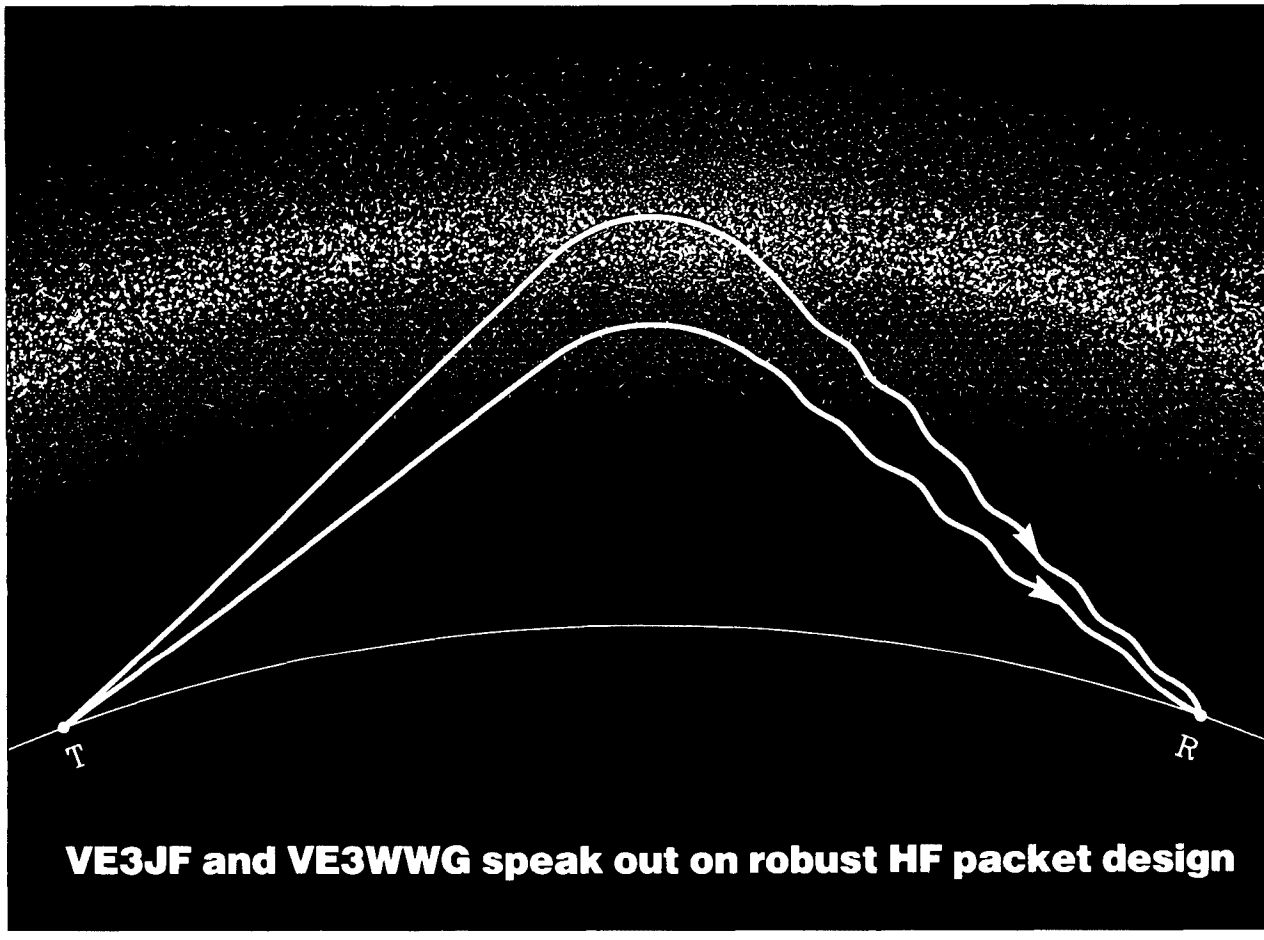
$1.75

QEX 97

WITH Gateway

ARRL Experimenters' Exchange

MARCH 1990



VE3JF and VE3WWG speak out on robust HF packet design

QEX: The ARRL
Experimenters' 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.

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: MCI MAIL ID:215-5052
(user name ARRL)
Telemail address: ARRL

Subscription rate for 12 issues:
In the US by Third Class Mail:
ARRL Member $10, nonmember $20;

US, Canada and Mexico by First Class Mail:
ARRL Member $18, nonmember $28;

Elsewhere by Airmail:
ARRL Member $38, nonmember $48.

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.

TABLE OF CONTENTS

COLUMNS

MARCH 1990 QEX ADVERTISING INDEX

THE AMERICAN RADIO RELAY LEAGUE, INC

The American Radio Relay League, Inc, is a noncommercial association of radio amateurs, organized for the promotion of interest 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 the 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.
MCI MAIL (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 7009, Station E, London, ON N5Y 4J9, tel 519-660-1200.

Purposes 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 double 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. . .

1990 the Year of the Amateur Satellite Program

Seven new amateur satellites are aloft. Six were launched on January 21 and the seventh on February 7. Here's the list with downlink frequencies, main functions and sponsors:

- UoSAT-OSCAR-14 (UoSAT-D)—435.070 MHz, packet, University of Surrey
- UoSAT-OSCAR-15 (UoSAT-E)—435.120 MHz, CCD camera, University of Surrey AMSAT-OSCAR-16 (PACSAT)—437.025/437.050 packet, 2401.1 MHz SSB beacon, AMSAT-NA
- DOVE-OSCAR-17—(BRAMSAT) 145.825 digital voice, 2401.2 SSB beacon, Brazil AMSAT
- WEBER-OSCAR-18 (WEBERSAT)—437.075/437.100 MHz, CCD camera, Weber State College
- LUSAT-OSCAR-19—437.125 CW beacon, 437.150 packet, AMSAT Argentina
- FUJI-OSCAR-20 (JAS-1b)—435.795 CW/PSK beacon, 435.8-435.9 MHz analog mode, 435.910 MHz PSK digital (packet) mode

The list of acknowledgements for those who worked on these satellite projects would be endless—then we'd probably forget someone. Suffice it to say that the organizations and inviduals involved have made this "1990 the Year of the Amateur Satellite Program," according to a resolution by the ARRL Board of Directors adopted at their January meeting (Minute 62).

These are migratory birds. If you don't have satellite-tracking software, check the Gateway column in this issue of *QEX.*

If you need some further background on the MICROSATs, refer to the two-part article, entitled "Microsat: The Next Generation of OSCAR Satellites," which appeared in the May and June issues of *QST. QEX* and *QST* will be bringing you more information on this series of amateur satellites in the coming months.

Welcome, Gateway

With this issue of *QEX*, we also bring you a new column called Gateway to keep you current on packet radio news. The newsletter *Gateway* premiered on August 14, 1984, as means of providing a news feed to packeteers. *Gateway*'s circulation grew to well over 3,000 at last count. Much has happened in amateur packet radio these 5 ½ years. It's been a revolution. As you can see, Stan is sticking with Gateway, the column. Stan is also conducting a new *QST* column called Packet Perspective.

Readers who subscribed to *Gateway* instead will receive *QEX* over the period of time left on their *Gateway* subscriptions. Those having subscriptions to both *QEX* and *Gateway* will have the two periods remaining added together.

Incidentally, we've tried to make as few changes as possible in folding *Gateway* into *QEX*, so it is in the same format for now.

We welcome articles from experimenters on a host of technical subjects: packet radio, microwaves, satellites (especially MICROSATs and ground station technology), spread spectrum, ATV, CCITT Group 3 FAX, future systems, and neat technical stuff. If you don't want to write an article, feel free to send us some information or comments for the Correspondence column. Authors like to get feedback, so please write them directly and drop *QEX* a copy if you think it appropriate.

From time to time, we print interesting articles that have appeared in a magazine of a foreign society. Those originally in English, such as from RSGB, are no production problem, as we use them camera-ready. Sometimes, we get complaints that the components used in such articles are not available in the States. We don't have the resources to select and procure U.S. equivalents, build up the circuits and validate the design as we might do for an original article published in *QST.* These are articles that you might not otherwise see, and while there may be problems translating them in to U.S. parts, experimenters should be much better off than if they had to design the thing from scratch. If you are interested in replicating one of these foreign designs and can find U.S. equivalents, please pass this information on to other *QEX* readers via the Correspondence column.

There is a lot of interesting experimental work happening overseas. We have some feelers out for possible *QEX* correspondents in Region 1 (Europe and Africa) and Region 3 (Asia). If you know someone living in either of these areas who could act as *QEX*'s eyes and ears in the experimental communities there, please drop me a line.

We're trying to make *QEX* more responsive to the needs of experimenters. Please let us know what you would like to see in the months ahead, and we'll do what we can to bring it to you.

Finally, advertisements help pay the costs of publication. When you talk to one of our advertisers, let them know that you saw their ad in *QEX.—W4RI*

High Accuracy Noise Bridge

By Frederick H. Schmidt, K4VA
3848 Parkcrest Dr NE
Atlanta, GA 30319

Greatly improved accuracy and resolution can be obtained with larger dials and three additional components.

The noise bridge is a valuable instrument for measurement of unknown impedances at radio frequencies. Although the noise bridge described in the *ARRL Handbook* is adequate for approximate measurements, it has insufficient accuracy and resolution for my requirements. However, it does give a sharp, deep null when carefully balanced indicating the basic soundness of the circuit. The noise bridge can be greatly improved by use of large dials, careful component selection, installation of two compensating capacitors and one compensating inductor, and by employing a better calibration technique. A BASIC computer program is presented for rapid and effortless conversion of dial readings into series resistance, series reactance, impedance, and phase angle of the unknown.

A photograph of the noise bridge is shown in Fig 1. The dials are nearly three inches in diameter. The resistance dial has 100 divisions covering 270 degrees and the reactance dial has 100 divisions covering 180 degrees. These dials provide good resolution; unfortunately, they also provide enhanced ability to observe measurement errors, particularly at high frequencies and when measuring high impedances.

Fig 2A shows the basic Wheatstone bridge circuit. It is a very simple circuit consisting of four impedance arms, Z1 through Z4. The generator is connected across opposite corners and the detector, usually the station receiver, is connected across the remaining opposite corners. When balanced by adjusting one or more of the arms, Z1 / Z2 = Z3 / Z4. This is an ideal bridge. Unfortunately, real bridges are more complex circuits containing undesirable but inevitable stray reactances in addition to the desired elements.

Fig 2B shows the noise bridge circuit. Potentiometer R1 and variable capacitor C1 in series form the variable arm one of the bridge. The unknown impedance ZU connected in series with fixed capacitor C2 forms arm two. L3 and L4 form the remaining two arms. They are two windings of a toroidal trifilar transformer. The third winding, L5, couples the noise generator into the bridge. If L3 and L4 are equal, the impedance of the unknown arm is equal to the impedance of the variable arm when the bridge is balanced.

The three predominant sources of error in the noise bridge are the unequal inductances L1 and L2, the stray capacitance CST, and the slight inequality of the transformer windings L3 and L4. Unless compensated, these error sources will cause the zero reactance setting of C1 to change with frequency and with unknown impedance even when the unknown is purely resistive.

