms with the need for the boom to be stronger near the center from other considerations.

Maximum Torsion for a Tube

A boom joint criterion that makes sense is to require the joint torsional capability to be equal to that of the boom section. While I have observed a number of joint failures, I have not seen any torsional failures, buckling or otherwise, in boom tubing.

The formula for maximum torsion T_{max} in a hollow tube is given by:[2]

$$T_{max} = \frac{\pi f_s (R_1^4 - R_2^4)}{2R_1}$$

where f_s is the shear yield strength, and R_1 and R_2 are the outer and inner radius of the tube, respectively. For 6061-T6 and 6063-T832, f_s = 20,000 lb/in^2. The outer and inner radius of the tube are related to the wall thickness, t, by the expression:

[1]Notes appear on page 8.

15300 Soda Springs Road
Los Gatos, CA 95030

$R_1 - R_2 = t$

For $t << R_1$, $(R_1^4 - R_2^4) \approx 4tR_1^3$ so T_{max} can be written as:

$$T_{max} \approx 2\pi f_s t R_1^2$$

Maximum Torsion in a Fastened Joint

Now consider the maximum torsional strength T_j of a joint in material of thickness t, using n fasteners of diameter d. The maximum bearing force is the product of f_b, the bearing yield strength of the material, and $d \cdot t$, the bearing area of the hole. For 6061-T6 and 6063-T832, f_b=56,000 lb/in². The force acts at radius R_1, so the maximum torsional strength from bearing strength considerations is:

$$T_j = f_b n d t R_1$$

If we set $T_{max} = T_j$, we have

$$nd = \left(\frac{f_s}{f_b}\right) 2\pi R_1$$

This has the curious result that the total of all the hole diameters is related to the circumference of the tube by the ratio of the shear and bearing yield strengths. For aluminum, this reduces to:

$$n \approx \frac{2.24R_1}{d}$$

As an example, if R_1=1 inch and d=3/16 inch, then $n \approx 12$ fastener holes required. This can be six through-bolts or twelve rivets, spaced at least three fastener diameters center-to-center. For R_1=1 inch and d=¼ inch, $n \approx 8$, so we can use four through-bolts.

Number of Fasteners Required

The shear strength of the fasteners themselves should also be checked. Ignoring for the moment the reduction in strength with high ratios of hole diameter to wall thickness, the shear capability of n bolts or rivets of diameter d and shear yield f_r is $n f_r \pi (d/2)^2$, so the torsion capability at radius R_1 is:

$$T = \frac{n f_r R_1 \pi d^2}{4}$$

If we set this equal to T_{max}, we find that, for equal shear yield (aluminum rivets and tube),

$$n = \frac{8 f_s t R_1}{f_r d^2}$$

which, for $f_r = f_s$, t=0.058 inch, R_1=1 inch and d=3/16 inch, yields a requirement from fastener strength considerations of $n \approx 13$. The ratio of d/t=3.23, which results in less than 2% reduction in rivet shear strength. For d=¼ inch, $n \approx 8$.

If we use steel fasteners, the required n is reduced by the ratio of bolt shear yield to aluminum shear yield, which can be taken to be ≈3 for heat-treated steel and ≈2 for stainless steel. As yet, I have not found reliable data that is more precise.

For thicker and/or larger diameter tubing, it is clear that bolt or rivet sizes must be increased to preclude an inconveniently large number of fasteners. I use four ¼-inch bolts through 2-inch thinwall material, and four 5/16-inch bolts through 2½-inch material with wall thickness of up to ⅛ inch. A typical hole pattern is one- to two-inch spacing of two bolts, with the other two at right angles with one spaced midway between the two in the other plane. Spacing is often determined by the stainless steel eyes used at the top and bottom of the boom for guying attachment.

If the sections are riveted, the hole pattern for the rivets should *not* be such as to reduce the strength of the underlying material or introduce a likely fatigue failure. For example, a single circumferential row of holes should be avoided.

Table 1 summarizes the calculations for various cases. It can be seen that, with twelve rivets or eight bolt holes, the boom torsional strength is limited by the joint fastening strength.

Table 1

Tubing torsion:	*Riveted*	*Bolted*	*Bolted*	*Bolted*
OD, inch	2	2	2.5	2
t, inch	0.058	0.058	0.125	0.12
f_s lb/sq inch	20000	20000	20000	20000
f_b lb/sq inch	56000	56000	56000	56000
T_{max} in-lb	7288	7288	24544	15080
Bearing force:				
d, inch	0.1875	0.25	0.3125	0.25
T_j per fastener	609	812	2734	1680
n (bearing)	12	9	9	9
Fastener shear:				
f_r lb/sq inch	20000	40000	40000	40000
T_j per fastener	552	1963	3835	1963
n (shear)	13	4	6	8
n	13	9	9	9

Practical Applications

Even though these calculations suggest that, with an adequate number of fasteners the problems of elongation or shear failure are minimized, there are some practical ways to improve things further. In race car practice, a washer is typically welded onto the thin tab through which a bolt is to be passed; this keeps the tab from bending the bolt while increasing the bearing area to an adequate degree. This can be applied to the case of a thin boom section, either by welding on washers at the bolt holes or making a ring or larger diameter tubing which is then welded onto the piece to be joined. Similarly, reinforcement or spacers inside the tubing to permit full tightening of the bolts without deforming the tubing seems a good idea.

The joint holes do not create a fatigue hazard because they are located in either the very-low-stress outer end of a section or in a reinforced, overlapped inner end of a section. Swage joints should point "out"; that is, the swage should be on the outer end of the inner section.

It should be noted that an internal reinforcement will not provide any increase in effective wall thickness for torsion unless it is fastened to the outer tube by welding, epoxy, Loctite or additional fasteners.

A very satisfactory boom design uses

either 2-inch IPS (2.375-inch OD, 0.154-inch wall) or 2.5 × 0.125-inch extruded tubing for the center section, 2 × 0.12-inch drawn tubing with inner reinforcement of 0.083-inch wall (standard Hy-Gain part for 204BA inner boom sections, with swaged end), and 2 × 0.058-inch drawn tubing for the boom tips. This can be guyed at the outer end of the reinforced part of the 0.12-inch wall, or with lower ratings at the joint with the tip tubing. Shimming or a machined tubular spacer is required to fit the 2-inch tubing to the inner diameter of the center section, since extruded pipe or tubing does not provide tight ID tolerances. For this boom design, four 5⁄16-inch stainless bolts with washers and elastic stop nuts (Nylock) are used for the inner joint, and four 1⁄4-inch stainless bolts with washers and stop nuts are used for the outer joint. I use a comparable setup on the 48-foot booms on my 10- and 15-meter Yagis. For a 40-meter beam, the use of 0.065-inch wall thickness might be appropriate for the tip sections.

Notes

[1]D.B. Leeson, *Physical Design of Yagi Antennas*, ARRL, Newington, 1992, Chapter 6.

[2]Aluminum Company of America, *ALCOA Structural Handbook*, Pittsburgh, 1956, pages 94-96.

Correction

Zack Lau's "RF" column in the May *QEX* neglected to mention the types of PC board material needed for the designs presented. The 5X multiplier should be built on 1⁄16-inch G-10 or FR-4 epoxy board (ε_r=4.8). The doubler board uses 1⁄32-inch, 2.55 ε_r Teflon board.

Measuring SINAD Using DSP

Measuring receiver performance by SINAD is easy with a SINAD meter like this one done using digital signal processing.

Jon Bloom, KE3Z
Editor, *QEX*

One of the subjects that interests me is the application of DSP to equipment testing. In that vein, I've developed a simple DSP application to measure SINAD of a signal modulated at 1 kHz. The application presented here was developed on an experimental TMS320C25-based DSP plug-in card for the IBM PC. The card was developed by Tucson Amateur Packet Radio (TAPR) but never put into production. Although the card itself is not available, the TMS320C25 processor is common, as are other processors in the TMS320 family. So while the program presented here may not run directly on the DSP board you're using, it can be ported to any TMS320-series fixed-point processor relatively easily.[1] And, of course, the techniques presented are applicable to any processor.

SINAD measurement is useful for more than just seeing if a radio is performing up to specifications. It's also just about the best way to tune up an FM receiver. Once you've set up the SINAD measurement equipment, you just "tune for max" on the receiver adjustments. The best SINAD will occur when the receiver is on frequency, optimally tuned, and with the discriminator/detector correctly adjusted. (Of course, it's always possible in some radios that interacting adjustments are better made using some other technique, but that's rare.)

[1]Notes appear on page 13.

225 Main St.
Newington, CT 06111
email: jbloom@arrl.org (Internet)

SINAD Explained

SINAD stands for "signal plus noise plus distortion to noise plus distortion ratio." What a mouthful! Fortunately, it's easy to explain what that means. Simply, a SINAD measurement determines the signal-plus-noise-to-noise ratio of a received signal. In other words:

$$SINAD = \frac{signal + noise}{noise} \qquad (1)$$

In this equation, I've lumped noise and distortion together as noise. That is, anything that isn't signal is noise. SINAD measurement is typically easier to do than a direct measurement of the signal-to-noise ratio (SNR), so SNR is seldom used in receiver performance measurements. It can be easily calculated from the SINAD, however:

$$SINAD = \frac{signal + noise}{noise} = \frac{signal}{noise} + 1$$

$$\frac{signal}{noise} = SINAD - 1 \qquad (2)$$

Since SINAD is a ratio, it is most usefully stated in decibels (dB). The standard formula for dB is used:

$$dB = 10 \log\left(\frac{signal + noise}{noise}\right) \qquad (3)$$

This assumes the signal-plus-noise and noise *powers* are measured. If the voltages are measured instead, the coefficient would of course be 20 rather than 10. In fact, one benefit of using decibel notation is that the final result is the same number whether the original measurement was done by measuring voltage or by measuring power. You'll almost always see SINAD expressed in dB.

