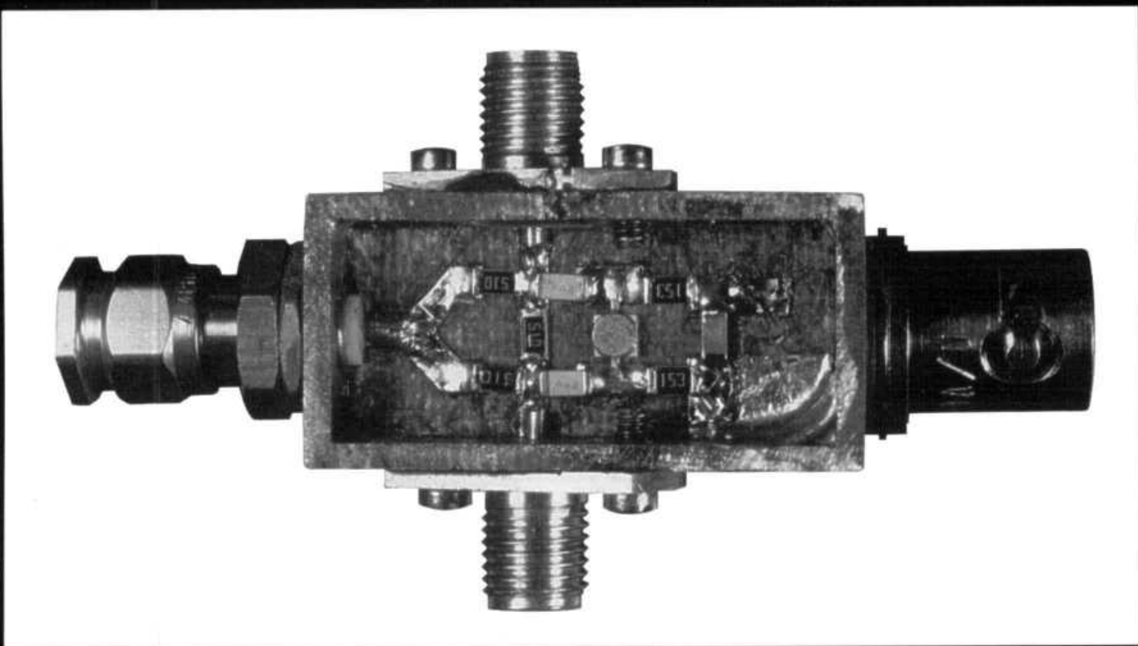
QEX

$1.75

ARRL Experimenter's Exchange

February 1995



Measuring SWR at UHF and Above

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

QEX (ISSN: 0886-8093 USPS 011-424) is published monthly by the American Radio Relay League, Newington, CT USA.

Second-class postage paid at Hartford, Connecticut and additional mailing offices.

Offices

225 Main St, Newington, CT 06111-1494 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: ARRL Member $12, nonmember $24;

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

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

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

QEX subscription orders, changes of address, and reports of missing or damaged copies may be marked: *QEX* Circulation. Postmaster: Form 3579 requested. Send change of address to: American Radio Relay League, 225 Main St, Newington, CT 06111-1494.

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

About the Cover

N1BWT's low-power SWR bridge, usable up through about 3.5 GHz, makes system measurements simple.

ISSUE NO. 156

Features

Columns

February 1995 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)

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*.

Any opinions expressed in *QEX* are those of the authors, not necessarily those of the editor or the League. While we attempt to ensure that all articles are technically valid, authors are expected to defend their own material. Products mentioned in the text are included for your information; no endorsement is implied. The information is believed to be correct, but readers are cautioned to verify availability of the product before sending money to the vendor.

Empirically Speaking

Get the Net

Over the past months, you've read a number of references to the Internet in *QEX*. This month, for example, Harold Price mentions the newly accessible FCC web page. More and more, information that was formerly available only in printed form, if at all, is becoming available for real-time access via the Internet.

Not only is the availability of electronic information growing at a geometric rate, but the amount of person-to-person communication is expanding, too. Using mechanisms such as Usenet news groups, electronic mail reflectors and direct email, people are participating in shared endeavors as never before.

While these changes are true generally, they are specifically true of the amateur experimenter community. Those working on such efforts as TAPR's DSP-93 project, AMSAT's P3D, projects to develop new HF digital protocols and simulators—and just about any experimental activity you can think of within Amateur Radio—are choosing Internet-based communications as the tool of choice to keep in touch and on top of the latest developments. It's rapidly getting to the point—if it's not already there—where if you don't have access to the Internet, you just aren't in the front lines of amateur technical development.

ARRL participation in the Internet has grown too, from a UUCP mail connection of a couple of years ago to a full-time connection providing email-based information services. Our friends on the net have provided well-connected sites such as ftp.cs.buffalo.edu and oak.oakland.edu from which you can get ARRL files via anonymous FTP. And recently, thanks to the Boston ARC, an ARRL web page has been made available (http://www.acs.oakland.edu/barc/arrl.html).

Only a couple of years ago, acquiring Internet access meant laying out big bucks (big for an individual, anyway) or having a connected employer or school. That has changed, as the major on-line subscription services are quickly adding Internet connectivity to their lists of capabilities. Most all of the services provide email gateways to the Internet. Many provide access to Usenet news, FTP and TELNET connections. And as this issue of *QEX* was going to press, Prodigy announced the availability of World Wide Web access. The other services won't be far behind. Probably within months, all of the major on-line services will have a full suite of Internet applications in place. And the cost of using the on-line services has dropped over time, too.

The upshot is: on a cost/benefit basis, amateur experimenters who want to stay on top of the exciting developments in amateur technology can't afford *not* to be on the net. Are you?

This Month in *QEX*

If you don't have a way to measure SWR at UHF and microwave frequencies, you just have to take it on faith that the components of your system are matched. That isn't always a good idea! Paul Wade, N1BWT, provides a partial answer with "A UHF+ VSWR Bridge."

Peter Traneus Anderson, KC1HR, continues development of his digital receiver, described previously in *QEX*, by adding "A Simple CW Demodulator for the DDC-Based Receiver."

Another thing we often take on faith is that our receivers aren't unduly distorting received signals. It would be nice to know for sure, though, and Jon Bloom, KE3Z, chimes in this month to describe "Measuring System Response with DSP."

Finally, in his "Digital Communications" column this month, Harold Price, NK6K, describes the digital communication capabilities planned for AMSAT's upcoming P3D satellite. He also reports on a new way of finding out what's going on at the FCC—via the Internet.—*KE3Z, email: jbloom@arrl.org (Internet)*

A UHF+ VSWR Bridge

Being able to measure VSWR at UHF and microwave frequencies can aid those homebrew projects

By Paul Wade, N1BWT

Hams frequently need to measure VSWR. This is paramount for antennas, as it is the only parameter conveniently measured, but it is also useful in many homebrew projects.

Most of the VSWR meters we use for antennas are based on directional couplers and require a significant amount of power for a useful reading. Many electronic devices are intolerant of that much power, and sometimes the frequency involved is outside the ham bands where it is undesirable as well as illegal to use transmitter power levels.

A simple resistive VSWR bridge can give good results over a wide frequency range with milliwatts of power. I found a nice surplus unit for the 2 to 12-GHz range which works so well that I wanted one for lower frequencies. I saw W1AIM using a bridge to trim the phasing harness for his EME array, and he reminded me that Joe Reisert, W1JR, had described one some years ago.[1] I later visited Joe at Antennaco and saw the original unit still being used to test antennas.

The W1JR unit is easily built in a small aluminum box and works up to about 450 MHz. He pointed out that the technique could be extended to several GHz—just what I was looking for. I decided to build one as small as I could using inexpensive chip resistors and capacitors that are now commonly available.

The circuit, shown schematically in Fig 1, is quite simple. How it works is more obvious if it is redrawn as a Wheatstone bridge, Fig 2, with R1 equal to R3.[2] R2 is the reference load connected to J2, and R4 is the unknown impedance connected to J3; if they are exactly the same, then the voltage at each end of R5 is the same, so there is no detected output and the VSWR is 1.0. This condition is referred to as having the bridge balanced. Any difference between R3 and R4 unbalances the bridge and causes an output from the detector. Detected output is proportional to bridge imbalance: the higher the VSWR, the greater the output.

At high frequencies, all components have inductance and capacitance. Since we would like the bridge to operate at as high a frequency as is possible, it is important to minimize the L and C by making the bridge as small as practical, and then to keep it balanced by making it symmetrical.

The smallest robust box that I could think of was a slice of X-band waveguide. I sketched out the compo-

161 Center Road
Shirley, MA 01464

[1]Notes appear on page 5.

nent dimensions to see if they would fit—it was too tight for my fingers in WR-90 waveguide. K1LPS had a spare piece of WR-112, the next larger size waveguide, with inner dimensions of 0.5 inches by 1.12 inches. The parts are mounted on a small Teflon PC board with the etching pattern shown in Fig 3. Since all the lines are as short as possible, dimensions are not critical; on the original board, I cut out the pattern using a hobby knife. Note that layout and construction are symmetrical so that J2 and J3 are electrically identical and interchangeable.