L1 and L2 represent the stray wiring inductances of arms one and two respectively. In my bridge, the variable capacitor and the potentiometer are mounted three inches apart to accommodate the large dials. This results in a large value of L1 due to the wire lead lengths. It is, perhaps, a good example of how *not* to construct a piece of high frequency equipment. Shorter lead lengths would introduce mechanical complexity from flexible shafts or right angle drives, so I accepted a large value of L1 and compensated for it by increasing L2. Even if L1 were reduced to a minimum, L2 would still have to be adjusted to achieve balance. L2 is adjusted by installing a suitable length of solid hookup wire between the unknown terminal and C2. Capacitor CST represents the stray capacitance shunting the unknown impedance ZU. It is composed of stray wiring

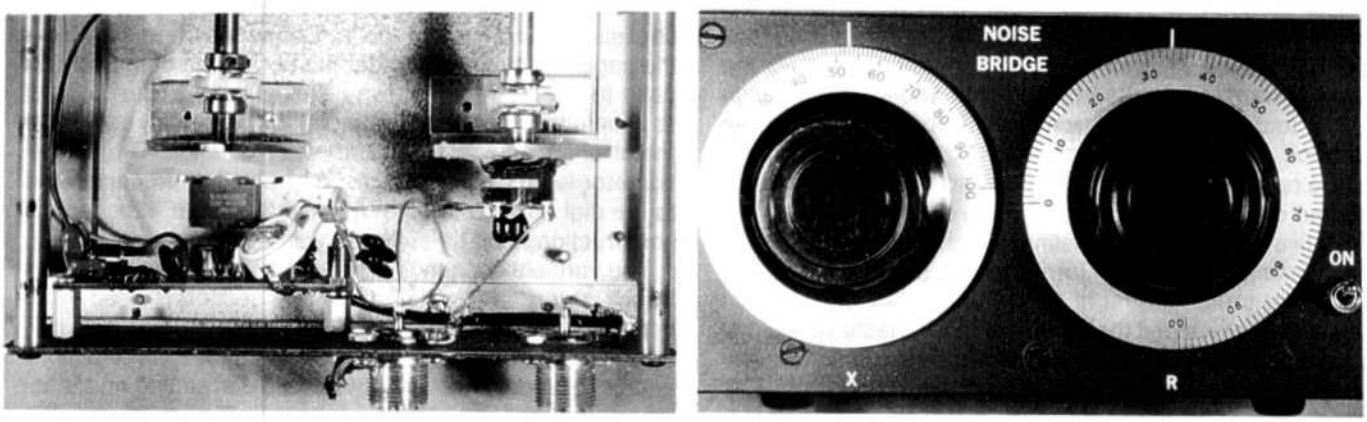


Fig 1—Interior and exterior views of noise bridge.

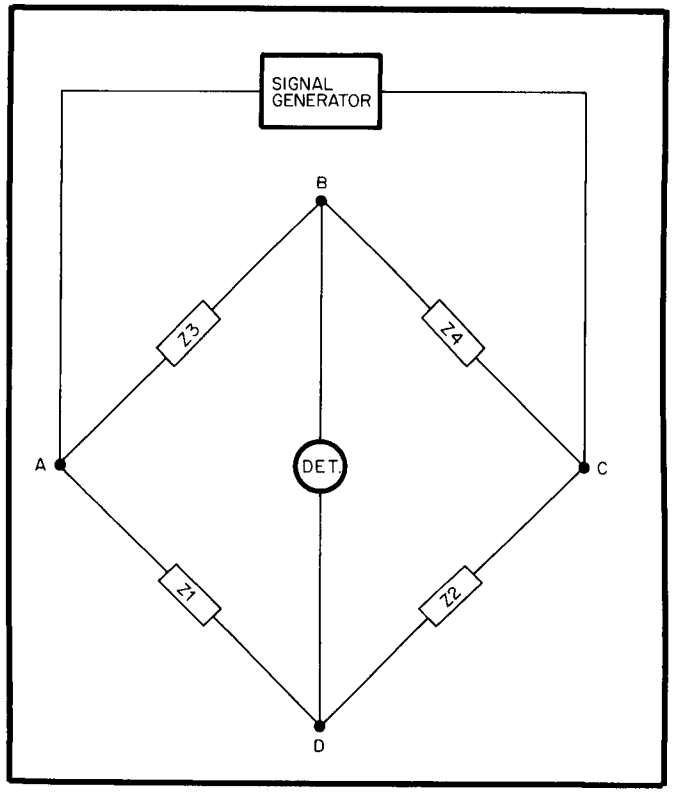


Fig 2A—Basic Wheatstone bridge circuit.

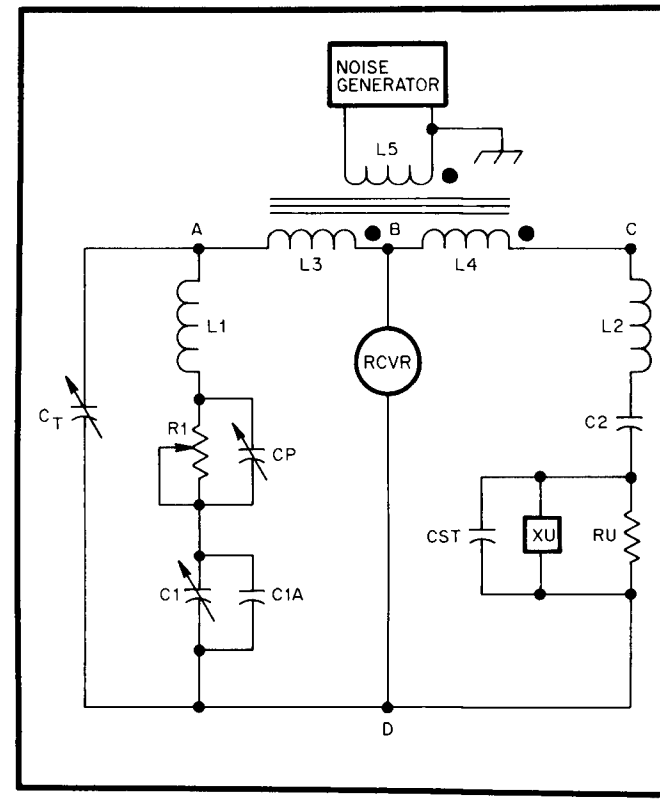


Fig 2B—Schematic diagram of noise bridge.
C1—355 pF variable, Radio Shack Cat #272-1337.
C1A—20 pF SM.
C2—120 pF SM.
CP—3-15 pF trimmer.
CST—Stray circuit capacitance.
CT—3-15 pF trimmer.
L1,L2—Stray circuit inductances. See text.
L3,L4,L5—Trifilar transformer. See text.
R1—250-Ω linear Bourns MOD POT, Allied Cat 754-9407
RU—Unknown resistance.
XU—Unknown reactance.

capacitance and the capacitance of the unknown terminal itself. CST is compensated by means of CP, a 3- to 15-pF trimmer connected directly across potentiometer R1. When properly adjusted, CP will be equal to CST. CT is a 3- to 15-pF trimmer connected between junctions A and D (across the unknown arm) to compensate for capacitive unbalance of L3 and L4.

If a pure resistance is connected to the unknown terminal and L2, CP and CT are properly adjusted, R1 and RU will be equal and C1 and C2 will be equal for all frequencies and for all values of RU. For an unknown consisting of a resistor in series with a capacitor, C1 will no longer equal C2 and R1 will no longer equal RU, but the bridge will remain balanced at all frequencies.

CONSTRUCTION AND COMPONENT SELECTION

The printed circuit board is the same as shown in the *ARRL Handbook*. The only addition to the board is a small standoff insulator mounted in the upper-left corner near the toroidal transformer. Its function is to anchor the junction of C2 and L2. C2 goes from a pad on the board to the standoff. L2 goes from the standoff to the unknown terminal at the rear of the cabinet. CP is a 3- to 15-pF trimmer connected directly across the terminals of potentiometer R1.

T1 is a transformer consisting of eight trifilar turns of wire on an Amidon FT-37-43 ferrite toroidal core. Make up a bundle of three lengths of no. 28 enameled wire. Fasten one end in a vise and the other end in a variable speed drill chuck. Apply tension to the bundle and run the drill at slow speed until there are seven or eight twists per inch. The bundle should be smooth and uniform throughout its length. Thread the bundle eight times through the toroid. This method of construction will ensure that L3 and L4 will be of equal inductance. Because one side of primary winding L5 is grounded, there will be slightly more capacitance to ground at the dotted (right) terminal of L4 than at the undotted (left) terminal of L3. CT is a 3- to 15-pF trimmer capacitor connected from L3 to ground to compensate for this capacity difference. It is important that the transformer connections be phased as shown in Fig 2B.

Variable capacitor C1 should be insulated from the chassis to avoid ground loops. Connect a short wire from the capacitor rotor to a solder lug under one of the screws used to attach the unknown terminal. Use a plastic shaft to couple the capacitor to the dial, or better yet, use a stiff metal shaft and an insulated shaft coupler. The potentiometer is mounted on a piece of plastic and connected to the dial through another insulated coupler. Mechanical construction must be solid to avoid all traces of backlash.

The variable capacitor is a dual 350-pF capacitor, Radio Shack part no. 272-1337. Only one section is used. This capacitor is very compact and has a straight line frequency characteristic. This is desirable because it gives a more nearly linear relationship between the dial setting and reactance. C1A is a 20-pF fixed capacitor in parallel with the variable capacitor to further improve the linearity. Capacitors with a straight-line capacity curve may be used,

but the reactance curve will be highly nonlinear (hyperbolic) and the measurable range of inductive reactance will be somewhat restricted at high frequencies.

The selection of a suitable potentiometer is important. The Ohmite type AB pot specified in the *Handbook* is neither readily available, nor is it the best. A better choice is the Bourns modular potentiometer, available from Allied Electronics, catalog no. 754-9407. It is very compact, and has a one-piece molded plastic rotor and shaft. It exhibits less noise, backlash, and reactance than the Ohmite pot.

CALIBRATION AND ALIGNMENT

The calibration and alignment steps are:

1. Calibrate R1 and C1
2. Adjust C2 to equal the midscale value of C1
3. Adjust CT, L2, and CP

1. Temporarily disconnect the wire between R1 and C1-C1A. Use a digital capacitance meter to measure C1-C1A at each dial division. I used the Elenco model CM-1550, available for under $70 from Elenco Precision, 150 W Carpenter Ave, Wheeling, IL 60090. Connect a digital ohmmeter across R1 and measure its resistance at each dial division. The resistance and capacitance data will be entered into the computer program. They may also be plotted on graph paper.

2. Reconnect R1 and C1. Connect a 10- to 20-Ω quarter-watt or half-watt composition resistor from input terminal to ground. Do not install the resistor in a plug because the plug will introduce several pF of additional stray capacitance. Solder the resistor with the shortest possible leads between the unknown terminal and a nearby ground lug. Set the receiver to the lowest available frequency, for example, 1 MHz or even 500 kHz. Turn on the noise bridge and obtain a null. The capacitor C1 should be at exactly midscale for readout convenience. If it is not, change the value of C2. I found it helpful to install a small trimmer capacitor across C2 to make the final adjustment.

3. With the same test resistor in the circuit, switch the receiver to 28 MHz and rebalance the bridge. If the null setting of C1 changes, it will be necessary to adjust L2. Change the length and/or shape of L2 until the capacitor null setting at 28 MHz is exactly as it was at the low frequency.