SINAD measurement is typically made using single-tone modulation because that makes it relatively easy to separate the signal from the noise, as we will see. Although SINAD measurement can be applied to almost any kind of receiver, my familiarity with it is from the measurement of FM receivers. Standard measurements of FM communication receivers call for SINAD measurements under varying amounts of signal level and modulation. The most important combination is 1-kHz, 60% modulation at a signal level that produces 12-dB SINAD. The signal amplitude that produces this result is called the *reference sensitivity*. In the case of amateur VHF/UHF FM voice systems, 60% modulation consists of ±3 kHz deviation, since 100% modulation requires ±5 kHz deviation.

Measuring SINAD

Typical SINAD meters work something like the one shown in block-diagram form in Fig 1. Our SINAD equation (1) requires two terms: the signal-plus-noise, which is just the input audio, and the noise alone, which is the input after the signal has been removed. The removal of the single-tone signal is done by use of a sharp notch filter tuned to the frequency of the signal. The result at the output of the filter is just the noise audio. Of course, the notch filter removes a little bit of the noise around 1 kHz along with the signal, but if the notch is sharp enough this won't substantially affect the SINAD measurement.

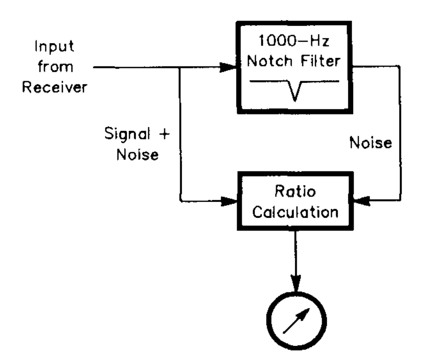


Fig 1—Block diagram of a SINAD meter.

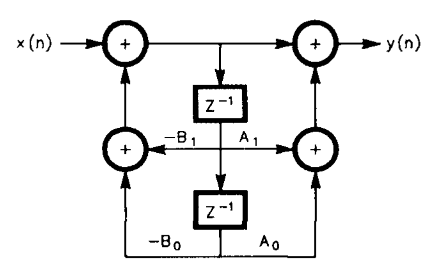


Fig 2—A second-order IIR filter. Each branch (line) of the diagram represents a multiplication. If unlabeled, the branch is a multiplication by 1 (which requires no processing, of course). A_0, A_1, B_0 and B_1 represent the coefficients of the filter. Each block labeled Z^{-1} represents a delay. Thus when, for example, sample x(3) is present at the input, the sample at the output of the upper delay element was derived from sample x(2) and the output of the lower delay element was derived from sample x(1). Practical filters often need a gain correction applied to the output, y(n), which is a multiplication of each output sample by a constant.

This same approach is used in distortion meters. In fact, a SINAD meter and a distortion meter are essentially the same thing. The differences are that a SINAD meter is calibrated in dB, whereas a distortion meter is calibrated in percent distortion (many meters are calibrated in both, of course); the SINAD meter often operates at one or two fixed frequencies, whereas the distortion meter usually has a tuneable notch so it can work throughout the audio range; and the SINAD meter may not be able to distinguish very small amounts of noise. (No one is interested in 60 dB SINAD readings.) These differences aside, a SINAD meter is a distortion meter is a SINAD meter.

Doing SINAD with DSP

With that background, we are ready to discuss how SINAD is implemented using DSP. In the DSP system, the input audio consists of a set of *samples*. Each of these samples represents the input voltage at the sampling time, and the samples are taken at a regular rate, or frequency. Note that the sampling frequency must be more than twice the highest frequency present in the input audio. More detail on sampling and digitized signals is available in the literature.[2,3]

There are three basic things the SINAD program will need to do: measure the signal-plus-noise power, filter the signal from the noise, and measure the power of the noise alone. The average energy (or power) of a set of voltage samples is easily computed using the formula:

$$E = \frac{1}{N} \sum_{N-0}^{N-1} [x(n)]^2 \tag{4}$$

where N is the total number of samples, and $x(n)$ is sample number n. If this formula intimidates you, it shouldn't. This notation simply describes the summation of all of the values of $[x(n)]^2$, where n is from 0 to $N-1$. If this sounds to you like something custom made for a simple program loop, you're right! The SINAD program uses exactly that technique to compute signal power. You can also get the average voltage by taking the square root of E, although this program doesn't use that. (Of course, to know the actual voltage or power you would have to know the resistance the voltage appears across. For our purposes, this resistance is unimportant since we are interested in the *ratio* of the powers.)

That takes care of the measurement problem. Now we have to do the notch filter. The technique used here is known as an *infinite impulse response*

```
                                                          PC-DSP v1.1
 System  Data  Filter  sPectrum  Graphics  Macro  Quit

 FILTER / IIR-FILTERS / DESIGN

 Filter type    :  r             Approximation :  i

 Spec. type     :  2

 Passband tolerance :  1d

 Stopband tolerance :  40d

 Edge frequencies:

 F1 =  970/8000    , F2 =  990/8000    , F3 =  1010/8000   , F4 =  1030/8000

 Printer report ? :  N    Variable name for filter data :  NOTCHER

 Variable = NOTCHER , Type = C , Size =   19

              F2: Plot                        <ESC>: Menu
```

Fig 3—The *PC-DSP* display specifying the notch filter design.

(IIR) filter. A second-order DSP IIR filter can be implemented using the flow diagram of Fig 2. The input samples are each processed individually through the system, with the output samples being a filtered version of the input. Higher-order filters can be obtained by cascading second-order sections—the output of a second-order section being applied to the input of the next section. This kind of filter is the DSP equivalent of a standard analog filter. Indeed, the most common design technique used for IIR filters is to design an analog filter with the desired response and then *transform* it to a digital filter. Fortunately, we don't have to go through the mechanics of doing that. Instead, we use an inexpensive program that designs the filter for us: *PC-DSP*.[4]

Figs 3, 4 and 5 show the design of the IIR notch filter using *PC-DSP*. Fig 3 shows the design specification for a band-reject (notch) filter designed using a Chebyshev type-II approximation, with 1 dB of pass band ripple and a 40-dB stop band attenuation. Using a sampling frequency of 8 kHz, the specification calls for a stop band from 990 to 1010 Hz and pass bands of 0-970 Hz and 1030-4000 Hz. (The regions between the pass bands and the stop band are the transition bands.) Fig 4 shows the filter response of this design, as computed by *PC-DSP*. Fig 5 shows the filter coefficients, ready to be plugged into our program. Note that this is an eighth-order filter requiring four cascaded second-order sections.

Fixed-Point Math Implementation

One of the fundamental decisions to make when deciding what DSP processor to use is whether to use a fixed-point processor or a floating-point processor. Since almost any useful DSP calculation will involve real (fractional) numbers, it is considerably easier to program a floating-point processor, since they can handle large and small real numbers with ease. Fixed-point processors are restricted to integer math, which means that useful DSP programming will require some "tricks" to handle real numbers. But fixed-point processors are considerably less expensive than floating-point processors (the TMS320C25 chip sells for less than $13 as of this writing), and they tend to be faster for equivalent cost/complexity. So there are advantages to using fixed-point processors. In the case of the TMS320-series fixed point DSPs, there are some built-in aids to performing fixed-point calculations on real numbers, principally the ability to shift values while moving them between registers and memory.

The key to computing using real numbers in a fixed-point processor is to select a bit position for the radix point. (The radix point is like the decimal point, except this being a binary number system, we can hardly call it the "decimal" point!) This is perhaps best explained with an example. Say we have a 16-bit binary number and we select the radix point to be after the third bit from the left. The numbers in this system can then be written:

NNN.NNNNNNNNNNNNN

where each N is a binary digit (0 or 1). The digit immediately to the left of the radix point is the 2^0 digit. The next digit to the left is the 2^1 digit, and so

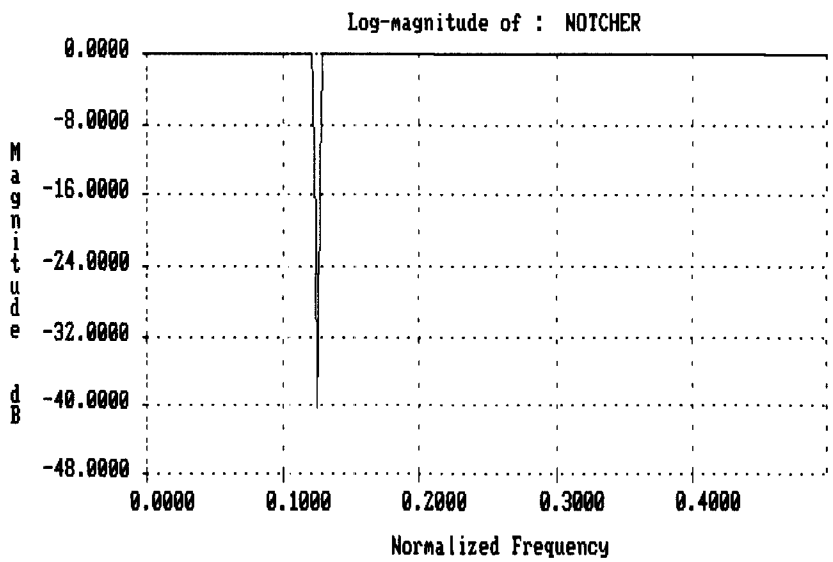


Fig 4—Notch filter response generated by *PC-DSP*. Note that the frequency axis is normalized. With an 8-kHz sampling rate, the graph displays frequencies from 0 to 4 kHz.

```
                                                                    PC-DSP v1.1
 System  Data  Filter  sPectrum  Graphics  Macro  Quit

 FILTER / IIR-FILTERS / ANALYZE

 Filter description variable :  NOTCHER      ---> Type = C , Size =  19

 Analysis type :  c

 Filter order  :  8            Gain factor :  9.63697E-01

     A2          A1          A0          B2          B1          B0

   1.0000     -1.4251      1.0000      1.0000     -1.4051      0.9743
   1.0000     -1.4190      1.0000      1.0000     -1.4280      0.9894
   1.0000     -1.4102      1.0000      1.0000     -1.3875      0.9740
   1.0000     -1.4039      1.0000      1.0000     -1.3854      0.9891

 F1: Help               F3: Print      <ESC>: Menu
```

Fig 5—*PC-DSP* analysis of the coefficients of the cascaded second-order sections of the notch filter. Each line of the table represents one filter section. Note that all of the A_2 and B_2 coefficients are 1, requiring no explicit multiplication.

on. The first bit to the right of the radix point is the 2^{-1} digit, the next one to the right is the 2^{-2} digit and so on. If all of the bits are 1, the result is:

$$2^2+2^1+2^0+2^{-1}+...+2^{-13}$$

Which in decimal is about 7.999878. So numbers arranged this way can represent values from 0 to 7.999878 (just less than 8) in steps of 2^{-13} ($\approx$ 0.000122). In their literature Texas Instruments refers to this form of numbering as "Q13" format. You can, of course, select other locations for the radix point. In the TMS320 processors, numbers can be treated as signed or unsigned. The SINAD program treats them as signed numbers, meaning that a 16-bit value has 1 sign bit and 15 binary digits. (Negative numbers are stored as two's-complement values.) Q13 format then has two magnitude bits to the left of the radix and can represent numbers in the approximate range of –4 to +4.