Construction is straightforward. The tinned PC board is trimmed to fit snugly in the waveguide, then the SMA connectors, J1, J2 and J3, are mounted and the center pins soldered to the board to hold it in place. Then the unit is inverted to solder the ground plane side to the inside of the waveguide. After applying plenty of flux, a ring of wire solder is fitted around the perimeter of the board, ready to flow into place as soon as it reaches the melting temperature. The outside of the waveguide is heated with a propane torch applied to the side (right side in Fig 5) that has no SMA connector until the solder melts and joins the PC board to the walls of the waveguide. Don't be timid with the torch—the idea is to get everything hot quickly and remove the heat as soon as the solder flows.

If waveguide is not readily available, a suitable enclosure may be fabricated from hobby brass, as described by NJ2L.[3] The pattern in Fig 3 extends past the waveguide dimensions to allow some flexibility in packaging.

After the flux is scrubbed off, components are soldered in place. Component placement is shown in Fig 4 and the photograph, Fig 5. A small wire passes through the board to the BNC connector and R8 underneath.

Operation of the bridge is also straightforward. A modulated RF signal is applied to J1, and the detected modulation, with amplitude proportional to VSWR, appears at J4. Usually the modulation is at 1 kHz, and the detected output is connected to a surplus SWR meter, such as the HP 415. (The solid-state 415D and 415E are more stable than earlier models, but any of them work fine, as do similar instruments from several other manufacturers.) Since the SWR meter is just a tuned audio amplifier with a calibrated meter, any audio amplifier driving an output meter will do the job.

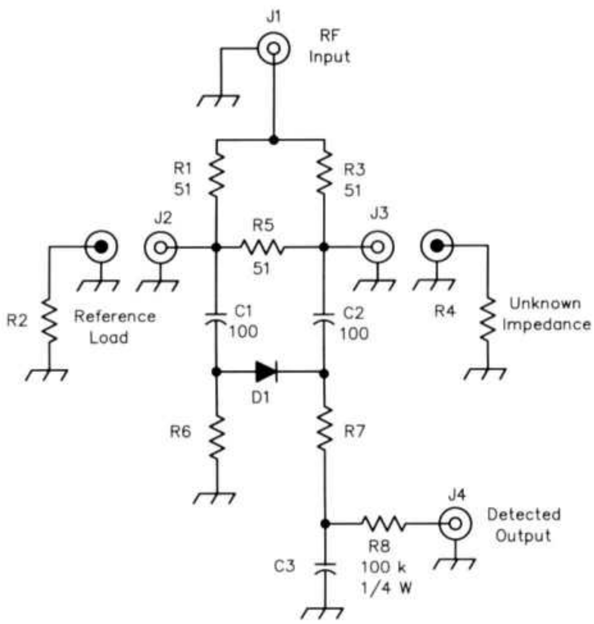


Fig 1—Schematic diagram of the VSWR bridge. All resistors and capacitors are chip devices except for R8. D1–Microwave mixer diode. R6, R7—10 to 15 kΩ.

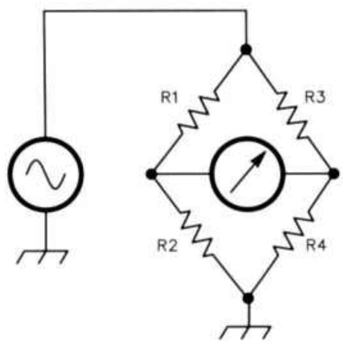


Fig 2—The Wheatstone bridge is the fundamental circuit within the instrument.

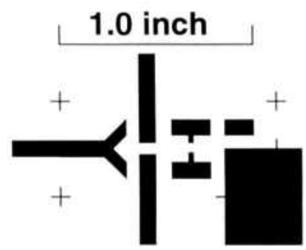


Fig 3—Etching pattern for the printed-circuit board. Use Teflon board and trim to fit the enclosure (1/32 inch recommended thickness).

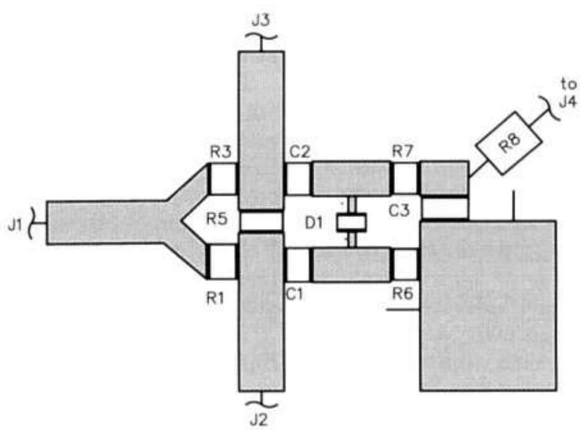


Fig 4—Component placement diagram for the VSWR bridge.

A good 50-Ω termination is connected at J2 as the reference load: the quality of the reference load is important since all other impedances are compared to it. In fact, we could measure VSWR on a 75-Ω cable by using a 75-Ω termination at J2.

To calibrate a measurement, connect a coaxial short circuit at J3 and adjust the SWR meter scale for infinite VSWR, or 100% power reflected. The unknown impedance is then connected at J3 to measure its VSWR. However, what a typical SWR meter such as the HP415 displays is return loss, the percentage of reflected power in dB. Most microwave engineers use return loss directly, but if you are more comfortable with VSWR, this is the equation for conversion (remember that *RL* is a negative number):

$$SWR = \frac{1 + 10^{\frac{RL}{20}}}{1 - 10^{\frac{RL}{20}}}$$

Since we have tried to keep the bridge balanced and symmetrical, a good test is to put good loads on both J2 and J3, then swap them. The measured VSWR should be low (high return loss) and identical in both cases.

Now that we've gone to the effort of using chip components and making the VSWR bridge as small as possible, how much have we gained? With good 50-Ω SMA terminations at J2 and J3, return loss was greater than 30 dB (VSWR < 1.2) from 10 MHz through 2304 MHz, while at 3456 MHz, the return loss was 22 dB (VSWR = 1.9), so performance is degraded but still usable. However, at 5760 MHz it was worthless. So we have increased the upper frequency limit to at least five times as high as the original version.

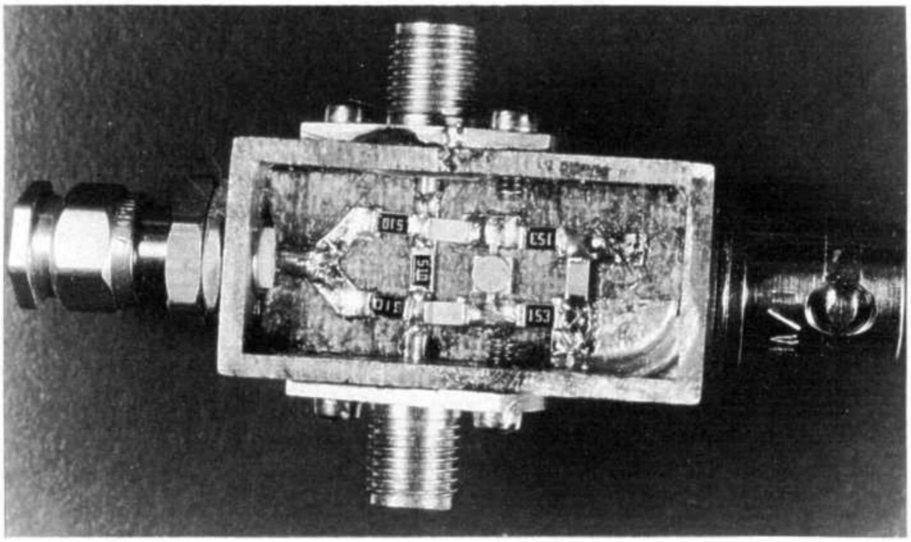


Fig 5—The completed bridge assembly.

It is possible that this style of VSWR bridge could be pushed even higher in frequency by making it even smaller, using tiny (and more expensive) microwave chip capacitors and resistors.

Notes

Parts and boards are available from:
DownEast Microwave
954 Rt 519
Frenchtown, NJ 08825
Tel: 908-996-3584

[1]Reisert, Joe, W1JR, "Matching Techniques for VHF/UHF Antennas," *ham radio*, July 1976, pp 50-56.

[2]Ryder, John D., *Networks, Lines and Fields*, Prentice-Hall, 1955, pp 32-37.

[3]Healy, Rus, NJ2L, "Building Enclosures for Microwave Circuits," *QEX*, June 1994, pp 15-17.

A Simple CW Demodulator for the DDC-Based Receiver

KC1HR's digital receiver is enhanced by an all-digital CW demodulator.

By Peter Traneus Anderson, KC1HR

As an Amateur Radio operator, I have always loved CW operation, even though I am not very good at copying code. Even the simplest Morse-decoding software does a better job than I do. I needed a CW demodulator to link my receiver output to a Morse-decoding computer.

A CW demodulator converts the receiver RF or audio to a logic level, recovering the on-off keying of the transmitter.

The simplest CW demodulator is a narrow band-pass filter followed by a rectifier and low-pass filter. This works well on strong signals with a high signal-to-noise ratio.