Remove the 10- to 20-Ω resistor and solder in a composition resistor of about 150 Ω. Carefully measure the resistor with the digital ohmmeter after soldering, then set R1 to exactly the same value of resistance. Once set, do not disturb. Switch the receiver to the lowest available frequency. Turn on the bridge and adjust C1 for minimum noise. Now use a plastic tuning rod to adjust CT for a null. Alternately adjust C1 and CT for best null without disturbing the setting of R1.

Leave the noise bridge on and switch the receiver to 28 MHz. Use the plastic tuning rod to adjust CP for a null. When CP is set correctly, the null setting of C1 will be the same for all frequencies.

You may find that the null setting of R1 increases slightly at 28 MHz. I have no data on the RF characteristics of small composition resistors. I assume the slight increase (about 2%) is the result of skin effect.

At this point it is wise to go back and repeat steps 2 and 3 because the adjustments may interact.

This completes the calibration and alignment. The measurement of any composition resistor at any frequency should yield a null with C1 at midscale and R1 equal to the value of the resistor.

CALCULATION OF UNKNOWN IMPEDANCE

A number of steps are required to convert the dial readings into series equivalent impedance when the unknown impedance is not purely resistive. Fig 3A shows the bridge arms Z1 and Z2. In Z1, stray capacitance CST in parallel with potentiometer R1 is in series with variable capacitor C1. At Z2, the unknown is shown as a parallel combination of RU and XU. XU may be inductive or capacitive. The unknown in parallel with CST is in series with fixed capacitor C2. The first step is to convert both arms into their series equivalents as shown in Fig 3B. The parallel-to-series conversion equations are:

$RS = RP / (1 + (RP/XP)^2)$

$XS = RSRP / XP.$

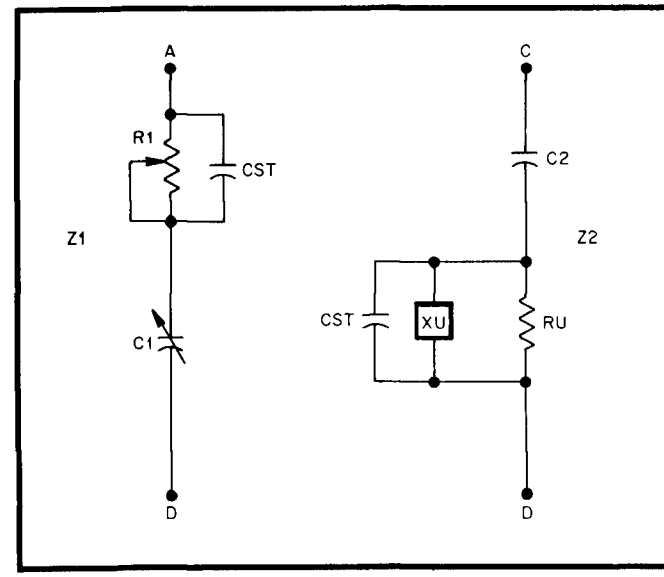


Fig 3A—Z1: R1 in parallel with CST.
Z2: Unknown ZU in parallel with CST

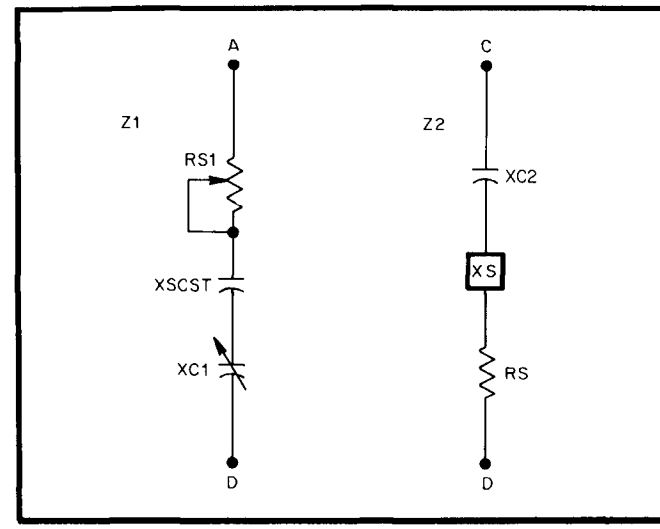


Fig 3B—Series representation of Z1 and Z2.
RS = RS1
XS = XC1 + XSCST - XC2

Because the bridge is balanced, RS1 = RS, and XC1 + XSCST = XS + XC2. Therefore, XS = XC1 + XSCST – XC2. XS is the series equivalent of XU in parallel with XCST. XS may be inductive or capacitive, depending on the values of C1, C2 and CST. If CST were zero, XCST would be infinite, and the unknown impedance would be equal to RS + XS. To correct for XCST, the series combination of RS and XS is converted to the parallel equivalent circuit by means of the following equations:

$RP = (RS^2 + XS^2) / RS$

$XP = RPRS / XS.$

The parallel equivalent is shown in Fig 3C. Since XCST and XU are in parallel, XU = XCSTXU / (XCST – XU). RP and XU are the parallel equivalent resistance and reactance of the unknown impedance. Finally, RP and XU are converted to series to yield the series resistance and reactance of the unknown impedance.

BASIC COMPUTER PROGRAM

In all, two parallel-series conversions and one series-parallel conversion are required to calculate the unknown impedance. The most efficient way to carry out these calculations is with a computer program such as NOISE.BAS, listed in Table 1. This program is written in BASIC for use on an IBM® PC. Some of the statements may have to be modified slightly for use on other computers. Several REMARK statements are included to make the program reasonably self explanatory. The variable-capacitor and variable-resistor calibration data are entered in DATA statements in lines 520-720. These data are read by the program in lines 110-140. The program asks for "RESISTANCE DIAL READING," "CAPACITOR DIAL READING," "FREQUENCY," and "STRAY CAPACITANCE" in lines 190-205. Resistance and reactance dial readings are then converted into resistance and capacitance by means of table-lookup and interpolation in lines 200-230. Line 250 is used to load the value of C2. This is the value of C1 when it is set to the zero reactance point (50 on my reactance dial). The program then carries out the calculations outlined previously. The program prints out

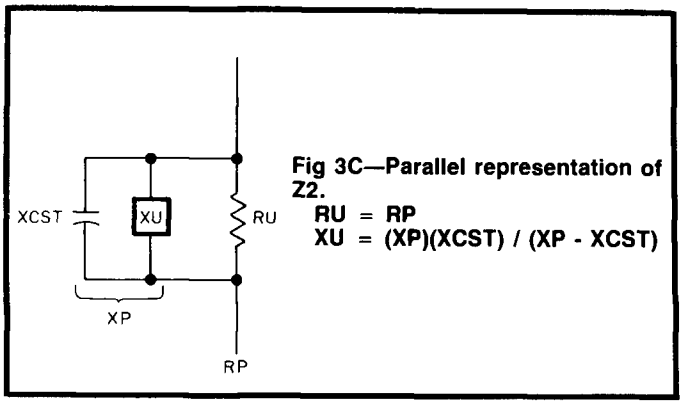


Fig 3C—Parallel representation of Z2.
RU = RP
XU = (XP)(XCST) / (XP - XCST)

TABLE 1

```
 5 REM PROGRAM NOISE.BAS
10 REM   -----NOISE BRIDGE CALCULATOR------
20 REM        F.H. SCHMIDT, K4VA
30 REM MARCH 13,1989
40 DEFDBL C,R,X,Z
50 DEFINT T
60 DIM R(100),C(100)
70 REM READ CAPACITOR CALIBRATION DATA
80 FOR J=3 TO 100
90 READ C(J)
100 NEXT J
110 REM READ POTENTIOMETER CALIBRATION DATA
120 FOR J=12 TO 100
130 READ R(J)
140 NEXT J
150 INPUT "RESISTANCE DIAL=";RD
160 IF RD<0 THEN 510
170 INPUT "REACTANCE DIAL=";XD
180 INPUT "FREQUENCY IN MHZ=";F
190 INPUT "STRAY CAP=";CST
200 T=INT(RD)
210 R=R(T)+(R(T+1)-R(T))*(RD-INT(RD))
220 T=INT(XD)
230 C=C(T)+(C(T+1)-C(T))*(XD-INT(XD))
240 XC=159154.9431#/(F*C)
250 CF=117.3#
260 XCF=159154.9431#/(F*CF)
270 REM CALCULATE XSST
280 XCST=159154.9431#/(F*CST)
290 RS=R/(1#+(R/XCST)^2#)
300 XSST=RS*R/XCST
310 REM XS=NET SERIES REACTANCE
320 XS=-(XSST+XC-XCF)
330 REM CONVERT RS AND XS TO PARALLEL FORM
340 RP=(RS^2#+XS^2#)/RS
350 XP=RP*RS/XS
360 REM REMOVE XCST
370 IF ABS(XP+XCST)<.001 THEN 430
380 XU=XP*XCST/(XP+XCST)
390 REM CONVERT TO SERIES FORM FOR FINAL RESULTS
400 R=RP/(1#+(RP/XU)^2#)
410 X=R*RP/XU
420 GOTO 450
430 R=RP
440 X=0
450 PRINT "RESISTANCE=";R;" OHMS"
460 PRINT "REACTANCE=";X;" OHMS"
470 PRINT "IMPEDANCE=";SQR(R^2#+X^2#);" OHMS"
```

```
480 PRINT "PHASE ANGLE=";ATN(X/R)*57.2957795#;" DEGREES"
490 PRINT:PRINT:PRINT:PRINT
500 GOTO 150
510 END
520 REM CAPACITOR CALIBRATION DATA
530 DATA 25.7,26.4,27.2,27.9,28.8,29.7,30.8,31.9
540 DATA 33,34.1,35.3,36.4,37.8,39,40.4,41.8,43.3,44.6
550 DATA 46.2,47.9,49.3,51.3,52.6,54.3,56,58.1,60,62
560 DATA 63.9,65.9,68.4,70.6,73.1,75.3,77.7,80.3,82.7,85.2
570 DATA 87.7,90.8,94.2,97.3,99.8,103.2,107,110,114.3,117.3
580 DATA 121.3,124.3,128.9,132.9,136.5,141.4,144.9,149.5,153.6,158.3
590 DATA 162.8,167.7,172.3,177.7,182.3,187,193,198,203,207
600 DATA 213,218,224,229,234,240,245,251,257,262
610 DATA 267,272,279,285,290,296,301.5,307,313,319
620 DATA 325,331,337,342,347,353,358,364,368,369
630 REM POTENTIOMETER CALIBRATION DATA
640 DATA 3.84,6.54,10.4,13.89,16.83,20.31,23.42,26.35,29.32
650 DATA 31.8,34.73,37.96,40.36,43.27,46.06,49.38,52.67,55.64,58.67
660 DATA 61.26,64.54,67.79,70.75,74.00,76.81,79.93,83.13,86.1,89.32
670 DATA 92.3,95.68,98.57,101.99,104.8,108.06,111.35,114.59,117.74,120.81
680 DATA 123.95,127.68,131.89,135.32,138.08,141.38,144.39,147.28,150.0,152.92
690 DATA 155.81,158.93,161.89,164.52,167.6,170.67,173.09,176.03,178.71,181.36
700 DATA 184.26,186.97,189.59,192.45,194.98,197.23,200.1,203.3,206.5,209.4
710 DATA 212.3,215.2,218.6,221.5,224.2,227.1,230.1,233.2,236.1,239.2
720 DATA 242.1,244.9,247.9,250.4,253.3,255.8,258.5,261.3,263.9,266.4
```

TABLE 2
TRI-BAND BEAM RESISTANCE MEASUREMENTS

Frequency MHz	*R(*) Ohms*	*R(**) Ohms*	*Ratio*
21.00	28.0	27.3	1.03
21.05	27.4	27.9	.98
21.10	28.7	27.9	1.03
21.15	30.9	30.9	1.00
21.20	34.7	35.3	.98
21.25	41.1	41.2	1.00
21.30	50.9	50.1	1.02
21.35	66.1	64.8	1.02
21.40	75.0	75.2	1.00
21.45	60.3	61.9	.97

*Measured with Bird wattmeter and calibrated RF ammeter.
**Measured with noise bridge.