One of the crucial questions to consider when selecting the radix point is the maximum value that will have to be processed using that numbering format. Within the sections of an IIR filter, intermediate values may get larger than you expect, even if the input and output values of the filter should be well within the range of usable numbers. While it is possible to determine the maximum possible intermediate value analytically, it probably is easier for most experimenters to just sweep the filter with all possible input frequencies, looking for oddities in the output signal.

Fixed-Point Arithmetic

Addition and subtraction of fixed-point numbers is quite straightforward and differs not at all from the operations performed when the numbers are considered to be integer values. In other words, if you add (or subtract) two Q13 numbers using an ADD instruction, the result has the same Q13 format.

Multiplication is more complicated. Consider two 16-bit integer (not Q13) numbers. When you multiply these numbers, you get a 64-bit result. The TMS320 multiply hardware thus takes two 16-bit values, multiplies them, and places the result in a 32-bit register. What happens when the two values are Q13 numbers? In essence, the radix point is "multiplied" along with the digits. The easiest way to locate the resulting radix location is to multiply 1 times 1 (in Q13 format):

$$\begin{array}{r} 001.0000000000000_2 \\ \times\, 001.0000000000000_2 \\ \hline 000001.00000000000000000000000000_2 \end{array}$$

The TMS320 lets you store either the most-significant or the least-significant 16 bits of the product. But if we want the result to be in Q13 format (and we do), the bits we are interested in are the 4th through 19th bits. So we need to shift the result left three times, *then* store the upper 16 bits. The designers of the TMS320C25 realized we would need to do that, so the design allows us to do the shift and store in a single instruction with no lost time. All of the multiplications in the SINAD program work this way.

One of the most common DSP operations is the "multiply-accumulate." Fig 2 shows why: Each multiplication is typically followed by an addition. Since many DSP algorithms consist of repetitive multiply-and-add operations, DSPs are optimized to perform this action. Indeed, most DSPs (TMS320s included) perform this operation in a single instruction. Often, the data that is being multiplied and added must then be shifted to the next storage location. For this reason, the TMS320C25 has an instruction that multiplies, adds and shifts the contents of a memory location to the next location in memory, all in one instruction. While this instruction takes longer than, say, a simple register load, it takes less time than several instructions would need to perform the same operations. And when you have to do a lot of these operations in a limited period of time, this savings adds up!

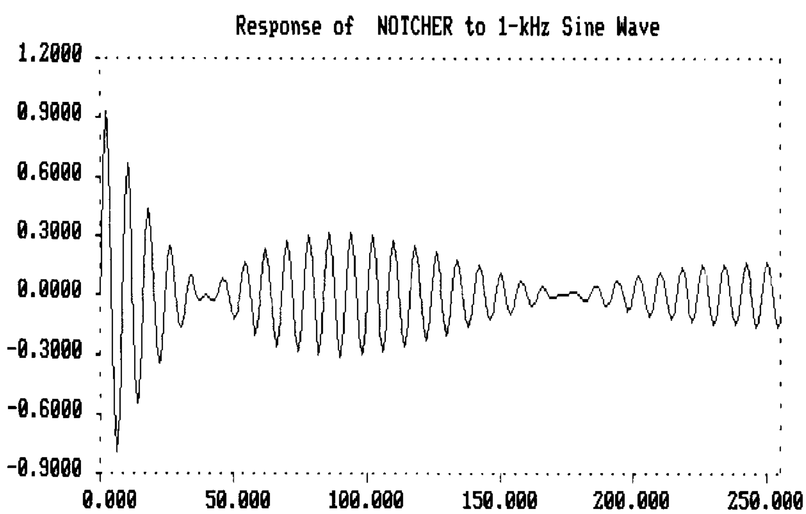


Fig 6—Response of the notch filter to a 1-kHz sine wave just after turn-on. Only 256 samples are shown, but clearly many samples will be needed before the response is damped out.

The SINAD Program

With that background we can discuss the DSP SINAD program in Listing 1. The program is organized as two nested loops. The inner loop (which starts at DAWAIT, line 156 of Listing 1) executes 512 times. It first waits for a sample to be ready from the A/D converter, which presents a sample every 1/8000 second. As each successive sample is read from the A/D converter the program: uses the sample to compute the signal-plus-noise energy; applies the sample to the "input" of the notch filter; and uses the output of the notch filter to compute the energy of the noise. After 512 samples have been processed in this manner, the outer loop computes the SINAD ratio and sends it to the PC for display. (The conversion of the SINAD to dB is done in the PC program, not by the DSP.)

Why 512 samples? Aside from being a convenient number (a power of 2), we must use a sufficient number of samples to avoid giving too much weight to the initial and final segments of the input waveform. The implicit assumption when measuring the average power of a signal is that you will average across a number of full cycles. If you average across one cycle or two cycles, you'll get the same average power (or voltage). But if you average across, say, 1⅛ cycles, you'll get a different result. Since we can't be sure we're measuring across an exact number of cycles of the input noise, we have to measure over enough cycles that a fractional cycle contributes almost nothing to the average. This requirement sets the minimum number of samples we can use. Again, you can determine this analytically, but the number 512 was determined experimentally since that's easier!

The SINAD calculation that begins on line 200 of Listing 1 requires a divide operation. The TMS320C25 has no divide instruction, however. What it does have is a conditional subtract instruction that, executed repetitively, performs a division. More details are available in the TI literature.[5]

Why a DSP?

You may wonder: Why is a specialized processor needed? Couldn't we just take 512 samples, process those, then go back and get 512 more samples to process? A PC could do that without having to keep up with the incoming data. Unfortunately, the use of an IIR filter precludes this approach. There cannot be any "lost" samples. This is true not only within the 512 samples used for computation, but from one 512-sample block to the next as well. If the sample that begins a 512-sample block is not the sample that immediately succeeds the last sample of the preceding block, the filter may begin not to work. Fig 6 shows the output of the IIR filter when a 1-kHz sine wave is applied to the input with the filter initialized (all storage locations set to zero). Notice that this is not the 40-dB-down signal you might expect. This is due to the slowly decaying impulse response of the filter; it takes a while for the filter to respond to the abrupt change. Eventually (within a few thousand samples), things will settle down and you'll get what you expect. But if any other abrupt change—such as lost samples—occurs during processing, this kind of response will reoccur.

The SINAD DSP program presented here can easily keep up with the input signal. Indeed, the processor spends most of its time waiting for the next sample. But if it is impossible to ensure continuous processing of the input signal, one strategy would be to collect enough samples that could be run through the filter to allow the impulse response to die out, performing the SINAD calculation on only the last 512 samples. It would take a lot of samples, though. Or you might use a finite impulse response (FIR) filter. This kind of filter has a relatively short impulse response so you wouldn't have to throw away thousands of samples, just a few hundred. But an FIR notch filter as sharp and deep as the IIR filter used here requires a lot more calculation by the computer, so much so that the resulting program would probably be very slow.

Quantization Effects

The system I used quantizes the input voltage to an 8-bit number. If the maximum input level (without overloading) were 1 volt, this would mean that the weight of each bit is 1/256 volts. (Actually, the input is treated as a signed value, but this discussion is simpler if we treat the input as 0 to 1 volts.) Obviously, a signal with an amplitude of less than 1/256 volts peak wouldn't even be noticed by the A/D converter. This effect sets the practical limit to our SINAD measurement. Measuring a 30 or 40 dB SINAD is quite doable with 8 bits. Measuring substantially higher SINADs would require more bits.

Alternate Techniques

The initial project on which this article was based included a separate implementation of a SINAD meter using fast Fourier transforms (FFT). The FFT can be used to transform a set of samples into a list of frequency "bins," showing how much of the signal is at each frequency. (See "A Receiver Spectral Display," by Bill de Carle, VE2IQ, January 1992 *QST*.) If you sum the magnitudes of each of the bins, you'll get the same total as we computed by summing the squares of the samples. (This equality is a result of *Parseval's theorem*.) Subtracting from this sum the magnitude of the 1-kHz bin gives the noise power. Once the signal-plus-noise and noise powers are known we are in business.

For some reason, many DSP tyros prefer applying an FFT to almost *any* DSP problem. It's instructive to do that for the SINAD meter: the FFT-based approach provided the same end result but with vastly increased computational complexity. The program is big, too. In fact, it was too big to fit into the available memory (4 kbytes) if written in the most speed-efficient form, so I had to compromise on speed somewhat. Also, the FFT-based program was much more critical with respect to the frequency of the 1-kHz signal. A small error in frequency resulted in a severe degradation of the measured SINAD due to spectral leakage. While this could have been addressed by applying a windowing function to the data, this would also have necessitated subtracting several bins from the sum, as the 1-kHz signal would have "spread" to adjacent frequency bins.