The DDC-based receiver already has an excellent band-pass filter built into the digital down converter (DDC). This filter is flat-topped with sharp skirts and linear phase response in the passband. In fact, the DDC was intended to drive a digital demodulator rather than the D/A converter I use to produce audio output.

As previously described, the DDC is programmed for passband widths of 2000 Hz and 400 Hz.[1,2] With its 25-MHz clock, the DDC is capable of narrower bandwidths, down to 107 Hz. Fig 1 shows the changes to the software of Fig 3 of Note 2 to add the option of 107-Hz bandwidth.

The 107-Hz bandwidth is not very useful for copying by ear, as the resulting audio passband is 42 to 149 Hz. To make the audio more audible, audio aliases through 550 Hz are allowed to pass through the post-D/A low-pass filter (Fig 7 of Note 1). Even so, the audio sounds like a dog's tail thumping on the floor: I feel it more than hear it.

The 107-Hz bandwidth really shows off the DDC filter: the –100-dB bandwidth is only 153 Hz. With this filter, signals are few and far between, even in the most crowded CW bands.

The only comparably sharp CW filter I have seen is a selective intermediate-frequency (IF) amplifier built by Dorothy Kaye, W6YIR, and John Kaye, W6SRY, in 1951.[3] Their IF amplifier operated at a frequency of 20 kHz using 12 toroidal inductors to achieve a flat-topped passband of 220 Hz at the 3 dB point, and a –100-dB bandwidth of 430 Hz.

Fig 2 shows the simple CW demodulator. This is a digital implementation of a full-wave rectifier followed by a threshold comparator and a low-pass filter.

The serial audio data from the DDC (U8 in Fig 5 of Note 1) is in 32-bit

990 Pine Street
Burlington, VT 05401
e-mail traneus@emba.uvm.edu

[1]Notes appear on page 10.

words, two's-complement format, with the most-significant bit (MSB) output first.

The first bit of each serial data word is the sign bit, which is captured by flip-flop U21A. Exclusive-OR (XOR) gate U20B multiplies the whole data word by the inversion of the sign bit: If the sign bit is one, the word is a negative number and the word passes through U20B unchanged. Thus, negative numbers are multiplied by +1.

If the sign bit is zero, the word is a positive number and U20B inverts every bit in the word. In two's-complement format, multiplying by −1 is accomplished by inverting every bit in the number and then adding 1 LSB (least-significant bit) to the result. I do not add the LSB, as an error of one part in 2^{31} is −187 dB, which is negligible. Thus, positive numbers are multiplied by −1, making them negative.

After this processing, the output of U20B is minus the absolute value of the original data word—it has been full-wave rectified. If there is no signal at all, the output of U20B is all ones. A tiny signal gives zero bits in U20B's output in the low-significant bit times. These are the bits sent last. As the signal grows, zeros appear earlier and earlier. Thus, an amplitude function is converted into a time function.

Flip-flop U21B and NAND gate U4C catch and hold zero output bits from U20B. U21B is cleared when the sign bit is captured. U21B stays cleared as long as U20B's output stays high. If U20B's output goes low, U21B is set, and remains set even if U20B's output goes high again.

Five bit times after the sign bit is captured, U23 is clocked. U23 is a fully synchronous four-bit shift register: all actions, even the parallel-load function, occur on the rising edge of the clock pulse.

U21B's output is applied to the mode-control input of U23. If U21B's output is zero (no signal), U23 is clocked in the shift mode. If U21B's output is one (signal present), U23 is clocked in the parallel-load mode.

The parallel-load data is all ones, and the serial shift-in data is a zero. Thus, the output of U23 (U23 pin 10) is one if there was a signal present at any of the last four times U23 was clocked. The output of U23 is zero only if there was no signal for any of the last four times U23 was clocked.

The output of U23 drives a transistor inverter. The output of the transistor is the desired logic output. An LED provides visual indication of the logic signal. The logic signal is high, and the LED dark, when there is no signal (key up). The logic signal is low, and the LED lit, when there is a signal above

```
case 'b':                    /* 2000 Hz bandwidth */
 fprintf(stdprn,"2100000001000000000000000000000000101001\n");
 fprintf(stdprn,"21010000011011111001111111011100011100101\n");
 fprintf(stdprn,"21100000000011101010100100100000001011010\n");
  break;
case 'n':                    /* 400 Hz bandwidth */
 fprintf(stdprn,"2100000001000000000000000000000000010011\n");
 fprintf(stdprn,"21010010001011100000101001100110100100101\n");
 fprintf(stdprn,"21100000000011101010100100100000111001000\n");
  break;
case 'm':                    /* 107 Hz bandwidth */
 fprintf(stdprn,"2100000001000000000000000000000000000001\n");
 fprintf(stdprn,"2101011111111111111110000000000000000101\n");
 fprintf(stdprn,"21100000000011101010100100100011010001111\n");
  break;
```

Fig 1—Software fragment to add 107-Hz bandwidth setting to the existing 2000 and 400-Hz bandwidth settings. Compare this to Fig 3 of Note 2.

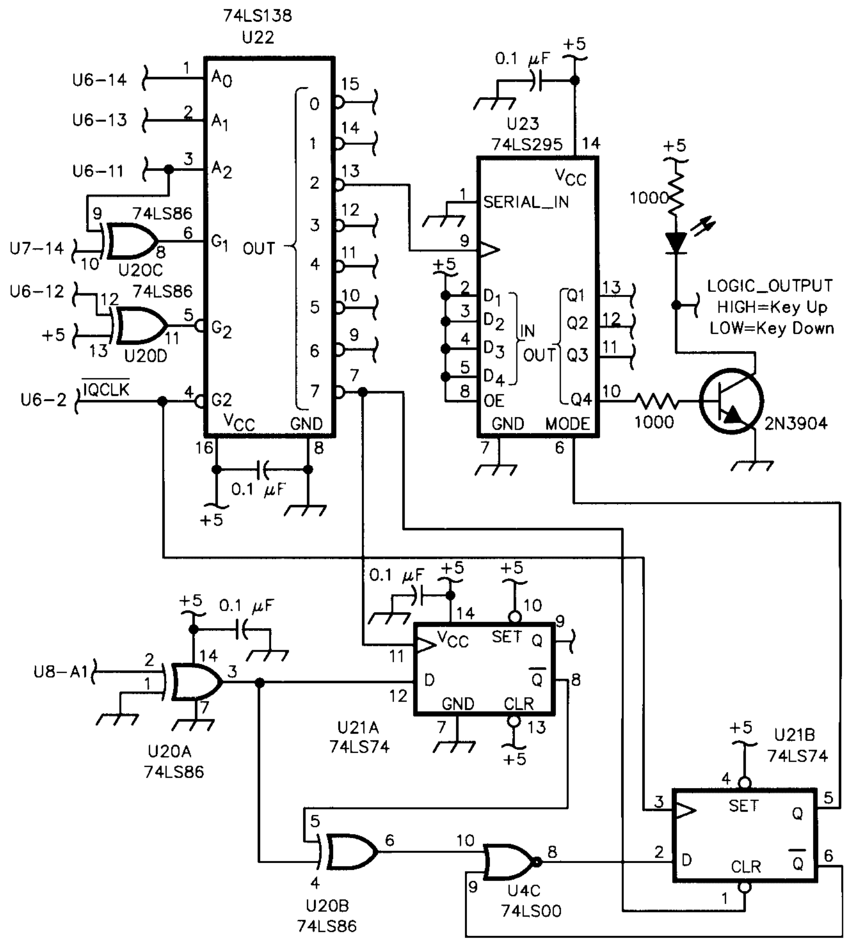


Fig 2—CW demodulator. The signals from U6 and U7 come from U6 and U7 in Fig 6 in Note 1. The signal from U8 comes from U8 in Fig 5 in Note 1. U4C is a spare gate from Note 1.

Drawings referred to in Notes 1 and 2, and in figure captions.

From March 1994 QEX, "A Simple SSB Receiver Using a Digital Down Converter," by P.T. Anderson, KC1HR.