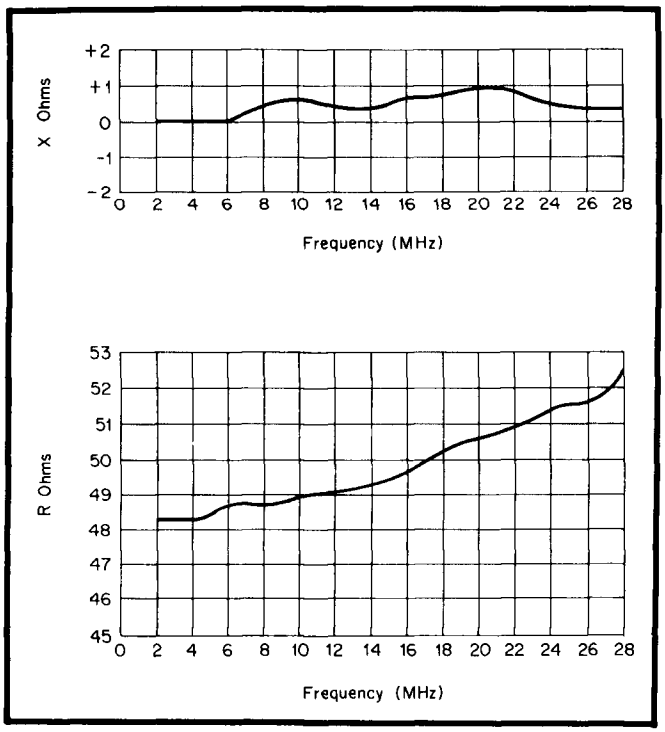


Fig 4—Resistance and reactance of MFJ-250 dummy antenna.

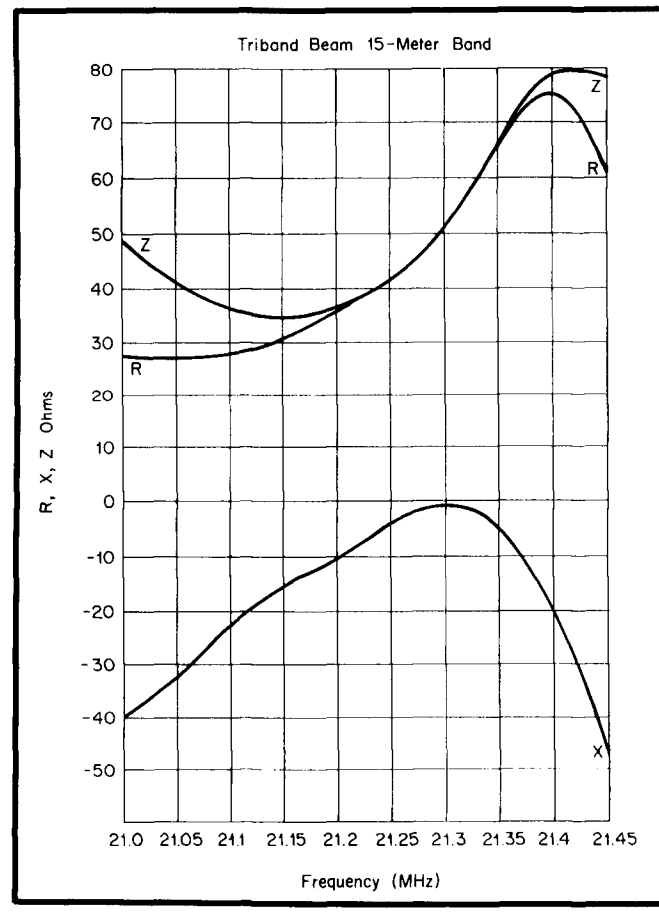


Fig 5—Resistance, reactance and impedance of Hy-Gain tri-band beam antenna fed with 150 ft RG-8/U on 15-meter band.

series resistance, series reactance, impedance, and phase angle. The reactance is preceded by a + or – sign to indicate inductive or capacitive reactance.

The value of "STRAY CAPACITANCE" must be found by trial and error. To determine CST, make up a dummy impedance of an approximately 100-Ω composition resistor in series with approximately 100 pF. Carefully measure the capacitor and the resistor after soldering. Use the shortest possible leads. Tune the receiver to 7 MHz, balance the bridge, and enter the dial readings into the program. Try different values of CST until the calculated values of R and X are as close as possible to the measured values. Once CST has been determined, program line 190 can be changed to:

190 CST = X.XX

where X.XX is the value just determined. CST will be approximately 6 pF, and will depend on the components used and the exact physical layout of the bridge.

ACCURACY

The noise bridge was calibrated with composition resistors in series with silver mica capacitors. The accuracy of the resistance measurement is better than 1 Ω. Reactance accuracy varies with frequency. It is about 1 Ω at 28 MHz and decreases to about 4 or 5 Ω at 1.8 MHz. This variation is related to the reactance tuning rate which varies at midscale from 1.4 ohms/division at 28 MHz to 23 ohms/division at 1.8 MHz. The reactance measurement accuracy can be improved by installing a 10:1 planetary drive and a turns counting dial. This arrangement would give a total of 500 divisions covering 180 degrees of capacitor rotation, or 1000 divisions if the dial is read to the nearest ½ division. The resistance dial rate is approximately 3 ohms/division and is independent of frequency. A turns counting dial and planetary drive would be helpful here also. Planetary drives are available from several sources, including RadioKit, PO Box 973-C, Pelham, NH 03076. Turns counting dials are listed in all the major electronics catalogs.

RESULTS

Fig 4 shows the resistance and reactance of an MFJ-250 VERSALOAD dummy antenna which was connected to the noise bridge through a one-foot length of RG-58A/U cable. The dc resistance is 48.15 Ω. The resistance gradually increases from 48.4 Ω at 4 MHz to 51.6 Ω at 28 MHz. The reactance is less than +1 Ω over the same frequency range.

Fig 5 is a plot of resistance, reactance and impedance of a Hy-Gain tri-band beam fed with 150 feet of RG-8/U coaxial cable at 21 MHz. The resonant frequency is 21.3 MHz. These curves change dramatically with beam orientation, humidity, temperature, absence or presence of foliage on nearby trees, etc.

As a final test of the noise bridge, I measured the beam resistance with a Bird wattmeter and a Weston 0-3 A RF ammeter and compared the results with the noise bridge measurements. I calibrated the RF ammeter against a true RMS digital ammeter at 60 Hz. The comparison is given in Table 2. The largest discrepancy is 3%.

These results suggest that the calibration with composition resistors and silver mica capacitors is reasonably accurate.

SUMMARY

The high accuracy noise bridge is easily constructed from readily available components. Calibration and alignment are straightforward, but do require a digital ohmmeter and a digital capacitance meter. The noise bridge and the accompanying program are a pleasure to use. Although this bridge is not in a class with commercially available laboratory instruments, it is an order of magnitude more accurate than presently available noise bridges.

More Ideas for a Robust HF Packet Signal Design

By Barry D. McLarnon, VE3JF
2696 Regina St
Ottawa, ON K2B 6Y1

I read Allan Kaplan's article on robust HF packet signal design with considerable interest[1]. There are some good ideas in his article, but I do not feel that they provide, as he terms it, "a skeleton of a robust HF packet scheme." In the following remarks I present some constructive criticism, and, more importantly, some alternatives which I believe could lead to a truly robust HF packet system.

It seems that just about everyone who contemplates improvement of amateur HF data communications seizes on forward error correction (FEC) as some sort of magic elixir which will cure what ails us. I don't buy it, especially when it involves adding 100% overhead to the transmitted information. I've said it before and I'll say it again: What HF packet needs is not FEC, but a well-designed ARQ system and better modems.[2] Look at it this way: The HF channel is a bursty channel, generally featuring periods of time with strong signals (and *potentially* error-free copy), punctuated by shorter periods with dense bursts of errors caused by fades. This is an ideal situation for ARQ, allowing us to push through the data with minimal coding overhead (error detection only) during the good periods, and repeating those packets which get hit by the bursts. Unfortunately, errors do occur during the good periods, and this is why the FEC proponents insist that their approach is needed. I submit, however, that a large proportion of these errors can be ascribed to inadequacies in the modulation scheme, and the way to eliminate them is to replace the modems, not apply the FEC bandage. Most people who have operated HF packet have experienced those instances when everything suddenly comes together, and the packets sail through one after another with no retries. This simply means that, along with the obvious factors like QRM abating and so forth, the multipath on that particular link has become low enough that the 300-baud modem can operate successfully. If the modem were improved, primarily by lowering the baud rate (not bit rate!), this would become a much less rare event. I have operated skywave HF data links at 75 or 100 baud which provided continuous data for periods of 10 minutes or more without a single bit error (obviously, there was no severe fading during these periods!). Under such conditions, FEC would just be burdensome overhead that greatly lowers the throughput of the data link. The point is, the more suited the modulation is to the channel, the better ARQ will perform, and the less attractive FEC will become. My advice, therefore, would be to improve the mode *first*, try it in an ARQ system, and then decide whether any FEC is warranted.

Instead of FEC with a fixed rate, I would much rather see a hybrid ARQ scheme in which the FEC check bits are only transmitted if errors are detected in the initial transmission of the data bits. This avoids the fixed overhead penalty when the channel is error free. The packet combining scheme discussed below is actually a simple form of this, using the trivial repeat code. It may be trivial, but it is simple to implement and is probably all that is really needed.

With regard to the particular approach to FEC advocated by Kaplan, he states that "With the exception of the four-tone FSK modems, it's all done in the TNC software." Is he claiming then that a Golay coder/decoder can be added to the TNC software? Not likely with the present generation of TNC hardware! He also excuses the overhead of the half-rate FEC code by claiming that "modulating two bits per channel symbol pays us back." This is not much of a payback since he has accommodated the overhead by doubling the occupied bandwidth! Maybe he was thinking of PSK, where you can indeed increase the bit rate (at the expense of reduced noise immunity) without paying a bandwidth penalty.

One of the more interesting aspects of Kaplan's proposal is the use of packet combining. This is an idea that I like very much, and have previously advocated (see Ref 2). His approach could use some refinement, however. One significant improvement would be to use "soft-decision" information, in which the analog value of each bit (ie, the amplitude of the eye opening at the sampling instant) is digitized and stored. Four or five bits of quantization is sufficient, so the A/D conversion hardware needed is very modest. With soft-decision information, there will obviously be more processing involved to compute bit-by-bit running averages instead of simple comparisons or majority votes, but the payoff will be more rapid convergence towards an error-free packet. For example, consider the case where a transmitted "1" bit is received incorrectly as a "0" on two successive transmissions of a packet. Suppose that a third copy of packet is then received in which that bit is correctly decoded as a "1" (but there are errors elsewhere in the packet, so it still fails the CRC check). A majority-vote scheme will still conclude that the bit is a "0", whereas with soft-decision information available, there is a good chance that a strong "1" will overcome the two previous weak "0" values and be correctly identified as a "1." This is due to the fact that if a bit is wiped out by noise, its eye value is statistically much more likely to be near the center of the eye than when the signal is strong. This is a heuristic argument, but there is abundant evidence in the coding literature which shows the benefit of using soft decisions in situations like this.