An intriguing hybrid technique, as yet untried, would use Goertzel's algorithm.[6] This algorithm is configured much like a second-order IIR filter. The output, however, consists of one bin of the FFT. That is, pass N samples through the algorithm and the result is the same as that of a given bin of the FFT. This is a good technique for calculating a small number of bins of the FFT at relatively low computational cost. The suggestion is to compute the signal-plus-noise energy using (1) and to measure the signal energy using Goertzel's algorithm. The difference would be the noise energy. Note, though, that to overcome the spectral leakage problem of the FFT, we would have to window the data and compute several bins using Goertzel's algorithm. Whether the end result would be simpler than the notch-filter-based SINAD meter is open to question.

Notes

[1]Texas Instruments recently announced "DSP Starter Kits" using the TMS320C26 (a close relative of the TMS320C25) and the TMS320C50, a more powerful DSP. The starter kits list for $99 and include a complete DSP system (processor, D/A and A/D converters and RS-232 serial port) on a printed circuit board, with development software for the IBM-PC/AT included. Contact a TI distributor, or call TI at (800) 336-5236 for information.

[2]ARRL, *The ARRL Handbook for Radio Amateurs* (Newington, CT: 1993), 8-20.

[3]D. J. DeFatta, J. G. Lucas et al, *Digital Signal Processing: A System Design Approach* (New York: Wiley, 1988).

[4]O. Alkin, *PC-DSP* (Englewood Cliffs, NJ: Prentice Hall, 1990).

[5]Texas Instruments, *TMS320C2x User's Guide* (Texas Instruments, 1990).