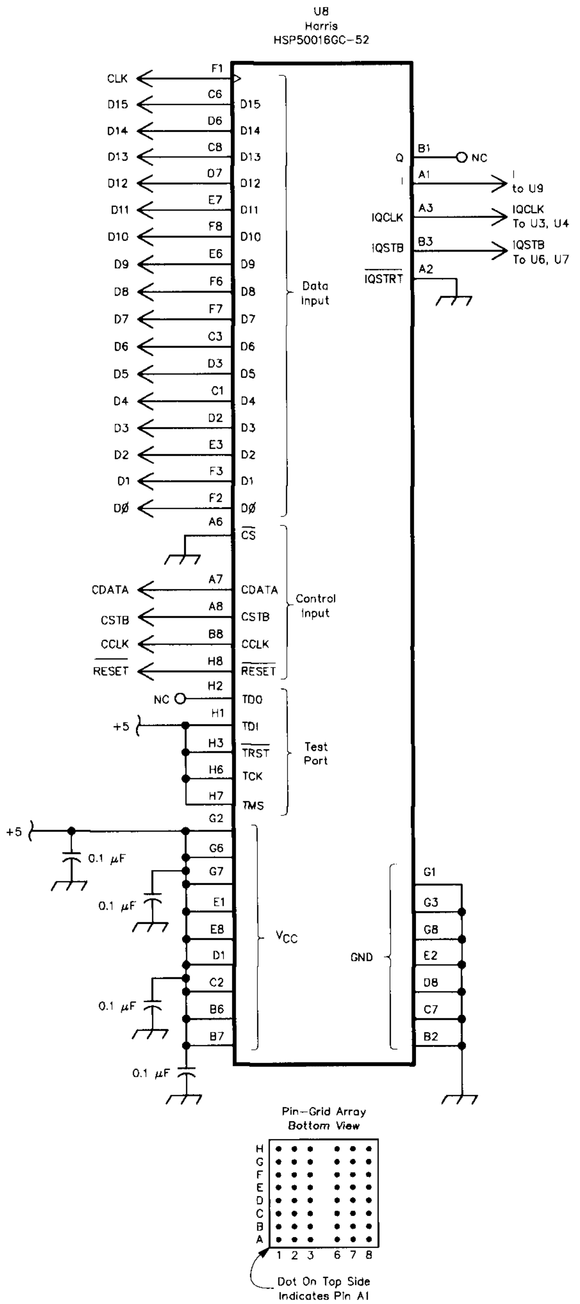


Fig 5

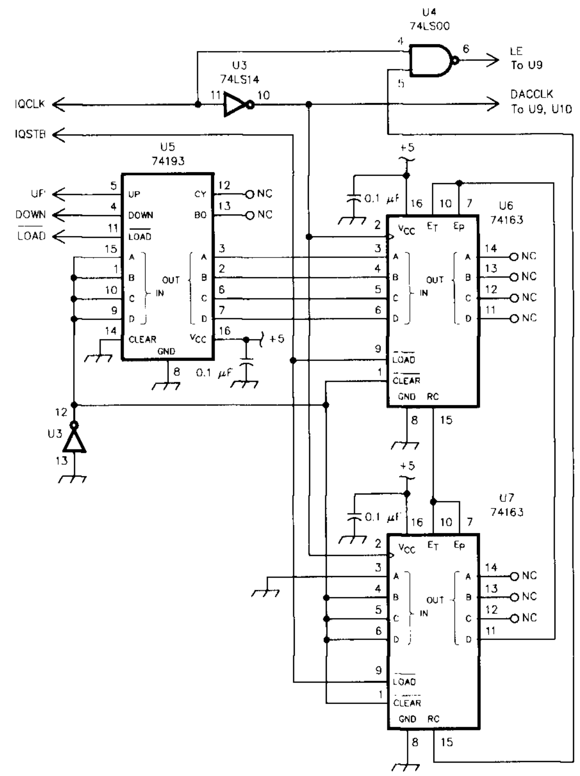


Fig 6

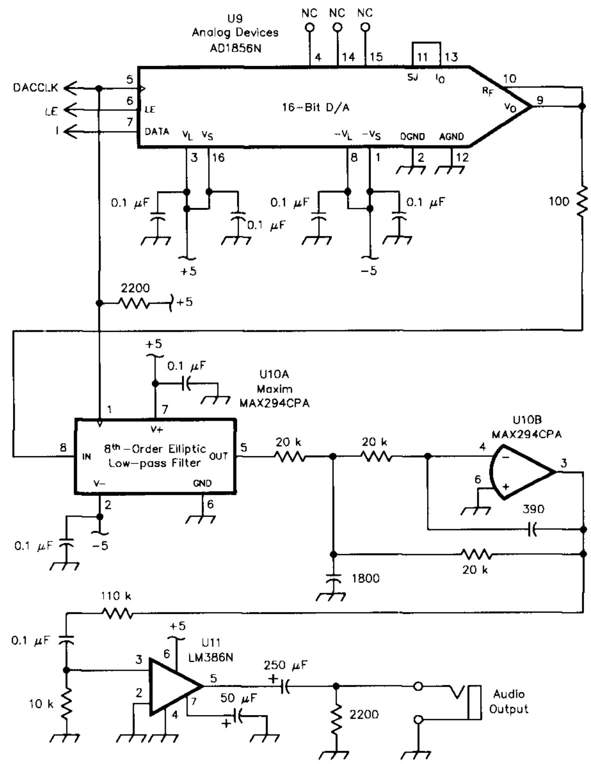


Fig 7

the threshold (key down).

The pulse-stretching action of U23 allows the demodulator to remember the presence of a signal when the signal goes through a zero-crossing (which the signal does twice per cycle). The pulse stretcher is a form of low-pass filter.

U23 is clocked at the DAC sample rate (382 Hz for the 107-Hz bandwidth setting). The stretched pulse should be at least half a cycle of the audio signal frequency. Thus the audio signal frequency should be at least one eighth the DAC sample rate (48 Hz for the 107-Hz bandwidth setting).

Signals should not be tuned to the low-audio-frequency edge of the pass-band (42 Hz for the 107-Hz bandwidth setting), because the pulse is not stretched quite enough to cover these frequencies.

Because U23 is clocked five bit times after the sign bit, the signal threshold

From November 1994 QEX, "A Better A/D and Software for the DDC-Based Receiver," by P.T. Anderson, KC1HR

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/* PC control program for 50016 receiver @ 25 MHz */
/* 20aug94 PTAnderson KC1HR */

main()
{
   int tempint = 0;
   int gain state = 0;
   int j = 0;
   int c = 'n';
   int exit char = '!';
   char preamble[] = "20011";
   char postamble[] = "00001";
   char ph inc string[] = "2001101100110010111010101000101111001000 1";
   long maxfreq = 12000000;
   long freq = 5000000;
   float tempfloat;
   double dfreq;
   double dsfreq;
   double dafreq;
   double dph inc;
   long ph inc;
   double fclock = 25000000.0;
   double two up32 over fclock;
   two up32 over fclock = 4294967296.0 / fclock;

/* initialize receiver to minimum gain */
   fprintf(stdprn,"@A\n");

/* initialize control registers 2 and 3 (same data for SSB or CW) */
   fprintf(stdprn,"2010000000000000000000000000000000000000\n");
   fprintf(stdprn,"2011000000000000000000000000000000000000\n");

/* print help table */
   printf("\nSimple SSB receiver set for lower sideband reception");
   printf("\n\ncenter of RF passband = frequency - 1.75 kHz");
   printf("\n\nexit to DOS:    set DDC clock kHz:     set frequency kHz:");
   printf("\n\n!                    #                      =");
   printf("\n\nup frequency:");
  printf("\n\n q=1MHz w=100kHz e=10kHz r=5kHz t=1kHz y=100Hz u=10Hz i=1Hz");
   printf("\n\ndown frequency:");
  printf("\n\n a=1MHz s=100kHz d=10kHz f=5kHz g=1kHz h=100Hz j=10Hz k=1Hz");
   printf("\n\ngain:                                        bandwidth Hz:");
   printf("\n\n   z=up   x=dn      c=min                   n=2000 m=400");
   printf("\n\nlast command, gain, alias, frequency in kHz now are:\n\n");