One thing that troubles me about packet combining is how to successfully apply it in a CSMA environment. If you are sharing a channel with other users who are using the

[1]Notes appear on page 10.

same type of packet structure, how do you avoid combining these undesired packets with the ones you want, and thereby destroying the information you're trying to extract? The answer probably lies in discarding any packets which differ significantly in the address fields from that which is expected. A possible refinement would be to protect the addresses and other vital parts of the packet header with FEC. It is also worth noting that since packet combining is a technique which can pull a weak signal out of the mud by averaging repeated transmissions, to get the most out of it you would have to remove the requirement that a signal be qualified by DCD in order to be decoded. On the other hand, the more stations sharing the channel, the more likely that the repeated weak packets will be clobbered by collisions and be unavailable for combining. This and other robust HF data strategems are best suited to point-to-point links on interference-free channels. In terms of the HF forwarding networks, this bespeaks a need for fewer BBS stations on a given channel, and good coordination of channel usage times.

The final comment I'd like to make on Kaplan's packet structure is that the 13-bit Barker sequence might be a little too short to use as a sync preamble. It would lead to a fairly high false sync rate (several false syncs per minute) on random data and hence degrade the combining effort. I would use a much longer sequence, say 24 bits, and use soft-decision information to do a maximum-likelihood (correlation) sync detector. Good sync sequences with correlation properties similar to Barker sequences have been found (by exhaustive computer search) for lengths up to at least 32 bits.

The modulation scheme proposed in the article, four-tone FSK with 300-Hz spacing, does nothing to address the problem of multipath-induced intersymbol interference (selective fading). The baud rate remains at 300, and hence the symbol period is still only 3.3 ms. Anytime the multipath spread exceeds a millisecond or so (and this will happen quite frequently, as detailed in Ref 2), the irreducible error rate will likely be too much for the EDAC to deal with. Packet combining may eventually pull something decodable together after several repeats, but this is a poor substitute for a system that could have gotten it right the first time. The point I'm trying to make here is that while coding and combining techniques are very useful, they are not an adequate substitute for a modulation technique having good resistance to multipath.

I believe that if the HF packet system is to become truly robust, the modem must operate at 100 baud (*not* to be confused with 100 bit/s!) or less. You may find higher baud rate HF modems in the commercial or military world, but they will likely fall into one of two categories: Either they will be very complex and have expensive serial modems with adaptive equalization, or they will be used in conjunction with ionospheric sounders by users with a high degree of frequency agility (guess who!). The sounder information and a large choice of frequencies can be used to avoid the worst multipath.

Kaplan proposes a 600-bit/s, 300-baud system using four FSK tones with 300-Hz spacing, for an occupied bandwidth of about 1200 Hz. Consider the following alternative: a 600-bit/s, 75-baud system using 16 tones with 75-Hz spacing, which again occupies a 1200-Hz bandwidth. What's the difference? In the first case, we have an MFSK system in which only one tone (out of a choice of four) is transmitted at a time. In the second, we have 8 tones transmitted at once, ie, 8 orthogonal binary FSK signals running in parallel. In return for greatly improved immunity to multipath intersymbol interference, we have traded off a somewhat reduced average power. For the HF channel, this is a highly beneficial trade off. Another possibility which makes more efficient use of the spectrum is to modulate all 16 tones with binary PSK and thereby produce a modem with a raw bit rate of 1200 bit/s. There are many other such possibilities, but the important element is to operate at a sufficiently low baud rate. The parallel form of such modems begs for the use of DSP techniques in their implementation.

Finally, we come to the method of encoding the data itself. Kaplan states, "for well over 90 percent of amateur communications, the Baudot code with upper-case letters plus the numbers and punctuation is all we really need to communicate." I object! My concept of the amateur packet network obviously differs from Mr Kaplan's, since I do not regard it as adequate to simply convey the essential information in a message to the other end. If I format a text file in a certain way, I want it to appear exactly the same way when it arrives at its destination. I would consider it unacceptable to have it rendered into upper-case (which appears to be SHOUTING!), or to lose useful control characters such as tabs, or to give up the other printable ASCII characters which are not available in Baudot code. HF packet is not merely an improved form of RTTY, it is part of an evolving network, and it should not impose any limitations on the network in terms of lack of data transparency.

A more forward-looking improvement to HF packet (and to the packet network in general) would be to ensure that the protocols were capable of handling binary data. Then the data, whether it be binary or text, could be compressed before transmission by means of the LZW algorithm, or some variant, as used in the popular archiving programs. These powerful compression techniques are routinely used in landline and satellite communications. Why not in packet radio, especially on HF, where the bandwidth and throughput limitations are much more severe?

Although this has largely been a critique of Ref 1, I hope it has also presented some new ideas (some of which were developed in more detail in Ref 2). I wish to thank Allan Kaplan for his thought-provoking article, and to *QEX* for providing a forum for discussion of issues related to HF data transmission. In my opinion, there has not been nearly enough such discussion amongst those who have some knowledge and experience in this area. I invite anyone who would like to continue the dialogue by more timely means to contact me via one of the e-mail addresses given below:

Internet: barry@dgbt.crc.dnd.ca
UUCP: . . . uunet!mitel!sce!cognos!dgbt!barry
Compuserve: 71470,3651
Packet: VE3JF@VE3JF

References

[1]Allan H. Kaplan, W1AEL, "Some Ideas for a Robust HF Packet Signal Design," *QEX*, November 1989, pp 5-7.
[2]Barry D. McLarnon, VE3JF, "New Directions in Amateur HF Data Transmission Systems - Part 1," *QEX*, December 1987, pp 3-9. (Part 2 in *QEX*, January 1988, pp 6-10).

Correspondence

Packet Error Control and LZW Compression

I enjoyed reading the article on possible methods for improvement for HF packet radio ("Some Ideas for Robust HF Packet Signal Design," November 1989 *QEX*). I feel however, I should voice some opinions and ideas of my own, now, rather than sit back and complain later, after it's too late.

There are two main points that I'd like to make. The first is just a thought on extending the use of Golay encoded words, to further accommodate error correction. The second has to do with improving AX.25 throughput, without using the loathsome Baudot code.

I like the idea of Golay encoding. But I think its beauty goes beyond merely providing a degree of correction. It breaks the data packet into sub-packet units. When uncorrectable errors are detected, I propose we consider retransmitting only the components that cannot be corrected. If the number of units to be retransmitted are small, this could be further extended to include the retransmitted components more than once, further enhancing the chance of success.

The general idea of data compression for better throughput is good, but let's not waste this opportunity on the Baudot code. Many modern modems provide compression/decompression on the fly, increasing their overall bit rates in that manner.

The AX.25 link is an end-to-end reliable link, where the data sequence is maintained at both ends, as the application sees it. If one logically inserts a compression algorithm between the PC application program and the TNC, then the transmitted traffic could be significantly less. What the traditional TNC sees here is still just plain old data (even though it is compressed data).

At the receiving end, the traditional TNC receives the compressed data, then feeds it to a decompression process, before delivering the final data to the application at the other end. Applications at both ends see no change, and the TNCs function as before, but with better throughput.

So the next generation of the TNCs could have the compression/decompression process built in at their front and back ends, but the AX.25 in the middle does not have to be concerned about this.

So what's better than Baudot? Several things come to mind here. I feel that the modified Lemple-Ziv Welch compression algorithm is ideal for a stream of data, and AX.25 fits this model perfectly. The algorithm requires no predefined tables, no handshakes, just adherence to table building rules. The maximum table size is practical, and the modified algorithm permits continued operation when the table gets full.

Compression occurs on the fly, and the table grows as it encounters new sequences of data that can be compressed. The clever part of the algorithm is that the receiver gets all it needs to know in order to build its own table on the fly, and decode the compressed incoming stream.

The modified algorithm was developed by Compuserve so that portable raster-scan image files could be kept and exchanged in compact form. A raster image in uncompressed form takes a lot of time to transfer on a modem. The file format is known to many as GIF files (pronounced "JIF").

So we now have a time tested algorithm to choose from, which has performed quite well in its use of transferring images. Due to the stream nature of AX.25, it seems like a natural application of this method.

The modified LZW algorithm is not expensive to implement in software either. There are a host of PC archive utilities that use LZW compression as just one of its compression schemes, and some of these programs are small in size. This leads me to believe that it could easily fit in a low-cost TNC.

The algorithm is a natural in a TNC because it would also increase the personal BBS storage capacity of a TNC's RAM. Text compresses quite nicely, and LZW could easily double storage capacity. This provides even more price/performance benefit.

However, as good as the algorithm is at this point in time, there will be other clever schemes in the future. We must provide for alternative methods if the future warrants it. Protocol ID or some other indication must be present to indicate the compression method chosen (or perhaps the lack of compression).

There is one down side to the use of LZW compression. Users who are just monitoring the channel will no longer be able to just "read the mail." This is because the TNC must build the table on the fly in identical fashion to the transmitting side. When monitoring, you might not get all of the packets needed to do this. It might be possible to pick up mid stream when the table gets flushed, but this will still present monitoring challenges, especially if traffic never causes a table flush. For this reason, call sign and control information must remain outside of the LZW realm.

Sometimes all that is needed to get the ball rolling is ready access to the information involved. So I've included a description of the modified LZW algorithm as I received it from the UNIX USENET network. I hope this helps. *—Warren W. Gay, VE3WWG, 3 Goldsmith Avenue, St Catherines, Ontario, Canada L2M 2V7.*

LZW and GIF Explained

I hope this little document will help enlighten those of you out there who want to know more about the Lempel-Ziv Welch compression algorithm, and, specifically, the implementation that GIF uses.

Before we start, here's a little terminology, for the purposes of this document:

character: a fundamental data element. In normal text files, this is just a single byte. In raster images, which is what we're interested in, it's an index that specifies the color of a given pixel. I'll refer to an arbitrary character as "K."

charstream: a stream of characters, as in a data file.

string: a number of continuous characters, anywhere from one to very many characters in length. I can specify an arbitrary string as "[. . .]K."

prefix: almost the same as a string, but with the implication

that a prefix immediately precedes a character, and a prefix can have a length of zero. So, a prefix and a character make up a string. I will refer to an arbitrary prefix as "[. . .]."

root: a single-character string. For most purposes, this is a character, but we may occasionally make a distinction. It is [. . .]K, where [. . .] is empty.

code: a number, specified by a known number of bits, which maps to a string.

codestream: the output stream of codes, as in the "raster data."

entry: a code and its string.

string table: a list of entries; usually, but not necessarily, unique.

That should be enough of that.

LZW is a way of compressing data that takes advantage of repetition of strings in the data. Since raster data usually contains a lot of this repetition, LZW is a good way of compressing and decompressing it.

For the moment, let's consider normal LZW encoding and decoding. GIF's variation on the concept is just an extension from there.

LZW manipulates three objects in both compression and decompression: the charstream, the codestream, and the string table. In compression, the charstream is the input and the codestream is the output. In decompression, the codestream is the input and the charstream is the output. The string table is a product of both compression and decompression, but is never passed from one to the other.