[6]J. Proakis and D. Manolakis, *Digital Signal Processing* (New York, Macmillan, 1988), 724.

```
SINDSP      320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                   PAGE 0001

                        IDT     'SINDSP'
     0000
                * SINAD SYSTEM FOR TAPR DSP-1 BETA BOARD
                *
                * F = 8 KHZ
                *
                * MEASURES AND COMPUTES THE SIGNAL-PLUS-NOISE-TO-NOISE RATIO OF
                * A 1 KHZ SIGNAL.  ALL NUMBERS IN Q13 FORMAT.
                *
                * THE INPUT SIGNAL IS PASSED THROUGH A NARROW BAND-STOP FILTER
                * CENTERED AT 1 KHZ.  THIS IS AN IIR CHEBYSHEV-II FILTER WITH THE
                * FOLLOWING DESIGN PARAMETERS:
                *
                *    PASSBAND RIPPLE    40 DB
                *    STOPBAND RIPPLE     1 DB
                *    FS1-H               0.12125 (970/8000)
                *    FP-L                0.12375 (990/8000)
                *    FP-H                0.12625 (1010/8000)
                *    FS2-L               0.12875 (1030/8000)
                *    GAIN                0 DB
                *
                * THE FILTER SOURCE IS IN THE FILE 'SINFILT.ASM'
                *
                * EACH SAMPLE OF THE INPUT SIGNAL IS SQUARED AND SUMMED OVER 512 SAMPLES.
                * THE SAME THING IS DONE TO EACH OF THE OUTPUT SAMPLES FROM THE FILTER.
                * THE RATIO OF OUTPUT/INPUT IS THEN COMPUTED, GIVING THE POWER RATIO OF
                * THE NOISE TO THE SIGNAL+NOISE, AS A 16-BIT NUMBER < 1.  THIS RESULT
                * IS OUTPUT TO THE HANDSHAKE PORT.  THE SUMS ARE THEN ZEROED AND THE
                * PROCESS IS REPEATED INDEFINITELY.
                *
                * USES BLOCK B1 FOR IIR COEFFICIENTS AND DATA
                *
                * WRITTEN BY J. BLOOM, KE3Z
                * 09 DEC 1992
     0000
          03E8  SPLDIV    EQU     1000     SAMPLE RATE DIVISOR FOR 8 KHZ RATE
     0000
                *  I/O PORTS (SPECIFIC TO TAPR BETA DSP-1 BOARD)
     0000
          0000  HSHAKE    EQU     0        HANDSHAKE PORT WITH PC
          0002  DA        EQU     2        A/D AND D/A
          0004  P8254     EQU     4        TIMER CHIP
          0005  RADIO     EQU     5        RADIO I/F PORT
     0000
                * MEMORY PAGE CONSTANTS
     0000
          0006  B1        EQU     >300/128
     0000
                * BLOCK B1 DATA MEMORY
     0000
                          DORG    0
     0000
     0000 0000  XN        DATA    0                 INPUT SAMPLE
     0001 0000  YN        DATA    0                 FILTER OUTPUT
     0002 0000  XSUM      DATA    0                 SUM OF INPUT SAMPLES SQUARED (32 BIT)
     0003 0000            DATA    0
     0004 0000  YSUM      DATA    0                 SUM OF FILTERED SAMPLES SQUARED (32 BIT)
     0005 0000            DATA    0
```

```
SINDSP      320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                   PAGE 0002

     0006 0000  ONE       DATA    0                 THE NUMBER 1 (Q13 FORMAT)
          0007  FILDA     EQU     $                 FILTER STUFF GOES HERE
     0007
                * PROGRAM MEMORY
     0007
     0000                 AORG    0
     0000
     0000 FF80            B       START
     0001 005B
     0002
                * TIMER SETUP DATA (8254)--TIMER 0 IS SAMPLE CLOCK, TIMER 1
                * ADJUSTS SAMPLE CLOCK PHASE (NOT USED HERE), TIMER 2 SETS
                * ANTI-ALIASING AND RECONSTRUCTION LPF FREQUENCIES.
     0002
     0002 03B6  TSETUP    DATA    >3B6              TIMER 2: (500-KHZ SQUARE WAVE)
     0003 0210            DATA    >210                       (LPF Fc = 10 KHZ)
     0004 0200            DATA    >200
     0005 0336            DATA    >336              TIMER 0: (8E6/SPLDIV) SQUARE WAVE
     0006 00E8            DATA    SPLDIV-((SPLDIV/256)*256)
     0007 0003            DATA    SPLDIV/256
     0008 0376            DATA    >376              TIMER 1: 10-KHZ SQUARE WAVE
     0009 0120            DATA    >120
     000A 0103            DATA    >103
          0009  NTSET     EQU     $-TSETUP
     000B 0040            DATA    >40               ENABLE A/D BIO ASSERTION
     000C
          000C  FILPA     EQU     $
     000C
                * IIR FILTER SOURCE IS INSERTED HERE
     000C
                          COPY    SINFILT.ASM
     0007                 DORG    FILDA
     0007
                * Section 0 coefficients
     0007 0000  FIL0A0    DATA    0
     0008 0000  FIL0A1    DATA    0
     0009 0000  FIL0A2    DATA    0
     000A 0000  FIL0B1    DATA    0
     000B 0000  FIL0B2    DATA    0
                * Section 1 coefficients
     000C 0000  FIL1A0    DATA    0
     000D 0000  FIL1A1    DATA    0
     000E 0000  FIL1A2    DATA    0
     000F 0000  FIL1B1    DATA    0
     0010 0000  FIL1B2    DATA    0
                * Section 2 coefficients
     0011 0000  FIL2A0    DATA    0
     0012 0000  FIL2A1    DATA    0
     0013 0000  FIL2A2    DATA    0
     0014 0000  FIL2B1    DATA    0
     0015 0000  FIL2B2    DATA    0
                * Section 3 coefficients
     0016 0000  FIL3A0    DATA    0
     0017 0000  FIL3A1    DATA    0
     0018 0000  FIL3A2    DATA    0
     0019 0000  FIL3B1    DATA    0
     001A 0000  FIL3B2    DATA    0
```

```
SINDSP      320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                   PAGE 0003

     001B 0000  FIL3NN    DATA    0
          0015  FILNC     EQU     21
     001C
                * Section 0 storage
     001C 0000  FIL0N     DATA    0
     001D 0000  FIL0N1    DATA    0
     001E 0000  FIL0N2    DATA    0
                * Section 1 storage
     001F 0000  FIL1N     DATA    0
     0020 0000  FIL1N1    DATA    0
     0021 0000  FIL1N2    DATA    0
                * Section 2 storage
     0022 0000  FIL2N     DATA    0
     0023 0000  FIL2N1    DATA    0
     0024 0000  FIL2N2    DATA    0
                * Section 3 storage
     0025 0000  FIL3N     DATA    0
     0026 0000  FIL3N1    DATA    0
     0027 0000  FIL3N2    DATA    0
          000C  FILNS     EQU     12
     0028
     000C                 AORG    FILPA
     000C
                * Section 0 coefficients
     000C 2000            DATA    >2000          1.0
     000D D266            DATA    >D266          -1.4251
     000E 2000            DATA    >2000          1.0
     000F 2DB1            DATA    >2DB1          1.4280
     0010 E057            DATA    >E057          -0.9894
                * Section 1 coefficients
     0011 2000            DATA    >2000          1.0
     0012 D298            DATA    >D298          -1.4190
     0013 2000            DATA    >2000          1.0
     0014 2CF6            DATA    >2CF6          1.4051
     0015 E0D3            DATA    >E0D3          -0.9743
                * Section 2 coefficients
     0016 2000            DATA    >2000          1.0
     0017 D2E0            DATA    >D2E0          -1.4102
     0018 2000            DATA    >2000          1.0
     0019 2C54            DATA    >2C54          1.3854
     001A E05A            DATA    >E05A          -0.9891
                * Section 3 coefficients
     001B 2000            DATA    >2000          1.0
     001C D313            DATA    >D313          -1.4039
     001D 2000            DATA    >2000          1.0
     001E 2C66            DATA    >2C66          1.3875
     001F E0D5            DATA    >E0D5          -0.9740
     0020 1ED6            DATA    >1ED6          0.9636 (GAIN)
     0021
          0021  FIL       EQU     $
                * Start with input sample in accum
     0021 3C1D            LT FIL0N1              Section 0
     0022 380A            MPY FIL0B1
     0023 3D1E            LTA FIL0N2
     0024 380B            MPY FIL0B2
     0025 CE15            APAC
     0026 6B1C            SACH FIL0N,3
```

```
SINDSP      320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                   PAGE 0004

     0027 CA00            ZAC
     0028 3809            MPY FIL0A2
     0029 3F1D            LTD FIL0N1
     002A 3808            MPY FIL0A1
     002B 3F1C            LTD FIL0N
     002C 3807            MPY FIL0A0
     002D CE15            APAC
     002E 3C20            LT FIL1N1              Section 1
     002F 380F            MPY FIL1B1
     0030 3D21            LTA FIL1N2
     0031 3810            MPY FIL1B2
     0032 CE15            APAC
     0033 6B1F            SACH FIL1N,3
     0034 CA00            ZAC
     0035 380E            MPY FIL1A2
     0036 3F20            LTD FIL1N1
     0037 380D            MPY FIL1A1
     0038 3F1F            LTD FIL1N
     0039 380C            MPY FIL1A0
     003A CE15            APAC
     003B 3C23            LT FIL2N1              Section 2
     003C 3814            MPY FIL2B1
     003D 3D24            LTA FIL2N2
     003E 3815            MPY FIL2B2
     003F CE15            APAC
     0040 6B22            SACH FIL2N,3
     0041 CA00            ZAC
     0042 3813            MPY FIL2A2
     0043 3F23            LTD FIL2N1
     0044 3812            MPY FIL2A1
     0045 3F22            LTD FIL2N
     0046 3811            MPY FIL2A0
     0047 CE15            APAC
     0048 3C26            LT FIL3N1              Section 3
     0049 3819            MPY FIL3B1
     004A 3D27            LTA FIL3N2
     004B 381A            MPY FIL3B2
     004C CE15            APAC
     004D 6B25            SACH FIL3N,3
     004E CA00            ZAC
     004F 3818            MPY FIL3A2
     0050 3F26            LTD FIL3N1
     0051 3817            MPY FIL3A1
     0052 3F25            LTD FIL3N
     0053 3816            MPY FIL3A0
     0054 CE15            APAC
     0055 6B01            SACH YN,3
     0056 3C1B            LT FIL3NN              Gain constant
     0057 3801            MPY YN
     0058 CE14            PAC
                * Filter output is in accum
     0059
                * END OF FILTER ROUTINE
     0059 6B01            SACH    YN,3           YN = FILTERED SAMPLE
     005A CE26            RET
     005B
                * PROGRAM STARTS HERE
```

```
SINDSP     320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                      PAGE 0005

0095 005B
0096      005B START    EQU    $
0097 005B C806          LDPK   B1              ALL DATA IN PAGE B1
0098 005C
0099               * INITIALIZE THE TIMERS (SAMPLE TIMER, PHASE TIMER AND
0100               * PROGRAMMABLE LPF CLOCK).
0101 005C
0102 005C CA02          LACK   TSETUP          DATA TABLE ADDRESS (PMA)
0103 005D C009          LARK   0,NTSET         # OF TIMER SETUP WORDS TO WRITE
0104 005E 5588          LARP   0
0105 005F 5801 TIMSET   TBLR   YN              GET NEXT TIMER SETUP WORD
0106 0060 E401          OUT    YN,P8254        SEND TO TIMER
0107 0061 CC01          ADDK   1               NEXT TABLE ENTRY
0108 0062 7F01          SBRK   1               ALL "NTSET" WORDS SENT?
0109 0063 FB80          BANZ   TIMSET,*        LOOP BACK IF MORE
     0064 005F
0110 0065
0111 0065 5801          TBLR   YN              GET RADIO PORT PROGRAM DATA
0112 0066 CC01          ADDK   1
0113 0067 E501          OUT    YN,RADIO
0114 0068
0115               * COPY FILTER COEFFICIENTS TO DATA MEMORY
0116 0068
0117 0068 D001          LALK   >300+FIL0A0     BLOCK B1
     0069 0307
0118 006A 6001          SACL   YN
0119 006B 3001          LAR    AR0,YN  * AR0 = DATA MEMORY PTR
0120 006C CA0C          LACK   FILPA           ACC = PGM MEMORY PTR
0121 006D C115          LARK   AR1,FILNC       AR1 = COUNTER
0122 006E 58A9 LDDH     TBLR   *+,AR1          GET COEFFICIENT, ARP = 1
0123 006F CC01          ADDK   1
0124 0070 FB98          BANZ   LDDH,*-,AR0     BRANCH IF MORE, ARP = 0
     0071 006E
0125 0072
0126               * ZERO THE FILTER STORAGE LOCATIONS
0127 0072
0128 0072 D001          LALK   >300+FIL0N      POINTER TO FIRST LOCATION
     0073 031C
0129 0074 6001          SACL   YN
0130 0075 CA00          ZAC                    ZERO TO STORE
0131 0076 3001          LAR    AR0,YN
0132 0077 C10C          LARK   AR1,FILNS       NUMBER OF STORAGE LOCATIONS
0133 0078 60A9 ZLOOP    SACL   *+,0,AR1        ZERO A WORD
0134 0079 FB98          BANZ   ZLOOP,*-,AR0    LOOP
     007A 0078
0135 007B
0136 007B 8201          IN     YN,DA           TOSS FIRST (OLD) SAMPLE, RESET BIO
0137 007C E201          OUT    YN,DA
0138 007D D001          LALK   >2000           A 1 IN Q13 FORMAT
     007E 2000
0139 007F 6006          SACL   ONE
0140 0080
0141               * HERE AT START OF 512-SAMPLE SUMMATION LOOP
0142 0080
0143      0080 CALCLP   EQU    $
0144 0080 CA00          ZAC
0145 0081 6002          SACL   XSUM            INIT SUMS TO 0
```

```
SINDSP     320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                      PAGE 0006

0146 0082 6003          SACL   XSUM+1
0147 0083 6004          SACL   YSUM
0148 0084 6005          SACL   YSUM+1
0149 0085 D001          LALK   512             INIT SUMMATION COUNT IN AR0
     0086 0200
0150 0087 6001          SACL   YN
0151 0088 3001          LAR    AR0,YN
0152 0089 5588          LARP   AR0
0153 008A
0154               *  HERE TO AWAIT NEXT SAMPLE
0155 008A
0156 008A FA80 DAWAIT   BIOZ   GETSMP          A/D READY YET?
     008B 008E
0157 008C FF80          B      DAWAIT          NO, WAIT FOR IT
     008D 008A
0158 008E
0159      008E GETSMP   EQU    $
0160 008E E201          OUT    YN,DA           ECHO FILTERED DATA TO ANALOG OUTPUT
0161 008F
0162 008F 8200          IN     XN,DA           GET INPUT SAMPLE
0163 0090 9800          BIT    XN,8            IS IT < 0?
0164 0091 F880          BBZ    GS1             NO, SKIP SIGN EXTENSION
     0092 0097
0165 0093 2000          LAC    XN
0166 0094 D005          ORK    >FF00           EXTEND THE SIGN BIT
     0095 FF00
0167 0096 6000          SACL   XN
0168      0097 GS1      EQU    $
0169 0097 3C06          LT     ONE             MULTIPLY INPUT SAMPLE BY 1
0170 0098 3800          MPY    XN              TO GET IT INTO HIGH ACCUM
0171 0099 CE14          PAC
0172 009A FE80          CALL   FIL             EXECUTE THE FILTER WITH INPUT IN ACCUM
     009B 0021
0173 009C
0174               * ADD THE FILTER OUTPUT TO ITS SUM
0175 009C
0176 009C 4004          ZALH   YSUM            GET 32-BIT SUM INTO ACCUM
0177 009D 4D05          OR     YSUM+1
0178 009E 3C01          LT     YN              GET FILTER RESULT
0179 009F 3801          MPY    YN              SQUARE IT
0180 00A0 CE15          APAC                   SUM IT
0181 00A1 6804          SACH   YSUM
0182 00A2 6005          SACL   YSUM+1
0183 00A3
0184               * ADD THE INPUT SAMPLE TO ITS SUM
0185 00A3
0186 00A3 4002          ZALH   XSUM            GET 32-BIT SUM INTO ACCUM
0187 00A4 4D03          OR     XSUM+1
0188 00A5 3C00          LT     XN              GET INPUT SAMPLE
0189 00A6 3800          MPY    XN              SQUARE IT
0190 00A7 CE15          APAC                   SUM IT
0191 00A8 6802          SACH   XSUM
0192 00A9 6003          SACL   XSUM+1
0193 00AA
0194               * LOOP BACK UNTIL 512 SAMPLES DONE
0195 00AA
0196 00AA FB90          BANZ   DAWAIT
```

```
SINDSP     320C25 FAMILY MACRO ASSEMBLER  PC 3.0 87.055    13:32:28  05-17-93
SINAD PROGRAM                                                      PAGE 0007

     00AB 008A
0197 00AC
0198               * CALCULATE THE SIGNAL-TO-NOISE RATIO
0199 00AC
0200 00AC 4004          ZALH   YSUM            GET 32-BIT SUM (FILTERED)
0201 00AD 4D05          OR     YSUM+1
0202 00AE 6F04          SACH   YSUM,7          SAVE AS NORMALIZED 16-BIT RESULT
0203 00AF 4002          ZALH   XSUM            GET 32-BIT SUM (INPUT)
0204 00B0 4D03          OR     XSUM+1
0205 00B1 6F02          SACH   XSUM,7          SAVE AS NORMALIZED 16-BIT RESULT
0206 00B2 4004          ZALH   YSUM            LOAD FILTERED (NOISE) TERM FOR
0207 00B3 CB0F          RPTK 15                FRACTIONAL DIVIDE
0208 00B4 4702          SUBC   XSUM            DIVIDE BY INPUT (SIGNAL+NOISE) TERM
0209 00B5 6002          SACL   XSUM            SAVE RESULT (UNSIGNED 16-BIT FRACTION)
0210 00B6 E002          OUT    XSUM,HSHAKE     OUTPUT TO PC FOR DISPLAY
0211 00B7
0212 00B7 FF80          B      CALCLP          START AGAIN
     00B8 0080
0213 00B9
0214                    END
NO ERRORS, NO WARNINGS
```