/* while not exit char */
   while(c-exit char){

     switch (c)
     {
       case 'z':              /* up gain */
   if( gain state > 14 )
     gain state = 15;
   else
   {
     gain state = gain state + 1;
     fprintf(stdprn,"C");
   }
   break;
       case 'x':              /* dn gain */
   if ( gain state < 2 )
   {
     gain state = 0;
     fprintf(stdprn,"A");
   }
   else
   {
     gain state = gain state - 1;
     fprintf(stdprn,"B");
   }
   break;
       case 'c':              /* min gain */
   gain state = 0;
   fprintf(stdprn,"A");
   break;
       case '=':              /* set freq in kHz*/
   printf("\rfreq kHz ");
   cscanf( "%f", &tempfloat );
   freq = 1000.0 * tempfloat;
   tempint = getch();
   break;
       case '#':              /* set fclock in kHz */
   printf("\rclock kHz ");
   cscanf( "%f", &tempfloat );
   fclock = 1000.0 * tempfloat;
   two up32 over fclock = 4294967296.0 / fclock;
   tempint = getch();
   break;
       case 'n':              /* 2000 Hz bandwidth */
   fprintf(stdprn,"2100000001000000000000000000000000101001\n");
   fprintf(stdprn,"2101000001101111100111111011100011100101\n");
   fprintf(stdprn,"2110000000001110101010010010000000101101 0\n");
   break;
       case 'm':              /* 400 Hz bandwidth */
   fprintf(stdprn,"2100000001000000000000000000000000010011\n");
   fprintf(stdprn,"2101001000101110000010100110011010010010 1\n");
   fprintf(stdprn,"2110000000001110101010010010000011100100 0\n");
   break;
       case 'i':              /* up 1Hz */
   freq = freq + 1 ;
   break;
       case 'k':              /* dn 1Hz */
   freq = freq - 1 ;
   break;
       case 'u':              /* up 10Hz */
   freq = freq + 10 ;
   break;
       case 'j':              /* dn 10Hz */
   freq = freq - 10 ;
   break;
       case 'y':              /* up 100Hz */
   freq = freq + 100 ;
   break;
       case 'h':              /* dn 100Hz */
   freq = freq - 100 ;
   break;
       case 't':              /* up 1kHz */
   freq = freq + 1000 ;
   break;
       case 'g':              /* dn 1kHz */
   freq = freq - 1000 ;
   break;
       case 'r':              /* up 5kHz */
   freq = freq + 5000 ;
   break;
       case 'f':              /* dn 5kHz */
   freq = freq - 5000 ;
   break;
       case 'e':              /* up 10kHz */
   freq = freq + 10000 ;
   break;
       case 'd':              /* dn 10kHz */
   freq = freq - 10000 ;
   break;
       case 'w':              /* up 100kHz */
   freq = freq + 100000 ;
   break;
       case 's':              /* dn 100kHz */
   freq = freq   100000 ;
   break;
       case 'q':              /* up 1MHz */
   freq = freq + 1000000 ;
   break;
       case 'a':              /* dn 1MHz */
   freq = freq   1000000 ;
   break;
       default:
         break;
     }
     if ( freq > maxfreq )
       freq = maxfreq;
     if ( freq < 10000 )
       freq = 10000;
     dfreq = freq;
     dsfreq = 0.001*dfreq;
     dafreq = 0.001*(fclock-dfreq);
     dph inc = (dfreq-1750) * two up32 over fclock;
     ph inc = dph inc;
     dph inc = ph inc;
    printf("\r%c %2.2u %9.3f %9.3f         ", c, gain state, dafreq,
dsfreq);
     strcpy( ph inc string, preamble );
     for( j = 0; j < 31; j = j + 1 )
     {
       ph inc = ph inc << 1;
       if( ph inc < 0 )
   strcat( ph inc string, "1" );
       else
   strcat( ph inc string, "0" );
     }
     strcat( ph inc string, postamble );
     for( j = 0; j < 41; j = j + 1 )
     {
       fprintf(stdprn,"%c", ph inc string[j]);
     }
     fprintf(stdprn,"\n");
     c = getch();
   }
 }
```

Fig 3

is one thirty-second of full scale.

U22 uses timing information from the digital gain setter in Fig 6 of Note 1 to derive the clock pulses needed by U21A, U21B, and U23. Because the pulses are derived from the digital gain setter, varying the gain setting varies the demodulating threshold in step with the DAC gain.

Thus the threshold is at a constant audio level rather than at a constant RF level. Software can vary the RF threshold level by varying the setting of the digital gain setter. Normally, I set the gain at the highest level that does not trigger on noise.

I could see no simple way of implementing digital AGC, so I did without it.

Fig 3 shows some copy from W1AW using this demodulator at the 107-Hz bandwidth. This copy was received using an ancient Morse-decoding program running on a Z80.[4] The copy was received on 80 meters in Burlington, Vermont, between 5 and 6 PM on December 29, 1994, using a small antenna. The signal was very strong.

Listings on page 120, December 1994 *QST*, show W1AW operating CW on 3581.5 kHz. The software for the receiver (Note 2) sets the center of the DDC passband to 1.75 kHz below the entered frequency. I entered the frequency as 3583.25 kHz and set the bandwidth to 107 Hz. W1AW was there without any frequency tweaking needed. The frequency accuracy of both the receiver and W1AW are impressive.

This simple CW demodulator works well on strong signals but is frustratingly miserable on weak signals. There are many signals that I can easily demodulate by ear, but that this demodulator cannot pull out of the noise, even at the 107-Hz bandwidth.

The solution, of course, is a better demodulator. A digital signal processor (DSP) is easily fast enough to implement very sophisticated algorithms.

When the DDC is set for the 107-Hz bandwidth, the sample rate is only 382 Hz. At this low sample rate, a 486 or faster PC is fast enough to apply many algorithms, without needing a separate DSP.

One algorithm of interest is the integrate-and-dump algorithm used for coherent CW (CCW).[5] A DDC-based receiver is ideal for CCW, where extreme frequency stability is needed. The DDC can directly provide the complex-signal (I and Q) components needed to drive the CCW gated integrators.[6]