The first thing we do in LZW compression is initialize our string table. To do this, we need to choose a code size (how many bits) and know how many values our characters can possibly take. Let's say our code size is 12 bits, meaning we can store 0->FFF, or 4096 entries in our string table. Let's also say that we have 32 possible different characters. (This corresponds to, say, a picture in which there are 32 different colors possible for each pixel.) To initialize the table, we set code #0 to character #0, code #1 to character #1, and so on, until code #31 to character #32. Actually, we are specifying that each code from 0 to 31 maps to a root. There will be no more entries in the table that has this property.

Now we start compressing data. Let's first define something called the "current prefix." It's just a prefix that we'll store things in and compare things to now and then. I will refer to it as "[.c.]." Initially, the current prefix has nothing in it. Let's also define a "current string," which will be the current prefix plus the next character in the charstream. I will refer to the current string as "[.c.]K," where K is some character. OK, look at the first character in the charstream. Call it P. Make [.c.]P the current string. (At this point, of course, it's just the root P.) Now search through the string table to see if [.c.]P appears in it. Of course, it does now, because our string table is initialized to have all roots. So we don't do anything. Now make [.c.]P the current prefix. Look at the next character in the charstream. Call it Q. Add it to the current prefix to form [.c.]Q, the current string. Now search through the string table to see if [.c.]Q appears in it. In this case, of course, it doesn't. Aha! Now we get to do something. Add [.c.]Q (which is PQ in this case) to the string table for code #32, and output the code for [.c.] to the codestream. Now start over again with the current prefix being just the root P. Keep adding characters to [.c.] to form [.c.]K, until you can't find [.c.]K in the string table. Then output the code for [.c.] and add [.c.]K to the string table. In pseudo-code, the algorithm goes something like this:

```
[1] Initialize string table;
[2] [.c.] <- empty;
[3] K <- next character in charstream;
[4] Is [.c.]K in string table?
    (yes: [.c.] <- [.c.]K;
      go to [3]
    (no: add [.c.]K to the string table;
    output the code for [.c.] to the codestream;
      [.c.] <- K;
      go to [3];
    )
```

It's as simple as that! Of course, when you get to step [3] and there aren't any more characters left, you just output the code for [.c.] and throw the table away. You're done.

Wanna do an example? Let's pretend we have a four-character alphabet: A, B, C, D. The charstream looks like ABACABA. Let's compress it. First, we initialize our string table to: #0 = A, #1 = B, #2 = C, #3 = D. The first character is A, which is in the string table, so [.c.] becomes A. Next we get AB, which is not in the table, we output code #0 (for [.c.]), and add AB to the string table as code #4. [.c.] becomes B. Next we get [.c.]A = BA, which is not in the string table, so output code #1, and add BA in the string table as code #5. [.c.] becomes A. Next we get AC, which is not in the string table. Output code #0, and add AC to the string table as code #6. Now [.c.] becomes C. Next we get [.c.]A = CA, which is not in the table. Output #2 for C, and add CA to the table as code #7. Now [.c.] becomes A. Next we get AB, which IS in the string table, so [.c.] gets AB, and we look at ABA, which is not in the string table, so output the code for AB, which is #4, and add ABA to the string table as code #8. [.c.] becomes A. We can't get any more characters, so we just output #0 for the code for A, and we're done. So, the codestream is #0#1#0#2#4#0.

A few words (four) should be said here about efficiency: use a hashing strategy. The search through the string table can be computationally intensive, and some hashing is well worth the effort. Also, note that "straight LZW" compression runs the risk of overflowing the string table—getting to a code which can't be represented in the number of bits you've set aside for codes. There are several ways of dealing with this problem and GIF implements a very clever one, but we'll get to that.

An important thing to notice is that, at any point during the compression, if [. . .]K is in the string table, [. . .] is there also. This fact suggests an efficient method for storing strings in the table. Rather than store the entire string of Ks in the table, realize that any string can be expressed as a prefix plus a character: [. . .]K. If we're about to store [. . .]K in the table, we know that [. . .] is already there, so we can just store the code for [. . .] plus the final character K.

OK, that takes care of compression. Decompression is perhaps more difficult conceptually, but it is really easier to program.

Here's how it goes: We again have to start with an initialized string table. This table comes from what knowledge we have about the charstream that we will eventually get, like what possible values the characters can take.

In GIF files, this information is in the header as the number of possible pixel values. The beauty of LZW, though, is that this is all we need to know. We will build the rest of the string table as we decompress the codestream. The compression is done in such a way that we will never encounter a code in the codestream that we can't translate into a string.

We need to define something called a "current code," which I will refer to as "<code>," and an "old-code," which I will refer to as "<old>." To start things off, look at the first code. This is now <code>. This code will be in the initialized string table as the code for a root. Output the root to the charstream. Make this code the old-code <old>. Now look at the next code, and make it <code>. It is possible that this code will not be in the string table, but let's assume for now that it is. Output the string corresponding to <code> to the codestream. *Now find the first character in the string you just translated. Call this K. Add this to the prefix [...] generated by <old> to form a new string [...]K. Add this string [...]K to the string table, and set the old-code <old> to the current code <code>. Repeat from where I typed the asterisk, and you're all set. Read this paragraph again if you just skimmed it!!! Now let's consider the possibility that <code> is not in the string table. Think back to compression, and try to understand what happens when you have a string like P[...]P[...]PQ appear in the charstream. Suppose P[...] is already in the string table, but P[...]P is not in the string table. The compressor will parse out P[...], and find that P[...]P is not in the string table. It will output the code for P[...], and add P[...]P to the string table. Then it will get up to P[...]P for the next string, and find that P[...]P is in the table, as the code just added. So it will output the code for P[...]P if it finds that P[...]PQ is not in the table. The decompressor is always "one step behind" the compressor. When the decompressor sees the code for P[...]P, it will not have added that code to its string table yet because it needed the beginning character of P[...]P to add to the string for the last code, P[...], to form the code for P[...]P. However, when a decompressor finds a code that it doesn't know yet, it will always be the very next one to be added to the string table. So it can guess at what the string for the code should be, and, in fact, it will always be correct. If I am a decompressor, and I see code #124, and yet my string table has entries only up to code #123, I can figure out what code #124 must be, add it to my string table, and output the string. If code #123 generated the string, which I will refer to here as a prefix, [...], then code #124, in this special case, will be [...] plus the first character of [...]. So just add the first character of [...] to the end of itself. Not too bad. As an example (a very common one) of this special case, let's assume we have a raster image in which the first three pixels have the same color value. That is, my charstream looks like: QQQ... For the sake of argument, let's say we have 32 colors, and Q is color #12. The compressor will generate the code sequence 12,32,... (if you don't know why, take a minute to understand it). Remember that #32 is not in the initial table, which goes from #0 to #31. The decompressor will see #12 and translate it just fine as color Q. Then it will see #32 and not yet know what that means. But if it thinks about it long enough, it can figure out that QQ should be entry #32 in the table and QQ should be the next string output. So the decompression pseudo-code goes something like this:

```
[1] Initialize string table;
[2] get first code: <code>;
[3] output the string for <code> to the charstream;
[4] <old> = <code>;
[5] <code> <- next code in codestream;
[6] does <code> exist in the string table?
   (yes: output the string for <code> to the charstream;
     [...] <- translation for <old>;
     K <- first character of translation for <code>;
     add [...]K to the string table;
     <old> <- <code>;
   )
     (no: [...] <- translation for <old>;
     K <- first character of [...];
     output [...]K to charstream and add it to string
     table;
     <old> <- <code>
   )
[7] go to [5];
```

Again, when you get to step [5] and there are no more codes, you're finished. Outputting of strings, and finding of initial characters in strings are efficiency problems all to themselves, but I'm not going to suggest ways to do them here. Half the fun of programming is figuring these things out!

Now for the GIF variations on the theme. In part of the header of a GIF file, there is a field, in the Raster Data stream, called "code size." This is a very misleading name for the field, but we have to live with it. What it is really is the "root size." The actual size, in bits, of the compression codes actually changes during compression/decompression, and I will refer to that size here as the "compression size." The initial table is just the codes for all the roots, as usual, but two special codes are added on top of those. Suppose you have a "code size," which is usually the number of bits per pixel in the image, of N. If the number of bits/pixel is one, then N must be 2: the roots take up slots #0 and #1 in the initial table, and the two special codes will take up slots #4 and #5. In any other case, N is the number of bits per pixel, and the roots take up slots #0 through #(2**N - 1), and the special codes are (2**N) and (2**N + 1). The initial compression size will be N + 1 bits at a time to start code. If you're encoding, you output the codes (N + 1) bits at a time to start with, and if you're decoding, you grab (N + 1) bits from the codestream at a time. As for the special codes: <CC> or the clear code, is (2**N), and <EOI>, or end-of-information, is (2**N + 1). <CC> tells the compressor to reinitialize the string table, and to reset the compression size to (N + 1). <EOI> means there's no more in the codestream. If you're encoding or decoding, you should start adding things to the string table at <CC> + 2. If you're encoding, you should output <CC> as the very first code, and then whenever after that you reach code #4095 (hex FFF), because GIF does not allow compression size to be greater than 12 bits. If you're decoding, you should reinitialize your string table when you observe <CC>. The variable compression sizes are really no big deal. If you're encoding, you start with a compression size of (N + 1) bits, and, whenever you output the code (2**(compression size) - 1), you bump the

compression size up one bit. So the next code you output will be one bit longer. Remember that the largest compression size is 12 bits, corresponding to a code of 4095. If you get that far, you must output <CC> as the next code, and start over. If you're decoding, you must increase your compression size *AS SOON AS YOU* write entry #(2**(compression size) – 1) to the string table. The next code you READ will be one bit longer. Don't make the mistake of waiting until you need to add the code (2**compression size) to the table. You'll have already missed a bit from the last code. The packaging of codes into a bitstream for the raster data is also a potential stumbling block for the novice encoder or decoder. The lowest order bit in the code should coincide with the lowest available bit in the first available byte in the codestream. For example, if you're starting with 5-bit compression codes, and your first three codes are, say <abcde>, <fghij>, <klmno>, where e, j, and o are bit #0, then your codestream will start off like:

byte #0: hijabcde

byte #1: .klmnofg

So the differences between straight LZW and GIF LZW are: two additional special codes and variable compression sizes. If you understand LZW, and you understand those variations, you understand it all!

Just as sort of a PS, you may have noticed that a compressor has a little bit of flexibility at compression time. I specified a "greedy" approach to the compression, grabbing as many characters as possible before outputting codes. This is, in fact, the standard LZW way of doing things, and it will yield the best compression ratio. But there's no rule saying you can't stop anywhere along the line and just output the code for the current prefix, whether it's already in the table or not, and add that string plus the next character to the string table. There are various reasons for wanting to do this, especially if the strings get extremely long and make hashing difficult. If you need to, do it.

Hope this helps out.—*Steve Blackstock*

Bits

Propagation Note

Referring to September 1989 *QEX*, page 2, "Contributing to the Propagation Art." You might like to know that the Brazil (24,930) beacon comes in at this QTH almost all the time, even when 12 meters is otherwise "dead." (440 with 75-m vee wire and tuner.)—*Bill Sweet, KC9MB, Allen County, Indiana*

Corrigendum

Apologies!!