Listing 1—TMS320C25 assembly language listing of the DSP SINAD program. Assembling this program generates an object file that is subsequently loaded into the DSP board by the PC program of Listing 2. This program can be downloaded from the ARRL BBS (203 666-0578) in file QEXSINAD. ZIP.

```
Page 1: sinad.c


/*  TAPR DSP-1 SINAD Meter
 *
 *  J. Bloom, KE3Z
 *  12/27/92
 *
 *  Used in conjunction with the SINDSP.ASM program running on the
 *  TAPR DSP-1 TMS320C25 board.  This program accepts SINAD ratios
 *  from the DSP board, converts the ratio to dB, and displays the
 *  result on a bar-graph display on the CRT screen using Borland's
 *  conio.h routines.
 */

#include <stdio.h>
#include <stdlib.h>
#include <bios.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <math.h>
#include <string.h>

#include "dspl.h"         /* DSP-1 port definitions etc. */

#define arysiz(x) (sizeof x / sizeof x[0])

unsigned char ctrlreg;              /* Needed by the DSP library routines */
int iobase = DSP_DEFBASE;           /*   "     "   "    "      "        "     */

#define DCH 'X'                     /* Bar-graph display character */
char *dspfile = "SINDSP.MPO";       /* File to load into DSP board */

int maxn;                           /* Maximum display column */
double mult;                        /* Display multiplier */

/*  s e t r a n g e
 *
 *  Sets up the display for normal (0-40) or expanded (0-15)
 *  display range.
 */
void
setrange(int expanded)
{
    clrscr();
    if (expanded) {          /* 0-15 dB displayed in 60 columns */
        maxn = 60;
        mult = 40.0;
        gotoxy(1, 1);
        cputs("                                        1   1   1   1   1   1");
        gotoxy(1, 2);
        cputs("0---1---2---3---4---5---6---7---8---9---0---1---2---3---4---5");
    } else {                 /* 0-40 dB displayed in 40 columns */
        maxn = 40;
        mult = 10.0;
        gotoxy(1, 1);
        cputs("         1         2         3         4");
        gotoxy(1, 2);
```

```
Page 2: sinad.c


        cputs("0----|----0-v--|----0----|----0----|----0");
    }
}

void
usage(void)
{
    fputs("Usage: sinad [-n] [dspfile]\n", stderr);
    fputs("       -n       display numeric (debug) result\n", stderr);
    fputs("       dspfile = .MPO object file to load\n", stderr);
    exit(1);
}

/*  m a i n
 */
int
main(int argc, char *argv[])
{
    unsigned int x;
    int i, j, n, ln = 0;
    char c;
    char num = 0, exp = 0, smooth = 1;
    double d;
    double smoothed[10];
    int sidx = 0;

    for (i = 1; i < argc; i++)
        if (argv[i][0] == '-')
            switch (toupper(argv[i][1])) {
            case 'n':
                num = 1;
                break;
            default:
                usage();
            }
        else
            dspfile = argv[i];
    /* Reset the DSP board and load/run the program */
    dsp_reset(0);
    if (dsp_file_load(dspfile) != 0) {
        perror(dspfile);
        return 1;
    }
    dsp_reset(1);
    setrange(exp);            /* Start in expanded range */
    do {
        /* Loop waiting for a SINAD measurement from the DSP board */
        while ((inportb(iobase+DSP_CTLSTAT) & DSP_S_WROTE) == 0)
            if (bioskey(1) != 0)
                switch (c = (bioskey(0) & 0xff)) {  /* Key command */
                case 0x1b:            /* Esc = exit */
                    return 0;
                case 'r':
                case 'R':             /* R = toggle range */
                    exp = !exp;
                    setrange(exp);
```

```
Page 3: sinad.c


                    ln = 0;
                    break;          /* # = set smoothing */
                case '0': case '1': case '2': case '3': case '4':
                case '5': case '6': case '7': case '8': case '9':
                    smooth = c - '0';
                    if (smooth == 0)
                        smooth = 10;

        /* Get a SINAD reading from the DSP board */
        x = inport(iobase+DSP_HSHAKE);
        if (x == 0)           /* "Shouldn't happen" */
            x = 1;
        if (num) {            /* Numeric (debug) display */
            gotoxy(1, 2);
            clreol();
            d = 10.0 * log10(65536.0 / (double)x);
            i = d + 0.5;
            cprintf("%5u %3d %g\n", x, i, d);
        } else {              /* Graphic display */
            /* Calculate # columns to "light up" */
            n = mult * log10(65536.0 / (double)x) + 0.5;
            if (n > maxn)
                n = maxn;
            /* Save for smoothing */
            smoothed[sidx] = n;
            /* Calculate smoothed average */
            n = 0;
            for (i = 0, j = sidx; i < smooth; i++) {
                n += smoothed[j--];
                if (j < 0)
                    j = arysiz(smoothed) -1;
            }
            n = n / (double) smooth;
            sidx++;
            if (sidx == arysiz(smoothed))
                sidx = 0;
            /* Update display */
            if (n > ln) {               /* Greater than last time */
                gotoxy(ln+1, 3);
                for (i = ln; i <= n; i++)
                    putch(DCH);     /* Make the display bar longer */
            } else if (n < ln) {    /* Less than last time */
                gotoxy(n+2, 3);
                clreol();           /* Shorten the display bar */
            }
            ln = n;
        }
    } while (1);
}
```

Listing 2—C language PC program to display the output of the SINAD program of Listing 1. The SINAD reported by the DSP program is the ratio of Eq (1). This is converted to decibels using Eq (3). The result is used to display a "bar graph" on the CRT screen. While running, striking the R key changes the range of the display, while striking a number key selects the amount of smoothing (averaging) of the display. The functions that load the program into the DSP board aren't shown.

Digital Communications

Harold E. Price, NK6K

Forward Error Correction

There are several ways of attempting to send perfect data over imperfect RF links. One is ARQ, Automatic Repeat Request. As is done in AX.25, enough information is added to the data for the receiving station to determine whether the data was received correctly. If it was not, the computer automatically requests a retransmission. This is more efficient than the use of error-correcting codes when a data error is not likely to occur. It is *required* if the data must be received perfectly.

Another method is FEC, Forward Error Correction. In this scheme, you assume that there will be errors, and/or the other side of the link can't send an acknowledgment. FEC sends enough additional information along with the data to allow the receiving station to detect errors and correct them. This does not guarantee perfect data reception; the original data cannot be recovered if too much information is lost. Perfect data communication requires ARQ along with FEC.

Although an FEC mode has always been available in AMTOR, FEC hasn't been used with packet radio up to now for several reasons. First, good FEC is computationally intensive and is usually implemented in hardware. General purpose CPUs haven't become fast enough to handle high data rates until recently. Second, pure FEC adds a fixed amount of overhead. Even if the redundant information is not needed, it is always sent. On most VHF/UHF links, overhead caused by noise-created error retries is less than the fixed overhead of common FEC codes. Most metropolitan area retries are caused by collisions.

5949 Pudding Stone Lane
Bethel Park, PA 15102
email: nk6k@amsat.org (Internet)
71635,1174 (CompuServe)

Finally, the ubiquitous TNC, with its HDLC chip, hard CRC checking, and AX.25 protocol, does not lend itself to easy FEC upgrades. We'll address the first of these problems in this month's column.

Compute power is now more readily available. FEC is put to good use in the amateur world for links with a very narrow margin, as CLOVER does on HF. It can also be used when the link margin would be good if not for the presence of an interfering signal, as is the case in Southern California with military radar on 70 cm. You can move away, as I did (it was one of the reasons, anyway), or you can try to do something about it, as Phil Karn, KA9Q, explains below. Phil recently moved to Southern California. He spoke on FEC at this year's TAPR meeting, and has provided the summary of that talk that appears below.

KA9Q on FEC

Forward Error Correction (FEC) has been around for a long time, but only recently has it become practical for radio amateurs to experiment with it using low cost computing equipment.

Much of the research into FEC came from NASA's need to make the most of low-power, very-long-distance communication links from planetary probes. The specific technique I will describe later was used for the *Pioneer* 10 and 11 probes that were launched in the middle 1970s, becoming the first man-made objects to escape the solar system. Thanks in part to FEC, these probes are still tracked by NASA's deep-space network.

Related (though different) schemes were used on later deep-space missions, such as *Voyager* 1 and 2.

All forward-error-correction schemes apply the same basic principle from Shannon's communication theory. According to Shannon, the capacity of a noisy, band-limited channel depends on both its signal-to-noise ratio and its bandwidth:

$$C = B \log_2(1 + S/N)$$

where

C = channel capacity, bits/sec
B = channel bandwidth, Hz
S = received signal power, watts
N = equivalent received noise power, watts
$\log_2$ = base 2 logarithm

Sometimes the signal-to-noise ratio, S/N, is replaced by the equivalent term S/N_oB, where N_o is the equivalent receiver noise power density in watts per hertz and B is the channel bandwidth, as before. This form points out that the received noise power is proportional to the receiver bandwidth; a wide receiver "lets in" more noise than a narrow one.

If we rewrite Shannon's equation to use parameters more commonly used with RF modems, we get:

$$C = B \log_2(1 + (E_b/N_o) \times (C/B))$$

where

E_b = received energy per bit, joules (watt-seconds)
N_o = equivalent received noise density, watts/Hz

The ratio E_b/N_o is the figure of merit for all digital modems; the lower we can make it, the less RF power we'll need to send data at a given rate. Now suppose we have infinite bandwidth available. How low can we make E_b/N_o?