```
+ QST DE W1AW HR ARRL CW100 FROM ARRL HQ NEWINGTON CT DECEMBER
21, 1994 TO ALL RADIO AMATEURS = TWO FORMER VOLUNTEER EXAMINERS
IN CALIFORNIA HAVE SURI ENDERED THEIR STATI ON AND OPERATOR
LICENSES IN AN EXAT INATION FRAUD CASE. A THIRD AMATEUR INVOLVED
IN THE CASE HAS AGREED TO A ONE YEAR OPERATOR LICENSE SUSPENSION
AND A 500 DOLLAR FINE. THE CASE STEMS FROM EXAMS TRONDUCTED IN
AUGUST 1993 AND AN FCC ORDER TO SHOW CAUSE AND SUSPENSION ORDER
IN SEPTEMBER 1994 + GL DE W1AW
```

Fig 3—Sample of W1AW copied in Burlington, Vermont with a receiver passband of 107 Hz centered on 3581.5 kHz.

For the transmitter oscillator, use a direct digital synthesizer (DDS) running from the same clock as the DDC. A counter can be used to derive the transmitter keying-rate pulse from the DDC clock. The resulting CCW transceiver would be entirely numerically controlled by software.

The CCW algorithm, as classically implemented, is optimized for narrow bandwidth, with a synchronous keying rate of 12 wpm. If the design keying rate is increased to 100 wpm, the CCW algorithm can be used to demodulate ordinary asynchronous CW at speeds of up to 30 wpm, without excessive dit-width distortion due to the CCW time quantization.

The CCW algorithm is optimal for pulling a CW signal out of random noise. The effective noise bandwidth of the CCW algorithm is very narrow, but the response skirts are broad: the response falls off at only 6 dB per octave. This makes the CCW algorithm susceptible to interference from nearby CW signals, particularly if the keying rate is increased. The DDC's sharp-skirted band-pass filter solves this problem.

Another possibility is adaptive CCW, where the frequency and keying rate are varied to lock onto the received signal.[7]

The DDC-based receiver should be of particular interest to designers of DSP-based frequency-shift keyed (FSK) modems, for example the modem described by J. B. Forrer, KC7WW.[8] Below, I describe the naive method of demodulating FSK from the complex-signal DDC outputs:

Set the center of the DDC's passband halfway between the mark and space frequencies. Set the DDC's bandwidth to be equal to the shift (difference between mark and space frequencies) plus one to two times the baud rate. Program the DDC to output a complex signal, I and Q (Note 6). Apply I and Q to your favorite DSP. The DDC does all the filtering, the DSP just demodulates the FSK.

The complex signal will have a frequency whose magnitude is one half the shift between mark and space, whether the tone is mark or space. The data is in the sign of the frequency (ie, in the direction of rotation of the phasor vector): positive (counterclockwise) for mark, negative (clockwise) for space, for example.

The phase angle of the phasor can be calculated at any time as the arctangent of Q/I. Do the calculation for every sample of I and Q. Subtract the previous phase angle from the current phase angle (remember to properly handle angles around multiples of 90°).

If the difference is positive, the output is mark; if the difference is negative, the output is space. Note that you get a valid output for every sample of the I and Q inputs, with no need to filter the output.

This is one time DSPers may want to use a 486DX: the 486DX has an arctangent instruction.

This algorithm should work very well for narrow shifts, where the mark and space tones are phase-coherent and selective fading is not a problem.

Notes

[1]Anderson, P. T., "A Simple SSB Receiver Using a Digital Down Converter," *QEX*, Mar 1994, pp 3-7.

[2]Anderson, P. T., "A Better A/D and Software for the DDC-Based Receiver," *QEX*, Nov 1994, pp 11-15.

[3]Kaye, D. and Kaye, J., "One dB per Cycle!," *QST*, Nov 1951, pp 29-31,102,104.

[4]Carlstrom, R., "Designing with the 8080 Microprocessor, Part 4: A Typical Program," *Popular Electronics*, December 1981, pp 74-78.

[5]Rusgrove, J. and Woodward, G., ed., *The Radio Amateur's Handbook*, 58th Edition, ARRL 1981, pp 14-34 to 14-36.

[6]Bloom, J., "Negative Frequencies and Complex Signals," *QEX*, Sep 1994, pp 22-27.

[7]The 100-Hz time code on WWV is a good challenge for a CW demodulator: The time code at 5000.100 kHz is sandwiched between the WWV carrier at 5000.000 kHz and the audio sideband above 5000.200 kHz.

[8]Forrer, J. B., "An Adaptive HF DSP Modem for 100 and 200 Baud," *QEX*, Nov 1994, pp 3-10.

Measuring System Response with DSP

Measuring the amplitude and phase response of baseband or narrow-band systems becomes easier with the use of DSP.

By Jon Bloom, KE3Z

Recently, we in the ARRL Lab decided to develop a better approach to measuring the response of receivers. Previously, we measured the amplitude response using sine-wave generators and an RMS ac voltmeter. That works, of course, but is time consuming and doesn't provide any phase-response information. The system described here produces automated amplitude and phase response measurement from 0 to 7200 Hz using a low-cost DSP board.

The idea is to measure the pass-band response of a receiver. To do this, we need to generate an audio signal, use that signal to modulate a carrier in the appropriate manner, and then apply the modulated signal to the receiver. The output from the receiver should be a signal that is the same as the signal we generated. If the modulation process is perfect, any differences between the original signal and the detected signal are due to distortion in the receiver. (By distortion I mean linear distortion—amplitude or phase errors—rather than nonlinear distortion. We'll assume that any nonlinear distortion present is negligible.)

225 Main Street
Newington, CT 06111
email: jbloom@arrl.org

If we wanted only to find the amplitude response of the receiver at various frequencies, we could use as our audio signal a simple sine wave, varying it in frequency to measure as many points as we want. An audio voltmeter could be used to measure the output of the receiver. But if we want to also find the phase response of the receiver, which is important for digital communication systems, we need to come up with another technique. One way of measuring the phase response would be with an oscilloscope. Viewing the original signal and the received signal on a dual-trace oscilloscope would let us measure the delay between the generated and received signal. From that, we can easily compute the phase difference between the two signals:

$$\theta(f) = -2\pi df \qquad \text{Eq 1}$$

where d is the measured delay from the input signal to the received signal and f is the frequency of the sine wave. The resulting value of $\theta(f)$ is in radians. There are several problems with this technique, however. One is that if the phase shift is more than 2π radians (one cycle), the output sine wave will "lap" the input sine wave, and the measured phase shift will be off by 2π. Phase shifts of multiple cycles will result in measurements that are off by multiples of 2π. The second problem is that it is difficult to get much accuracy in a delay reading from the screen of an oscilloscope. We will see shortly why this is a concern. The final and most troubling problem occurs if we are measuring an SSB receiver. In this case, there will be some difference in frequency between the input signal and the received signal unless the receiver is tuned *exactly* to the carrier frequency, which is unlikely. In this case, it is quite impossible to measure the phase using an oscilloscope, as the time difference between the two displayed signals is constantly changing.

One other point should be made. Since the signal we are generating must first go through a modulator to create the signal applied to the receiver, we are measuring the phase shift caused by that part of the measurement system as well. Fortunately, we seldom are interested in the absolute phase shift, or delay. What we are more interested in is the *differential* delay: the difference in the delay of a signal at one frequency compared that at another frequency. When a modulated signal passes through the receiver, if some frequencies are delayed more than others the resulting output will be distorted. This is shown in Fig 1. If all of the frequencies of the signal are delayed by the same amount, the signal is undistorted. Note, though, that Eq 1 shows that with a constant delay, the phase shift will be different for each frequency. In fact, the phase shift will vary linearly with frequency. So, what we are looking for in a distortion-free system is a linear phase shift versus frequency, arising from a constant delay at all frequencies.

Suppose instead of measuring the delay at a particular frequency we instead measured the phase directly. We could then determine the delay from Eq 1. But recall that the phase shift we measure may be off by a multiple of 2π radians, which would make any delay calculation based on phase subject to large errors. We can correct for this by considering the measured phase response at two frequencies that are close together. In this case, we can subtract the two phase values as follows:

$$\theta_2 - \theta_1 = -2\pi d f_2 + 2\pi d f_1 = 2\pi d(f_1 - f_2) \qquad \text{Eq 2}$$

Solving for the delay gives:

$$d = \frac{\theta_2 - \theta_1}{2\pi(f_1 - f_2)} \qquad \text{Eq 3}$$

In Eq 3, an error in the phase values of a multiple of 2π will disappear when the phase values are subtracted. Unless, of course, the two values are off by different multiples of 2π. That's why we want to use frequency values that are close together: to minimize the likelihood that the two values will be off by different multiples of 2π. If we were analyzing a system purely mathematically, we'd use frequencies that are infinitesimally different in frequency, using calculus to deal with the infinitely small difference. Doing so would give us an equation like so:

$$d_g = \frac{d\theta}{df} \qquad \text{Eq 4}$$

and now we call the result *group delay*. (Here, the d on the right-hand side of the equation refers not to delay, but to the differential.) If the phase shift is linear, the group delay will be constant across the frequency span. Thus it is *variation* in group delay across the spectrum that indicates phase distortion in the system.

Measurement using DSP

The system described so far requires an audio sine-wave generator, audio voltmeter and an oscilloscope. We would prefer a simpler system, and such a system can be built using DSP. The DSP system can generate the required input signal and measure the signal coming out of the receiver, eliminating the need for other audio test equipment. Of course, we still will need a way of modulating a carrier with the generated signal.

Even with DSP, using a sine-wave signal is problematic, especially if we are measuring an SSB system. The system may change its characteristics during the time we are trying to measure it. In the case of the SSB system, this occurs because of the frequency error between the carrier and the receiver local oscillator. In a system that includes AGC, the leveling effect of the AGC may mask differences in amplitude response at different frequencies, as the AGC struggles to maintain a constant output level. What we need as an applied signal is one that contains energy at a number of frequencies, so we can measure the response at all of the frequencies simultaneously. The questions now become: what kind of signal has energy at a number of frequencies, what particular frequencies do we need to have present, and how do we make a simultaneous measurement of all of those frequencies?

To answer those questions, it's best to begin with the last one. The technique we'll use to make the measurements is the fast Fourier transform (FFT). This algorithm takes N samples of the incoming signal and computes the amplitude and phase at $N/2$ frequencies. The particular frequencies it calculates are at multiples of the sampling frequency (f_s) divided by N, up to half the sampling frequency. To give a valid result, the sampled signal should contain energy at only those particular frequencies. Energy at other frequencies will show up—

$f1(t) := \sin(2\cdot\pi\cdot 1000\cdot t)$	A 1000-Hz sine wave
$f2(t) := \sin(2\cdot\pi\cdot 1100\cdot t)$	An 1100-Hz sine wave
$x(t) := f1(t) + f2(t)$	The two sine waves delayed by the same amount (0)
$y(t) := f1(t) + f2(t + .0007)$	The 1100-Hz signal delayed by an additional 0.7 ms
$t := 0, .00005 .. .01$	

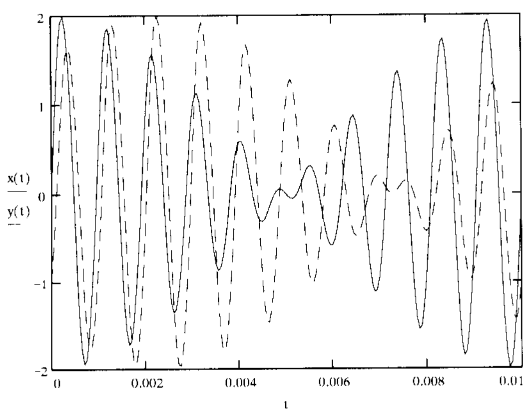
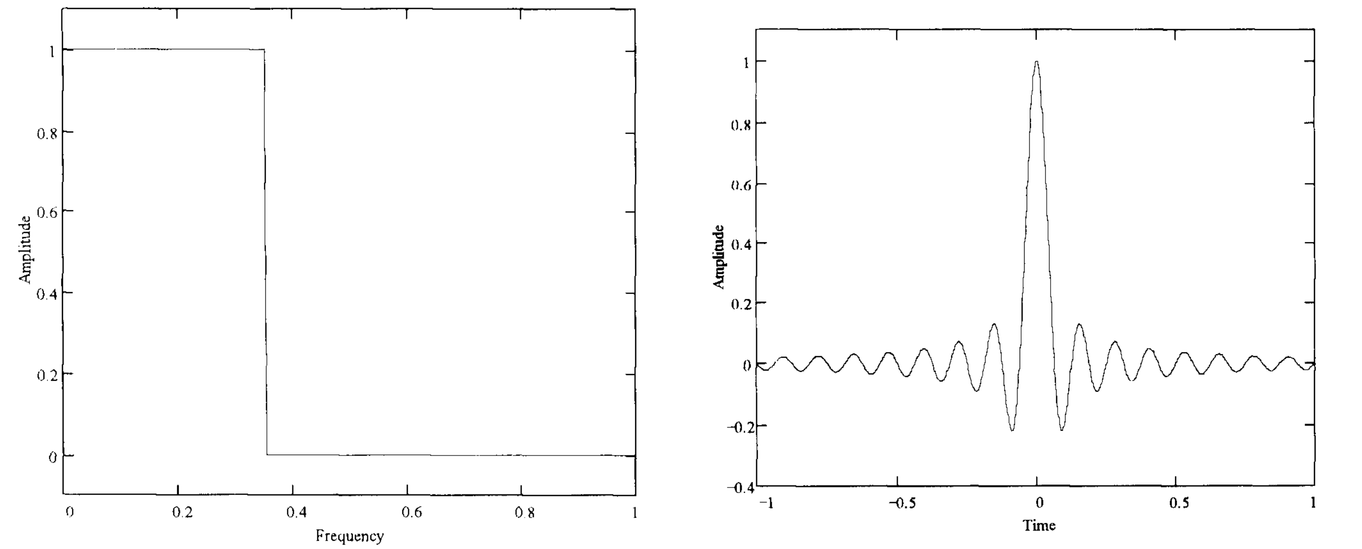


Fig 1—This *Mathcad* sheet shows the effect of phase distortion on a signal. The solid line shows a signal composed of two sine waves. The dashed line shows the same signal, but with one of the sine waves delayed more than the other. Note that the two signals are significantly dissimilar.

erroneously—in the calculated values at the particular frequencies. This phenomenon is called *spectral leakage*. Another way of stating this requirement is to say that the sampled signal must be periodic, with a frequency equal to f_s/N, and must have no energy at frequencies above one half the sampling frequency.

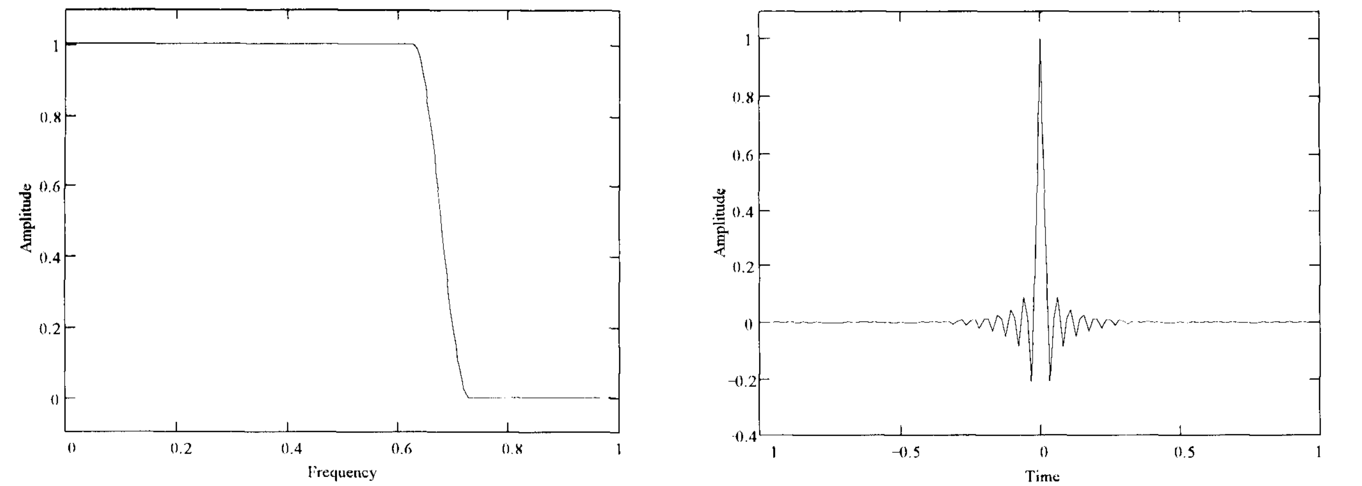
Now that we know what frequencies must be present in the signal, we can turn our attention to the question of how to generate such a signal. Ideally, our signal would have an equal amount of energy at all of the needed frequencies (including 0, or dc). A signal composed of a single, infinitely narrow pulse repeated at a rate of f_s/N would have an equal amount of energy at 0, f_s/N and all of the multiples of f_s/N. But there are two problems with this idea. First, how does one generate an infinitely narrow pulse? And second, we don't want energy at frequencies above $f_s/2$.

What we want is a pulse that contains only the frequencies of interest. If we could generate a pulse with energy at only the frequencies we're interested in, what would it look like? Fig 2A shows the spectrum we want, and Fig 2B shows the pulse that generates such a spectrum. At least, it's *part* of the pulse. The problem is, the pulse extends out to infinite time! Clearly, we can't generate a series of such pulses. Here we run up against a fundamental principle: a signal that is absolutely bandlimited—having no energy in its spectrum above some specific frequency—can't also be time limited. The reverse is true, too; a pulse that goes to zero at some point in time and stays there has a spectrum that goes to infinity. What to do? The answer is to generate a pulse that *essentially* has a zero spectrum above some frequency. That is, a pulse that, while it never goes quite to zero, is so



(A) (B)

Fig 2—An ideal signal for measuring system response would have a spectrum that extends from zero to the highest measurement frequency, shown at A. Unfortunately, the required waveform to generate such a spectrum, part of which is shown at B, requires an infinite amount of time.



(A) (B)

Fig 3—A better measurement signal has a flat spectrum (A) out to the highest measurement frequency, then rolls off, reaching zero at some point. Although the waveform corresponding to this spectrum is also infinitely long, it quickly reaches values very close to zero and stays there, allowing us to truncate the pulse while not significantly affecting the spectrum.

close to zero at all times after the main part of the pulse that its amplitude is "in the noise." Fig 3 shows the spectrum and waveform of such a pulse.

The spectrum of Fig 3A is flat out to a frequency we'll call f_p and reaches zero at a frequency f_b. Between those frequencies, it has the shape of a cosine curve. The waveform in Fig 3B dampens out to values close to zero. In practice, we will have to truncate the waveform in order to generate our series of pulses. That means the spectrum doesn't quite stay at zero amplitude above f_b, but the amplitude at frequencies above f_b is so small that we can ignore it.

We can calculate the sample values that will form this pulse once we select the sampling frequency (f_s), f_p, f_b and the number of samples we want to use to form the pulse, N. The sample values are:

$$y(n) = \left(\frac{\sin(2\pi f_0 t)}{2\pi f_0 t}\right)\left(\frac{\cos(2\pi f_\Delta t)}{1-(4 f_\Delta t)^2}\right) \qquad \text{Eq 5}$$

where:

$$f_0 = \frac{f_b + f_p}{2}$$

$$f_\Delta = \frac{f_b - f_p}{2}$$

$$t = \frac{2n - N - 1}{2 f_s}$$

and n runs from 0 to N-1. In the system described here, two frequency ranges are used. The high range sets f_p=7200 Hz and f_b=8400 Hz, at a sampling rate of 23041 Hz, while the low range sets f_p=1800 Hz and f_b=2200 Hz at a sampling rate of 5760 Hz. N ranges from 16 to 512, using integer powers of 2.

In practice, the spectrum of the signal we generate using Eq 5 will not be flat out to f_p due to sin$x$/x roll-off.[1] Fortunately, the system described here accounts for this amplitude roll-off during a calibration procedure, so we need not concern ourselves with this few-dB error.

The pulse we are generating contains a finite amount of energy that is spread across the spectrum of interest. As N increases, the repetition rate of the pulses drops, meaning there are more harmonics of the fundamental frequency contained in the spectrum up to $f_s/2$, so the energy of the pulse is divided into more signals. That is, the amount of energy in each harmonic will be less as N increases, so the signal-to-noise ratio at a given frequency will decrease. This is one of the fundamental limitations of this technique.