On page 13 of the January 1990 issue of *QEX*, there appears an article, entitled "HY-brid HI-Power," presumably written by Tom Pettis, KL7WE. Well, Tom is actually Tim. *Tim, please accept our apologies for the proofreading* oversight.

Components

By Mark Forbes, KC9C
PO Box 445
Rocklin, CA 95677

NATIONAL SEMICONDUCTOR LM1211

Because of the astonishingly rapid chain of developments in digital and microprocessor circuitry, analog and specifically RF circuits don't seem to get as much attention in the electronics press. But, the components for processing analog signals have developed at least as rapidly. Taking the development of analog RF ICs one step further is the National Semiconductor LM1211 single-chip IF detection system.

National's announcement calls the LM1211 the "first IF detection device to combine all the active components in one package." The part has improved bandwidth and dynamic range compared with earlier devices. External parts count can be as low six.

The system is built around a product detector, based on the Gilbert Multiplier, similar to the one found in National's LM1496/1596 balanced modulator/demodulator chips. Providing the other necessary functions for the IF system-on-a-chip are a 5-stage, high-gain IF amplifier, an adjustable-threshold AGC comparator, a detector and phase correction circuit, and a wideband output amplifier.

The IF amplifier is similar to the MC1350, however, the upper frequency limit is pushed to 80 MHz. The AGC comparator gives at least 40 dB of control to the IF amplifier. You can also adjust the attack/decay time with the AGC bias.

The detector and phase correction circuit allows compensation for both internal and external phase shifts. Correction for over 90 degrees is possible. Both AM and FM signals can benefit from the correction circuit.

The final amplifier is a 25-MHz bandwidth output amplifier, which allows it to accept data rates of up to 10 Mbit/s when used in communications applications. Its output can be either ac or dc coupled.

As you can imagine, the applications possible with this chip are extremely broad. It can obviously be used in AM, CW, SSB, or FM systems. It can also be used in data systems, such as in an FSK receiver, or in automation networks.

The LM1211 comes in a 20-pin plastic DIP package. Operating range is 20-80 MHz, although they claim that it should work down to 100 kHz for lower frequency applications and even LORAN-C reception. If you think this is a fantastic part with a lot of possibilities, wait until you hear the price: At quantities of 100, it's less than $4.00! If you're interested in buying a few of these, drop me a line; if we can get enough people together to get a good price, we'll order as a group. (Please don't send money!)

To obtain more information on this part, write National Semiconductor Corporation, 2900 Semiconductor Drive, Santa Clara, CA 95052.

BNC TO BANANA PLUG ADAPTERS

If you've ever needed to go from a BNC to a banana plug, you'll find this little kit interesting. ITT has a kit with six adapters to connect about anything. Included in the kit are BNC male and female to double banana plugs, BNC male and female to double binding posts, BNC male to male, and BNC female to female. If you'd like more information on this kit, contact: ITT, Pomona Electronics Division, Pomona, CA, phone 714-623-3463.

SPEAKING OF CONNECTORS...

The Heath Company has enhanced its connector kit, which allows you to make just about any kind of connector adapter. The connectors simply screw together to form any variation with the contents of the kit. Included are male and female versions of: N, F, RCA (phono), BNC, UHF (PL259/SO239), SMA, TNC, and mini UHF connectors. It's not exactly cheap, but if you've ever priced some of these adapters, you know they can run from $5.00 to $12.00 each. The kit comes in a zippered leather case, and sells for $79.95 (Heath HCA-3001). Contact Heath at PO Box 8589, Benton Harbor, MI 49022-8589, or call 800-253-0570.

SPDT WIDE FREQUENCY ANTENNA SWITCH

Sage Laboratories makes a nifty little antenna switch for use from dc all the way to 1 GHz. The self-contained switch has a single input, and switches between two possible outputs. It is capable of handling up to 200 W at 1 GHz, insertion loss is less than 0.5 dB and isolation between outputs is a minimum of 60 dB. It's tiny, at only 1.34- × 0.76- × 0.5-inch and comes with SMA female connectors. Contact Sage Laboratories, Natick, MA, phone 508-653-0844.

QUAD PIN DIODE DRIVER

The quad PIN diode driver contains four independent PIN diode drivers. Reverse bias is 50 V, and forward current is 300 mA. HIGH input voltage is 2 to 5.5 V, and LOW is 0.8 V or less. LOW input current is 1 mA, and HIGH input current is only 5 μA. For information on this product contact Silicon General, Garden Grove, CA, phone 714-898-8121.

30- to 1500-MHz FILTERS

The "Z" series of surface-mount filters operates over a frequency range of 30 MHz to 1.50 GHz, with up to six sections of complexity. Standard designs are Chebyshev or Butterworth; others, such as linear phase filters or Bessel functions are also available. 3 dB bandwidths from 2 to 150% can be specified with greater than 60 dB stopbands. The surface mounting allows it to occupy a very small amount of PCB space, and it has four lugs for enhanced grounding and mechanical integrity of mounting. To find out more, contact Trilithic Corporation, Indianapolis, IN, or phone 800-344-2412.

Gateway

Conducted by
Stan Horzepa, WA1LOU
75 Kreger Drive
Wolcott, CT 06716-2702

PROTOTYPE 905-MHZ DIGITAL RADIO COMPLETED

The first prototype 905-MHz "user" digital radio has been completed. The intent of the prototype phase was to arrive at a reproducible design using readily available components which could provide good performance at low to moderate cost. These radios are intended to provide moderate speed, 250 to 500 kbit/s, user access to an Amateur Radio network.

The prototype radio consists of four boards, PA, LO, RX and TX, mounted on a U-shaped chassis. This modularity was used to facilitate design; the pilot run design will be more integrated. The present effort involves laying out two PC boards, one for the PA and another containing the entire low-power radio, power and control circuits. It all appears to fit in a 5- x 7-inch enclosure with a top consisting entirely of a heat sink. Interface to the radio is differential ECL: TXD, RXD, RTS, DCD and S-meter output, antenna connector and 12 V dc @ 0.3A on receive and under 4A on transmit.

An attempt was made to produce local oscillator signals of good signal purity and stability. This was thought to be important since similar radios will probably be used in the near term at backbone IP switch sites and will need to operate in congested RF environments. There has also been some thought given to full-duplex operation of this hardware, but such operation is not being considered at this time. Dual outputs of +10 dB at approximately 96 MHz and 764 MHz are provided for transmitter and receiver mixing. Both output frequencies are derived from a single crystal oscillator. Spurious signals are below -80 dB.

Transmitter output power is in excess of +41 dB (12 watts) with modulation related sidebands down at least 70 dB (at 250 kbauds, modulation index equal to 1.4) at the edges of a 2-MHz wide, 905-MHz centered channel with out of band spurious performance also over 70 dB down (this excludes the second harmonic which is approximately -60 dB). Suppression of adjacent-channel interference is achieved by control of the deviation and an active filter ahead of the VCO tuning input. The VCO is presently free running at 45 MHz and doubly converted up to 905 MHz. This provides 905-MHz output frequency stability and accuracy essentially the same as that of the 45-MHz VCO alone. This performance seems acceptable, but alternative plans are being considered for providing complete crystal derived frequency accuracy along with fast transmit start-up and 0.5 Mbaud sorts of modulation, if necessary. The present architecture has the advantage of being easily adapted to transverter operation by a slight change of the crystal frequency and the substitution of a multimode HF transceiver for the 45-MHz VCO and receiver demodulator circuits.

The receiver has good BER down to below -90 dB. Turnaround time is better than expected. Since adding a little design effort to the switching circuits, it is now possible to square-wave modulate the RTS (PTT) line at rates of several kilohertz and generate a clean RF square wave at full power. In between these square-wave 12-watt output pulses, the receiver is able to demodulate a -90 dBm signal applied to the same antenna connector (with a 30-dB power attenuator inserted to protect the signal generator). The receiver is fully alive less than 40 microseconds after the falling edge of RTS. It is truly a "packet radar" and calculations show that with an 8-foot TVRO dish antenna or something similar, the radio should be able to copy its own packets reflecting off a 747 aircraft at a 4 to 10 miles distance. In normal use, the radio should easily be back on receive before the previously transmitted packet has reached a distant node.

The first pilot run is expected to be twelve radios and will undoubtedly reveal some areas where change can improve performance and reduce cost. Some thought is being given to a "one-board" lower power radio which can plug into a PC slot. To reduce cost, such a radio might use the phased-lock of an on-channel FSKed VCO instead of the dual conversion approach.

PC board design is presently in progress for the pilot run radios. It is anticipated that these will initially be used as backbone/switch radios as well as for user access radios as a part of the prototype Northern California moderate speed IP network now being constructed. Plans have not yet solidified for construction of additional radios, but it is expected that the next design effort will be aimed at providing similar radios for the digital allocations at 1249, 1251 and 1298 MHz.

by Glenn Elmore, N6GN

TEXNET-TNC 2 FIRMWARE ON THE HORIZON

Some of you may have heard rumors of the effort to transport the TexNet software to TNC 2s, which would support smaller "feeder" networks to link to the larger long-haul backbone networks, among other uses. This effort continues and second beta testing is underway and going very well. No releases are yet available.

from *Texas Packet Radio Society Quarterly Report*

MICROSAT SOFTWARE DEVELOPMENT REPORT

Some of you might be wondering what we (Harold Price, NK6K, and Bob McGwier, N4HY) are doing with the MICROSATs and why you can't connect up yet.

The development cycle for these launches was very short. Most

of the time available for software development, beyond getting the basic operating system and I/O drivers running, was in writing support code for ground-based hardware check-out. A fully operational CPU wasn't mated to a fully operational spacecraft until very late in the game. After that, much of the available software time went into qualifying the ROM-based bootloader. Given the short time available, the goals were to:

1) Check out all of the hardware on the ground.

2) Have the best chance possible of getting good telemetry as soon as possible once the spacecraft reached orbit.

This resulted in a limited-feature, but very reliable set of launch support software. This is the software that was running when the transmitter-on commands were sent.

Once the satellites were in orbit, several shortcomings in the high-level software loader and the battery management routines were seen. Bob and I have been working to fix these, in addition to commissioning the ground-stations which have never talked to MICROSATs before, and spacecraft which have never talked to the ground before, times 4. Once procedures are worked out, they are documented so that new command stations can be brought on line.

Here is a typical day in the life of Bob and Harold. Each day has been like this since launch (times approximate).

1200-1600 UTC: Bob wakes up, reads the mail from Harold for the previous night's passes, copies the early passes, checks the action of the software loaded last night, reconfigures the spacecraft as required, finishes testing the new command and control software for the next spacecraft, and sends it to Harold.

1500-1800 UTC: Harold wakes up, reads the mail from Bob, gets the new command and control software from Bob, integrates it with the new loader code, and tests it on the spacecraft CPU simulator.

1800-2000 UTC: Harold goes to TRW, starts the upload of the new software to the spacecraft (this is a bit tricky, which is why a new loader is being sent). The TRW ARC crew tapes or copies the other spacecraft in real-time. Harold reviews all of the telemetry for all of the spacecraft.

2200 UTC: Harold sends .EXE, which was partially uploaded to Bob to finish the uploading, sends a list of the diagnostic variables to the WEBERSAT ground-station crew to download on their next orbits or calls them to coordinate the use of uplinks to allow the WEBERSAT ground-station crew to access their experiments.

2200 UTC: Bob prepares and tests the new command and control software for the next spacecraft. Harold works on making the ground upload software distribution. The WEBERSAT ground-station crew tests the final camera code on the software simulator at Weber in preparation for it sending to Harold to run on the hardware simulator.