The answer is –1.6 dB, the "Shannon bound." It is theoretically impossible to go lower. But this is still a lot lower than what we see in practice. The best RF binary modems (BPSK) require S/N ratios of at least 10 dB for good performance. Less efficient modems (eg, FSK) require at least 13-14 dB, and the very worst (AFSK/FM) may require 20 dB or more. And these figures are independent of the receiver's bandwidth; once you have enough bandwidth, more doesn't buy you anything.

Unless you use FEC. This is the key to trading off bandwidth for power.

With it we can approach (but not reach) the Shannon bound. In practice, it's fairly easy to take the required E_b/N_o for PSK to 4-5 dB, but 2 dB is about the practical limit with known techniques. Still, this is quite a bit better than the 10 dB required for PSK without FEC.

These gains, called "coding gains," are quite real. If the FEC in use has a coding gain of 5 dB, you can drop your transmitted power (or antenna gain) by 5 dB and still transfer data *at the same rate as before!* Anyone involved in, say, DXing or moonbounce operation knows that generating large amounts of effective radiated RF power is expensive and getting more so all the time. FEC reduces the need for RF power, and the computers needed to do FEC are getting cheaper and more powerful all the time. FEC thus represents a real triumph of "brains over brawn."

These figures are based on an important assumption: that all of the noise (and interference, if any) is *additive* and *Gaussian*. Additive simply means that the channel (and your receiver front end) is linear. Gaussian implies *white*—the kind of noise you get from the thermal motion of molecules in a preamp, or (usually) from the cosmic background. But not all noise is Gaussian. On the 70-cm ham band, for example, one important source of noise in many areas is military radar—and the short, high energy pulses from these things are about as non-Gaussian as you can get!

The effect of strong radar QRM is to cause regular bit errors at a rate that depends on the radar's pulse duration and repetition rate and the duration of each data bit. To illustrate the problem, I'll consider a 56-kbit/s amateur link being strongly QRMed by a type of 70-cm military radar that seems common to both the east and west coasts of the US: a pulse duration of 10-20 microseconds at a pulse rate of about 300-400 Hz. Since the radar pulses last about as long as a single bit at 56 kbit/s (17.9 microseconds to be more precise), simply blanking out the radar pulse won't help. Every 140-186 data bits or so, a radar pulse will take out one (possibly two) data bits. And if the radar pulses are, say, 40-dB stronger than our data signal, the only way to overcome the problem by brute force is to crank up the power by 40 dB—and this may well be impossible.

FEC provides a more elegant way out. Many error-correcting codes can easily handle the relatively low bit-error rate of 0.5-0.7% on our radar-QRMed channel, at some cost in through-put. And for our example of a radar that's 40-dB stronger than our desired signal, this represents a coding gain of 40+ dB! Here the choice is clear: FEC is *far* more cost effective than increased transmitter power or antenna gain. Indeed, another motivation (besides NASA deep space exploration) behind the early development of FEC was the need to protect US Navy shipboard data links from strong radar interference.

FEC is now a standard technique found in many commercial and military radio modems. There are many different FEC codes, about which many textbooks have been written. However, I can summarize them very briefly while introducing the technique I've been experimenting with.

Error-correcting codes fall into two broad categories: block and convolutional. Block codes, as the name implies, work on fixed-sized blocks of data. These work well when the medium is already divided into fixed blocks, such as words in a computer memory or blocks on a magnetic disk, or when the medium is a semi-infinite stream of bits that can be divided into blocks (such as a compact disk). The Hamming code is one block code that's popular in computer memory designs such as those on the Microsats, and the Reed Solomon code is another (it's used on compact disks). Similar (BCH) codes are used to protect the digital signaling messages in cellular telephone systems.

The other category of FEC, the convolutional code, works with arbitrary amounts of data. This makes it more suitable for variable-sized blocks of data such as those in packet radio, and this is the one I've chosen for my experiments.

Convolutional codes are extremely easy to generate. One feeds the data to be sent down a shift register. Taps at preselected points on the shift register feed networks of exclusive-OR gates. The outputs of these XOR networks are actually sent over the channel. A little thought will show that any given data bit will continue to affect the outputs of the XOR networks until that bit "falls off" the end of the shift register; this length is called the constraint length, *K*, of the code. The number of XOR networks determines the rate, *r*, of the code; in a rate 1/2 code (read as "rate one-two"), there are two output bits (produced by two different XOR networks) for each input data bit. Other rates are possible, for example, a rate 3/4 code could be generated by shifting in three data bits and then sending the outputs of four different XOR networks.

The taps to use are determined by the *polynomials* of the code in use. Many good code polynomials are tabulated in textbooks, so we don't have to find them ourselves.

Generating a convolutional code is the easy part; decoding it is the fun part. The receiver has to determine which bits were sent by observing the outputs of the sender's encoder (with individual bits possibly corrupted by channel noise) and comparing them to a local copy of the encoder. There are two types of algorithms for decoding convolutional codes: parallel and sequential. The parallel (Viterbi) algorithm tries all possible data sequences in parallel, eliminating at each step those that could not possibly be correct. Because of its parallel nature, the Viterbi algorithm is a natural for hardware implementation, and chips are commercially available that will run at multi-megabit speeds.

The sequential algorithm, on the other hand, tries only one sequence of data bits at a time. As long as the sequence being tried seems "reasonable," ie, its encoded representation faithfully matches what is actually received, the decoder continues forward until it finishes regenerating the sender's entire message. Should a channel error cause the decoder's own copy of the encoder start to diverge from the incoming encoded stream, however, the decoder will "back up" and try other sequences until it finds another one that gets it back on track.

The big advantage of the sequential decoder over the Viterbi decoder is its speed when the channel error rate is low, especially when the constraint length (encoder shift register length, *K*) is long. The decoder just keeps moving forward, rarely if ever backing up. Indeed, because it must try 2^K paths in parallel at all times, Viterbi decoding is impractical when *K* is more than about 9, but sequential decoding is routinely done with K=32 or 48. On the other hand, sequential decoding "blows up" (fails to make progress) when the error rate becomes very high, while a Viterbi decoder will always decode at the same rate (although it may produce incorrect results).

In general, sequential decoding is preferable when you have to implement it in software, while Viterbi decoding is the way to go if you have one of the specialized decoder chips available. The variable decoding time of sequential decoding is not a real problem in packet radio since the sender can always time out and resend a packet if the receiver fails to decode it in time.

To see if sequential decoding is a

practical technique for amateur packet radio, I implemented the Fano algorithm (the most popular sequential decoding technique) in C on a 33-MHz 486. I used the same rate 1/2 constraint length, K=32 code that was used on the *Pioneer* 10 and 11 missions and documented in several textbooks as a "NASA Planetary Standard" code. My decoder ran fast enough to keep up with a channel rate of 56 kbit/s (28 kbit/s user data rate) as long as the channel error rate was below about 2%, well above that required to deal with the military radar QRM figures mentioned earlier.

This demonstrates that with modern microcomputers now available, this 30-year-old algorithm is finally a practical alternative for high-speed amateur packet radio. Nevertheless, much more work remains to be done before a practical system can be built.

The first requirement is a new packet framing format. Because FEC can decode a packet successfully even when many of its bits are in error, we need a more reliable start-of-frame indication than the HDLC flag. A pseudorandom "sync vector" of, say, 64 bits would do. Such a long sequence could be reliably detected with a reasonably small probability of false alarms, even if a few of the bits are corrupted. This requires a correlator to look for the sync sequence, but this can easily be done in software on the same CPU that does the Fano decoding.

Another practical feature is a way to use FEC only when it is really needed, since this will save fully half of the channel capacity (for a rate 1/2 code). One way to do this starts with a different convolutional code with the "systematic" property—the original data appears in the output as one of the code bits. (You implement this in the encoder by making one of the XOR networks have only one tap on the shift register). Systematic codes perform slightly worse than nonsystematic codes, but not by much.

The first transmission of each frame could consist of just the original data, plus a CRC. If no channel errors occur, the receiver will verify the CRC and acknowledge the frame just as it does in conventional packet radio. The sender then goes onto the next frame without sending the additional parity information, and without invoking a decode operation at the receiver.

On the other hand, if the first frame is received with errors, the receiver can signal this fact with a negative acknowledgment ("NAK"—as an aside, another advantage of a long sync vector is the ease with which an errored frame can be distinguished from purely random noise). The sender then sends not the original data again, but the first set of parity bits from the convolutional encoder. The receiver combines this frame with the previous version it received and attempts to decode it as a rate 1/2 code. If it succeeds, it finally acknowledges the frame and the sender proceeds to the next frame. If this also fails, a second set of parity bits (different from the first) could be sent and the receiver could attempt a decode of a rate 1/3 code.

Note that successful decoding is possible even if both the original data and parity frames are riddled with errors, as long as the total percentage of bits in error doesn't exceed the error correcting capability of the code. This is in stark contrast to present (noncoded) practice, where the sender must blast the same frame again and again until it finally gets lucky and gets one through without errors. The power of FEC comes from being able to make the maximum use of received information, even when it is contaminated with errors. And since the success of a transmission depends on the error rate, not the absolute total number of errors in the frame, there is no longer a need to keep frames small (and turnaround overhead large) to ensure reasonable throughput.—*KA9Q*

NK6K here. FEC with ARQ is already in use on the ham bands. HAL's CLOVER HF modem uses Reed-Solomon coding at 60%, 75% or 90% efficiency (ratio of user data to total data). Experimenters interested in FEC at higher speeds and frequencies should contact Phil: karn@unix.ka9q.ampr.org.

Windows

A surprising number of hams, myself included, are using *Windows* on our PC clones. In my case, a client wanted me to produce documentation using *Word for Windows,* so I was forced into it. Once there, I found that the ability to exchange documents, graphs, and graphics with similarly equipped friends was enticing. Much of my DOS software, including spacecraft simulators and command programs, runs fine in the *Windows* DOS box. The pseudo-multi-tasking ability (you can still get hung up by the floppy and the printer) is good enough for some tasks. The price-performance of packages like *Word* and *Excel* is quite amazing. The new standard price for software is $99 to $129. If you can't find a way to bootstrap from zero to "competitive upgrade," you aren't trying. In my case, I bought a perfectly legal $5 version of *Wordstar* from DAK and turned it into a $99 *Word*, a savings of at least $300.