0100-0400 UTC: Bob completes the upload of the new software, prepares the source code for the next spacecraft for the start of the new cycle, answers fan mail, and calls Harold to confirm that the software is uploaded.

0400-0600 UTC: Harold goes back to TRW, holds his breath, tells the spacecraft to terminate the old program and start the new one, calls Bob to discuss the day, and gets a call from Weber on the diagnostic values. Bob goes to bed.

0600-0800 UTC: Harold and the TRW ARC crew tape or copy the other spacecraft in real-time and review all of the telemetry for all of the spacecraft. Harold puts the status message on Telemail.

1200 UTC: Bob wakes up and starts again.

Coming events: Harold and Bob polish their resumes as the above schedule leaves no time to pick up their pay checks, let alone do work.

Once the basic hardware is checked out and the batteries are in good shape, this frantic schedule will change. And once the loader and command software are tweaked for the last time, they will be turned over to the production command groups, hopefully by the beginning of February.

The next goal is to load the application software. Weber will be ready to do that in a few days. I have to take a business trip and more development needs to be done, which will delay the uploading of the first user-access software until late February. This will allow time for the final hardware check-out, which will include memory tests on the 8-Mbyte of mass memory, determination of the channel center frequencies, required uplink power, and the like.

by Harold Price, NK6K

SOFTWARE FOR THE PACKET-RADIO STATION

Lately, a lot of new and updated software for the packet-radio shack has become available. A synopsis of these programs follows.

APLink Version 3.94, the IBM PC AMTOR mailbox/packet-radio BBS software package, fixes a minor bug in the packet-radio log-off routine. The software is available on disk from TAPR.

Atari TCP/IP (KA9Q's NET) has been released by PE1CHL in a new version for the Atari computer. It is available directly from PE1CHL, however, TCP/IP fans in the US can obtain it by sending two double-sided or three single-sided disks, a self-addressed stamped disk mailer and a note stating what you want to Mike Curtis, WD6EHR, 7921 Wilkinson Ave, North Hollywood, CA 91605-2210.

from Julian Macassey, N6ARE

DOSgate Version 1.13 is now available. This version should be compatible with most TNCs and has been tested with the AEA and Kantronics TNCs. DOSgate is an IBM PC program that allows any remote packet-radio user to connect to and use a PC compatible computer as if the user was sitting in front of the PC (see *Gateway*, Volume 5, Number 23). To obtain a copy of DOSgate, send a 360-kbyte formatted 5-1/4-inch DOS disk, a self-addressed stamped disk mailer and your request for DOSgate to Rich Bono, NM1D, 7 Redfield Cir, Derry, NH 03038.

DRSI TCP/IP PC-Packet Driver is available in a new version that fixes all of the reported bugs in the earlier releases, now supports the NOS version of TCP/IP and marks the first release of the source code for the DRSI drivers. The pexecutables and DRSI driver source files may be downloaded from CompuServe's HamNet (file name: DR_NOS.ZIP) or may be obtained on disk from DRSI, 2965 Range Rd, Clearwater, FL 34625.

G4YFB Packet Mailbox Version 3.24 for the IBM PC and compatibles may be downloaded from CompuServe's HamNet (file name: YFB324.ZIP).

G8BPQ Packet Switch Version 3.53 has a number of changes and enhancements including a remote SYSOP feature for entering route/node information, node CTEXT for users accessing the switch via its alias, a BYE command and STATS information.

from Carl R. Finke, WB5DDP, via CompuServe's HamNet

KC8JN PBBS Version 8.21 for the Radio Shack TRS-80 Model I, III or IV (in the III mode) includes reverse-forwarding and BID store-and-forward. It is available by sending three unformatted 5-1/4-inch disks plus $2 (to cover mailing and documentation costs) to Bill Ritchie, N8FIS, 725 Rawson Ave, Fremont, OH 43420. Note that Bill is only able to copy the Model III disks because he does not own a Model I; anyone that wants to run this program on a Model I will have to convert it themselves.

MSYS Version 1.06, the IBM PC and compatible multiuser, multiport PBBS with support for TCP/IP, gateways and packet-radio clusters may be downloaded from CompuServe's HamNet.

Packet-Radio Satellite Tracking - The recent launches of amateur packet-radio satellites has generated a lot of interest among packeteers worldwide. In order to use the satellites, it is necessary to know when they will be "visible" to your station. To find out when satellite visibility occurs, the following satellite tracking software is available from the Radio Amateur Satellite Corporation (AMSAT) at PO Box 27, Washington, DC 20044, telephone 301-589-6062. Write or call AMSAT for pricing and ordering. Note that the software is discounted for AMSAT members.

"InstantTrack 1.00" for the IBM PC and compatibles was written by Franklin Antonio, N6NKF, and features speed, ease of use, instant visibility, real-time displays, automated orbital element entry and time-setting, satellite and station databases, a 1754 city database, grid square support, satellite co-visibility, squint angle, path loss, real-time rotor control, background mode, sun and moon tracking, fast rise-time finder, multiple station tracking and on-line help.

"Orbits II" is for IBM PCs and compatibles with 256-kbyte of RAM memory, CGA and compatible monitor, one 5-1/4-inch floppy diskette drive and DOS 2.0 or later.

"Orbits III" is functionally equivalent to ORBITS II, but has been designed for use with EGA and an Enhanced Color Display.

"Orbits IV" is the equivalent to ORBITS II, but has been designed for use with VGA.

"N4HY Quiktrak 4.0" is for IBM PCs with DOS 3.0 or later. An 8087 math co-processor is not needed, but the program will run much faster with it. CGA requires 512-kbyte of RAM, while EGA requires 640-kbyte of RAM.

"Quiktrak For IBM" is for IBM PCs without graphics capabilities.

"C-64 Supertrac" for the Commodore C-64, combines N4HY's Quiktrak program and enhanced graphics. It provides sharp, clear and colorful graphics with excellent scheduling flexibility. This program also has the ability to drive commercially available dual-axis rotors. It is available on disk only.

"Apple Quiktrak" is an Apple II menu-driven program for tracking and scheduling of amateur satellites. It incorporates a very fast algorithm for finding usable passes of the satellite of your choice. Satellite pointing angles relative to your QTH and a "window track" mode are included. The window track feature identifies mutual "windows" between your QTH and other specified locations around the world. It includes menu driven utilities for data entry and requires an 80-column card and 64-kbyte of RAM.

"Satellite Helper," written by W7HR for the Apple Macintosh, provides tabular data output of tracking and scheduling information for up to ten satellites. Graphic displays include rectangular, polar and great circle world maps, and there is a "view" mode which shows the Earth as seen from the satellite at any time. A real-time mode displays data as it changes and is compatible with the KLM/Mirage antenna tracking interface. Data can be displayed on the screen or printed on an Imagewriter printer. A propagation prediction package is also included which calculates the maximum usable frequency (MUF) to any point, sunrise and sunset times, bearings and distances, and displays the gray-line. Satellite Helper requires 1-Mbyte of RAM.

"C-128" is a version of the Orbits program that is written to take advantage of the unique capabilities of the Commodore C-128.

It uses time-based incrementing with user-selected increments of time, which results in uniform time increments between data lines. The program features automatic page formatting and pagination. As many as 20 satellites may be entered into Keplerian files. It requires an 80-column monitor and a Commodore compatible printer.

"Commodore Amiga" is designed for the Amiga operating in 80-column mode. It has the same features as C-128, but runs considerably faster. The program has provision for mode and phase programming to alert you as to which transponder is active. It requires 256-kbyte of RAM.

"Quiktrak for TRS-80 Model 4/TRSDOS 6.0" is the N4HY Quiktrak program modified to run on the TRS-80 Model 4.

"W3IWI Tracking Program" is available for the following computers: Tandy/TRS-80 Color Computer 1, 2 and 3 (cassette or disk); TRS-80 Model 3 (32-kbyte of RAM, one disk drive); Atari 400 and 800 (disk only); Each program provides all of the data needed for tracking satellites, space shuttles, etc in an easily understandable tabular form.

"HP-41 Programmable Calculator Orbit I" will output azimuth and elevation in real-time or all W3IWI parameters in the prediction mode. Special requirements: HP-41C plus QUAD Memory, HP-41CV, HP-41CX; Card Reader is desirable, but not essential.

"HP-41 Programmable Calculator Orbit II" is the same as Orbit I with the addition of functions from the Time Module for output of time and input from the time clock in a real-time mode.

PRMBS Version 1.05, the ROSE packet-radio mailbox and server for the IBM PC and compatibles fixes two major problems: the crash-inducing response to a non-standard "Link State is" message generated by a Kantronics KPC-4 and the failure to detect carrier loss of a landline modem connection in the "super-user state." The software may be downloaded from CompuServe's HamNet. The software is also available on disk from TAPR.

WØRLI Mailbox Version 11.8 removes support for MBL3.13, port types AEA and KAM, which is no longer required, and the line "?_name" from config.mb; fixes several cases of mail file corruption when GM. and several different TNC hang situations; and adds the V command (like T with FLOW ON). The software's file name is FS1108.EXE and it may be downloaded from N1EDF at 508-949-3590 or WA6RDH at 916-678-1535. The software is also available on disk from TAPR.

WA7MBL PBBS Version 5.13 adds limited support to hierarchical-addressed messages. It may be downloaded from CompuServe's HamNet or via anonymous ftp on Internet from either tomcat.gsfc.nasa.gov (\public\bbs\wa7mbl) or biocat.chem.usu.edu (\pub). Its file name is MBL513.ARC. The software is also available on disk from TAPR.

IP ADDRESS COORDINATOR LIST UPDATE

The following list, courtesy of Brian Kantor, WB6CYT, may be used to update the regional IP address coordinator list that was published in *Gateway*, Volume 6, Number 7. (An IP address is required in order to use the KA9Q Internet Protocol Package for amateur packet-radio TCP/IP operation. Contact your region's coordinator for an address assignment.)

IP LOCATION	IP ADDRESS	COORDINATOR
US Coordinators		
CA: San Bernardino and Riverside	44.018	KE6QH
DC	44.096	WA1IVD
ID	44.012	K7JD
KS	44.122	WIØR
MI: upper peninsula	44.092	KD9UU
MN	44.094	WDØHEB
NJ: southern	44.065	WB2MNF
NY: upstate	44.069	WA2WPI
OR: northwestern and Portland	44.116	WS7S
PR	44.126	KP4QG
WA: eastern	44.012	K7JD
WA: Vancouver	44.116	WS7S
WI	44.092	KD9UU
WY	44.086	WB7CJO
International Coordinators		
Chile	44.157	Flavio Llanos c/o CE6EZG
France	44.151	FC1BQP
Portugal	44.158	CT1DIA
Spain	44.133	EA4DQX

GATEWAY CONTRIBUTIONS

Submissions for publication in *Gateway* are welcome. You may submit material via the US mail or or electronically, via CompuServe to user ID *70645,247* or via Internet to *70645.247@compuserve.com*. Via telephone, your editor can be reached on evenings and weekends at 203-879-1348 and he can switch a modem on line to receive text at 300, 1200 or 2400 bit/s. (Personal messages may be sent to your *Gateway* editor via packet radio to WA1LOU @ N1DCS or IP address 44.88.0.14.)

The deadline for each installment of *Gateway* is the tenth day of the month preceding the issue date of *QEX*.