Now that many of us are on *Windows,* we're upgrading our ham-related software. *Windows* provides several methods of data exchange between programs. The authors of various Ham packages need to make better use of these facilities. Dave New, WB4SBE/8, has been talking this up on the Internet. I asked him to expand on his comments for *QEX* readers; you'll find them in the Correspondence section of this issue.

More on Computer Control

My previous column on computer control of radios, "We're Still Doing it Wrong" (April 1993 *QEX*) has generated several comments. Some people seem chilled by the thought of the computer having priority access. Others don't think I went far enough. A friend and long-time radio/computer pioneer, Larry Kayser, VE3PAZ/WA3ZIA, wants his radio to have a front panel as shown in Fig 1. He writes:

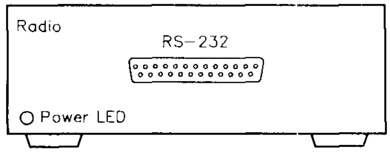


Fig 1

> "Your comments on the computer link into the radio are in my opinion not quite complete. Don't just do the computer link first, *also don't do the computer link with the radio having a front panel!* My TS440 sitting here is almost computer controlled except for the badly needed carrier insertion, volume control, RF gain control, and IF notch. In addition, I am stuck with all those useless function buttons on the front panel. I can do a much better human interface on my PC than this radio will ever have. I can get it to do what I want it to do, not what some designer or worse yet a marketing committee decides I want to do with my radio."

Larry runs his HF station remotely, using a Direct Digital Synthesis VFO on a PC adapter card, and old Heathkit exciters. While a faceless radio is not for everyone, there may be enough of us in the market. I had the chance to make my pitch to a collection of Ten Tec engineers and marketeers at Dayton, but I don't think they believed me. Drop them a line. □□

Correspondence

Windows Standards

I would like a serious inquiry and proposal made for a DDE (Dynamic Data Exchange or "hot-link") standard for related Amateur Radio programs written for the *Windows* environment.

Imagine a scenario where your screen is full of various windowed applications, all communicating among themselves, minimizing the amount of "double-entry" work on your part. A satellite tracker informs you of the next risetime of your favorite satellite, feeding the antenna azimuth and elevation angles to a rotor client, and the Doppler shift information to a radio control client window. You receive a DX spot on your favorite land-based packet network, and the information is automatically fed to your logging window and the radio control window, setting up the contact. You connect to the station through the satellite, using another TNC application instance, which received the required call-sign information from the first TNC window. Once connected, the log information is automatically completed with date, time, call sign, and uplink/downlink frequencies. The programs running on the screen are actually from a half-dozen different vendors, having made your choice from among several popular offerings.

Fairy tale? Not necessarily. There are a number of new *Windows*-based programs available that do logging and such (Kenwood's *HamWindows* and PDK's *Log View*, for instance), and more recently, a *Windows*-based satellite-tracking program (Paul Traufler's *WinTrak*).

Some of the logging programs sport a DDE interface to exchange data between the logging and/or radio-control application and a companion TNC program to obtain local DX-spotting information. So far, the DDE protocol in use has been proprietary, meaning that data can be exchanged only with other programs marketed by the same vendor. At least one company contacted, though, is considering publishing their protocol, to encourage third parties to write compatible modules.

One item that seems to have been ignored, though, is the needs of the satellite community. To my knowledge, no one has offered a satellite tracker with DDE-driven tracking/tuner companion clients. The vendors who support DDE in their software don't consider the satellite market large enough to pursue, and the ones who have written satellite software are satisfied with the current Kansas City Tracker/Tuner DOS TSR interface.

Some companies I talked to at Dayton are interested in DDE, but are waiting for someone to establish a standard they can write to.

Admittedly, you can get a number of quite good DOS programs to run inside DOS windows under *Windows*. They will even multitask fairly well when running in 386 enhanced mode, if you tinker around with .PIF files and such, although graphics-mode DOS applications can be especially bothersome. Even then, there is very little communications, if any, going on between any of these programs. There is no standard for automatically grabbing, say, a connected station's call sign from a DOS packet program and plugging it into a logging program. Sure, you can cut and paste, but I don't want to have to *think* about it. That's the computer's job.

Some DOS program vendors have approached this problem by supplying a monolithic do-all, be-all rig control/contest/DX-chasing/logging program. Often, these programs contain a ton of almost unrelated features that consume lots of memory and disk space. They seem to be the result of attempting to "please all of the people all of the time." Using one of these programs locks a user into a particular vendor's interface (and file system) for all his radio applications. Maybe a user likes the logging interface from vendor X, but prefers the packet interface from vendor Y. Currently, he is out of luck. In order to keep both programs synchronized, the user must resort to double-entry operation.

The monolithic ham program will eventually die a sure death, just as programs like *Symphony* and countless other all-in-one programs have. All of these programs do "everything," but don't do anything particularly well. Instead, a cooperative suite of small, high-quality, specialized modules that operate in a client/server paradigm will allow a completely different breed of user-directed on-screen integration of what the user wants, at the quality/feature level he wants, without being locked into a particular vendor's file structures or communications idiosyncrasies.

DDE is really just a stepping-stone to a smoothly integrated cooperative client/server environment. Object Linking and Embedding (OLE) is starting to supplement DDE, where appropriate. Ultimately, the user should have a drag 'n drop interface that allows him to point at a data item in a window (satellite azimuth, for instance) and "drag" the item to an input field in another window (a rotor control program, for instance), thus creating a live connection between these two applications. He should be able to save this setup, and modify it at will, to try different vendor's applications that supply similar services. This is an exciting area, and one where interested readers could take a lead position in producing a workable standard.—*Dave New, WB4SBE/8, wb4sbe@amsat.org (Internet)*

TCP/IP for Networking

I agree that more people should try TCP/IP ["Digital Communications," Feb 1993 *QEX*], but I believe that a different emphasis might get things going faster. First, I would stress that TCP/IP is for building networks—it's something to replace, say, NETROM—but it's not necessary for someone to run TCP/IP on their own machine in order to benefit from it.

The trick is for the network-savvy

hams to provide "service access ports"—JNOS BBSs, for instance—to which AX.25-only users can connect, preferably directly. From the service node, a user can telnet to any other service node, read news and mail, download files, etc. In other words, the nodes look similar to other BBS systems, except that they talk to each other via TCP/IP. The user needn't even be aware of it.

I've also concluded that Amateur Radio networking, at least in my area, will become much more useful if we stop insisting on using RF paths for every link. The Internet-based AMPRNET gateways have shown what is possible if we use other existing networks. The most obvious choice for most people is the plain old dial-up telephone network. I've been experimenting with dial-on-demand IP over phone lines, and it works quite well (as long as you have a phone line dedicated to it!). Dial-up IP capability is available commercially in the Telebit NetBlazer (which is really just a PC running a much-hacked KA9Q NOS). KA9Q NOS already does dial-on-demand, and it can easily be added to most other variants.

Whereas RF links would be nice to have, for channel availability and other reasons (this is ham *radio*, after all), I'd rather not hold up the development of the network just because the RF hardware is not available yet. This problem has kept interesting things from happening in Northern Virginia. (And I'm just as guilty of "waiting for RF" as anyone!). It's better to use phone lines, commercial packet networks, or whatever else you can scrounge, than install RF infrastructure as it becomes feasible.—*Mike Gallaher, WA2HEE*

Better Iron-On PC Patterns

[The following originally appeared on Usenet news in response to a question about photocopier films used for making printed-circuit board resist patterns. Neither the author nor ARRL warrant the products mentioned here.—Ed.]

After a lot of effort, I had limited luck with the TEK film. The biggest problem was getting good adhesion of the toner from the copier or laser printer to the film. I tried many different machines but much of the time sections of it would either fall off or smear or else come off when I peeled the film away. Getting the iron-on phase right takes some learning too. I was successful in making some moderately critical circuits, edge-coupled microstrip band-pass filters and such, however I finally gave up when I found a better way.

I got some of the new DynaArt Designs material that is now offered from quite a large number of suppliers. It is basically paper with a transparent water-soluble carrier onto which you copy or print the toner. You then iron it on as with the TEK film, but rather than peel away the film you just put the whole works, PC board and all, into water. The carrier then dissolves and the paper backing comes off leaving *all* the toner on the copper. This works much better than the film and also gets you from design (or scan) to resist with fewer steps much of the time. I can usually go from the PC-board layout tool on the computer to an etched board which is ready to have parts loaded in under an hour. About the most important aspect of this is to have good toner which gives saturated black on paper. The manufacturer suggests some sources of particularly good toners. I just used the standard paper toner which came with my HP laser printer—and it isn't even a new cartridge.

I've used the process to generate fairly width-critical circuits, 90-degree microstrip hybrids and complete image-reject mixers at 1200 MHz, with quite good results. I even managed to get reasonable yield on some very narrow lines which happened to be on one design: one pixel wide at 300 DPI which amounts to about 0.003 inch. I don't recommend trying to do fine lines this way, but it gives an idea of how well it all works.

The material is about $3 per A-size (8.5 by 11 in.) sheet, but you can pack many circuits onto a sheet and only cut out and use the ones you want. I think that the cost/effectiveness ratio is much better with this process than with anything else I've tried. A bunch of people appear to handle the DynaArt material (I think it's about $15 for 5 sheets):

DynaArt Designs
3535 Stillmeadow Lane
Lancaster, CA 93536-6624
805 943-4746 (0700-1700 PST)

I bought mine from

DC Electronics
PO Box 3203
Scottsdale, AZ 85271
800 467-7736
800 423-0070

I have no connection with either of these places except as a customer.—*Glenn Elmore, N6GN, N6GN @ K3MC (packet BBS), glenn@SantaRosa.ampr.org (amateur IP), glenne@sr.hp.com (Internet)*