What we will feed into the FFT algorithm is a set of samples of the received signal covering one period of the pulse train. We will do all of our calculations on this single received pulse. It may at first seem that this means we need only to generate a single pulse through the system. But consider how a typical receiver will react to a single pulse: the AGC will attack that pulse, which takes time. Therefore, the gain of the receiver will be changed during the period of the pulse, distorting it. So, we need to generate a repetitive series of pulses, allowing the receiver to achieve its steady-state response before we measure one of the pulses. This requirement isn't limited to receivers. Many electronic circuits exhibit a marked *transient* response that could interfere with our measurement if we transmitted only a single pulse.

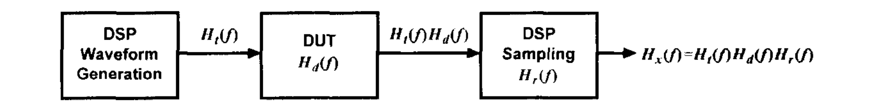


Fig 4—Block diagram of the simple measurement system. Each signal has a spectrum, and each device a frequency response, that can be written as a set of complex numbers, one complex number for each frequency.

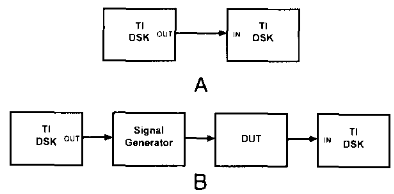


Fig 5—Calibration is performed using the system at A. The measurement system for a receiver (B) includes a signal generator, with some response, $H_g(f)$. The generator response isn't included in the calibration procedure, so it will be part of the measured response.

Implementation using the TI DSK

The system described here is implemented on the Texas Instruments TMS320C26 DSP Starter's Kit (DSK). This consists of a small PC board that includes a TMS320C26 DSP and a TLC32040 analog I/O chip, which includes A/D and D/A converters and programmable switched-capacitor filters on the audio input and output. The DSK also has an RS-232 interface chip to allow communication with a host computer. The TMS320C26 processor has an embedded ROM program that allows the host computer to download a program into the board. With 1568 words of RAM on-chip, the TMS320C26 can hold a significant amount of data and program. This application uses 512 words of the internal memory for program storage, 32 words for variable data and 1024 words for sampled pulse data: 512 transmitted samples and 512 received samples. This allows N to be as large as 512.

The DSP software consists of two components. The first component is a communication kernel that is loaded before the main application program. This kernel program is part of the DSK software package and was originally used with the DSK_SPEC program supplied with the DSK. A supplied loader program, DSKL.EXE, loads this kernel, which then participates in loading the application program and in providing serial I/O routines to that program.

Serial I/O between the DSK and the host computer is performed using software timing loops that require that the TMS320C26 interrupts be disabled during I/O. This means that analog I/O cannot occur during serial I/O, which impacts the design of the application program. The DSP program must be able to generate the pulse train described above and recover the sampled data from the receiver output, all without communicating with the host computer. Once all of the sampled data is in the system, the program must shut down the analog I/O, then dump the received data serially to the host.

The sequence of events that leads to our measurement results is a cooperative combination of processing on the DSP and the host computer. The sequence is:

- The host computer tells the DSP what to generate.
- The host computer tells the DSP to run the measurement signal.
- The DSP begins generating the pulse train.
- After some specified delay, the DSP stores N samples of the received signal.

[1]Notes appear on page 20.

• The DSP sends these N samples back to the host computer.
• The host performs the FFT and associated calculations and displays the result.

It may seem odd that the FFT is performed on the host computer rather than on the DSP, which can do it much faster. But there is no particular need for speedy calculation in this application. And adding FFT code to the DSP would leave that much less room for data, as some of the available memory would have to be used for the FFT code. Also, the TMS320C26 is a 16-bit fixed-point processor. It cannot compute the FFT with nearly the accuracy of the host program, which uses double-precision floating-point calculation.

One of the parameters the host passes to the DSP before the measurement is taken is the start-up delay. This specifies how many samples of the pulse train to generate before taking the measurement. Having this value supplied by the host allows for adjustment of the absolute phase of the signal. That is, the position of the received pulse in the N-sample measurement window can be adjusted by changing the start-up delay value and retaking the measurement. This proves valuable, as we'll see.

Calculating the System Response

As I've described, the DSP board does little more than generate the pulse train and sample the resulting signal from the receiver. It is the host processor that converts the sampled data into a system response measurement. It does so by applying the FFT algorithm to the samples received from the DSP. The FFT takes in N samples of the signal and produces N complex numbers. Each of these complex numbers describes the signal at a particular frequency, from $-f_s/2$ to $f_s/2$. Because the input samples are all real numbers, the FFT output values at the frequencies from $-f_s/2$ to 0 are just the complex conjugates of the values at the positive frequencies.[2] For that reason, we can ignore them; we are interested only in the values from 0 to $f_s/2$.

Each of the complex numbers computed by the FFT contains a real and an imaginary part, from which we can obtain the amplitude and phase of the signal at the corresponding frequency. The amplitude is computed thus:

$$A(n) = \sqrt{a_n^{\,2} + b_n^{\,2}}$$

where $a_n + jb_n$ is the complex value for frequency n/N. The phase is calculated by:

$$\theta(n) = \tan^{-1}\left(\frac{b_n}{a_n}\right)$$

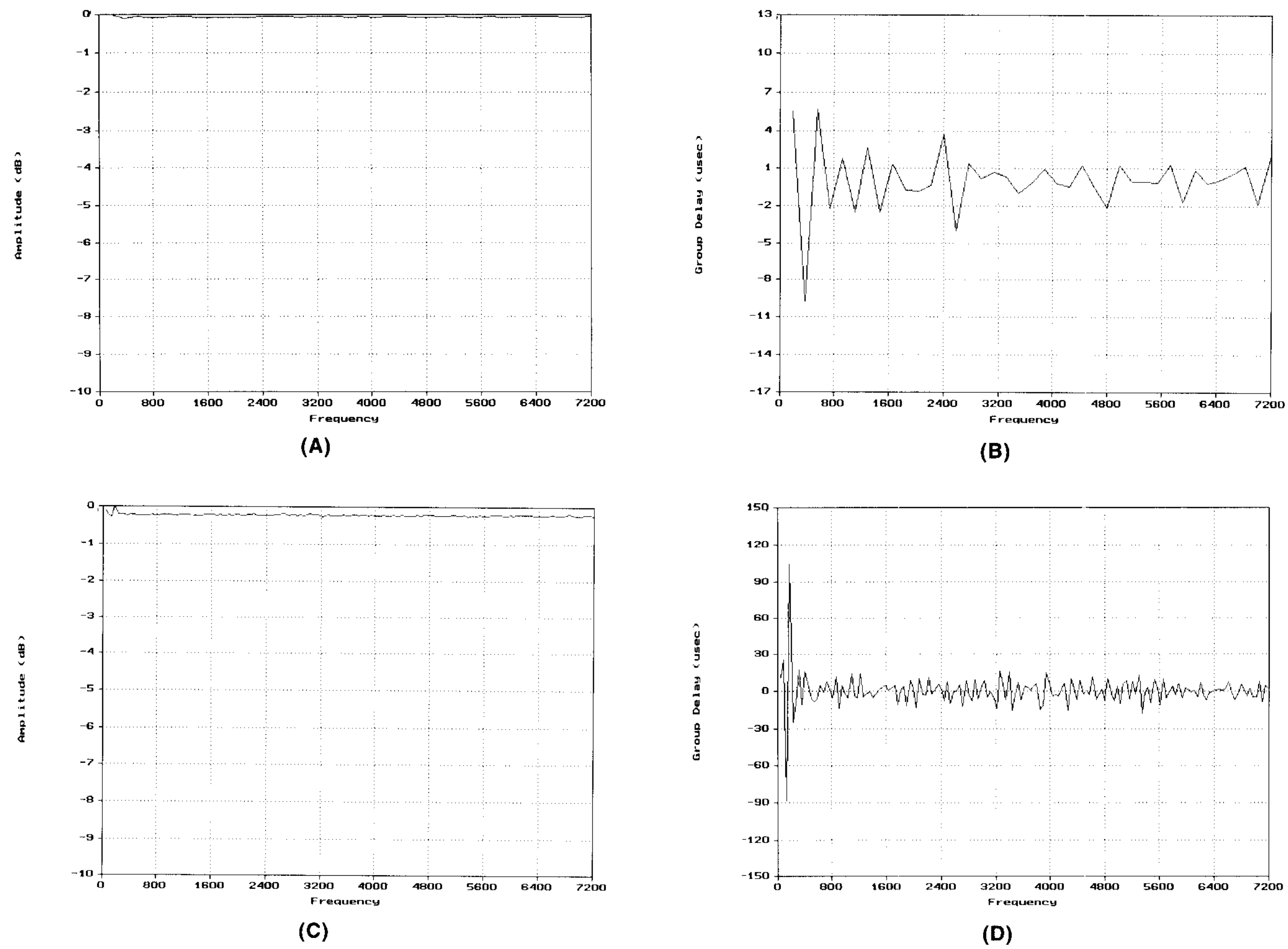


Fig 6—The number of samples used affects the measurement accuracy, as these measurements of the calibration signal show. At A and B are the amplitude and group-delay responses for a measurement of a single pulse (averaging=1) using 128 samples. C and D show the same measurement using 512 samples. (Note that the graphs of B and D use different vertical-axis scales.)

A four-quadrant arc tangent function must be used to get a proper result.

The FFT used in this program is a *radix-2* algorithm. This requires that the number of samples used be 2^M, where M is an integer of 2 or more. In practice, values of M from 6 to 9 are used, although the program allows values of M as low as 4.

The next issue to consider is the phase of the received signal relative to the signal applied to the receiver. While we are most interested in the relative delay between different frequencies, it would be nice to also be able to find the absolute delays through the receiver. We generated the signal in the DSP and sampled it there, too. Therefore, we could, in theory, know the absolute timing of the signal applied to the receiver. But the signal samples generated by the DSP require time to appear at the input of the receiver, and the signal from the receiver passes through the DSP system's input circuitry, which also adds delay. While we could probably analyze these delays, there is an easier way to account for most of them. This technique relies on some linear system theory.

Consider Fig 4. The signal coming out of the DSP system has an amplitude and phase at each of the frequencies present in the signal. Mathematically, we will call this spectrum of signals $H_t(f)$, where $H(f)$ is a complex number at each frequency. In fact, $H(f)$ is exactly what the FFT calculates from a sampled signal. This spectrum of signals is applied to the device under test (DUT), which has a particular response. This response can also be described as a set of similar complex numbers, $H_d(f)$. What is important here is that the output of the DUT at any frequency is mathematically simply $H_t(f)H_d(f)$. That is, if we knew the amplitude and phase of the input signal at a particular frequency, as a complex number, and the response of the system as a complex number at that same frequency, we could multiply the two complex numbers to find the result. Finally, this resulting signal passes through the DSP input circuitry, which has its own response, $H_r(f)$. The signal that actually gets sampled, $H_x(f)$, is thus the product of the original signal spectrum and the two responses:

$$H_x(f) = H_t(f)H_d(f)H_r(f) \qquad \text{Eq 6}$$

In our case, what we have measured is $H_x(f)$, and what we want to find is $H_d(f)$. We can calculate $H_d(f)$ by rearranging Eq 6:

$$H_d(f) = \frac{H_x(f)}{H_t(f)H_r(f)} \qquad \text{Eq 7}$$

But we don't actually know $H_t(f)$ because we haven't accounted for the delays in the DSP system generating $H_t(f)$, and we don't know $H_r(f)$ for similar reasons. So, what we need to do is to measure the values of $H_t(f)$ times $H_r(f)$. We

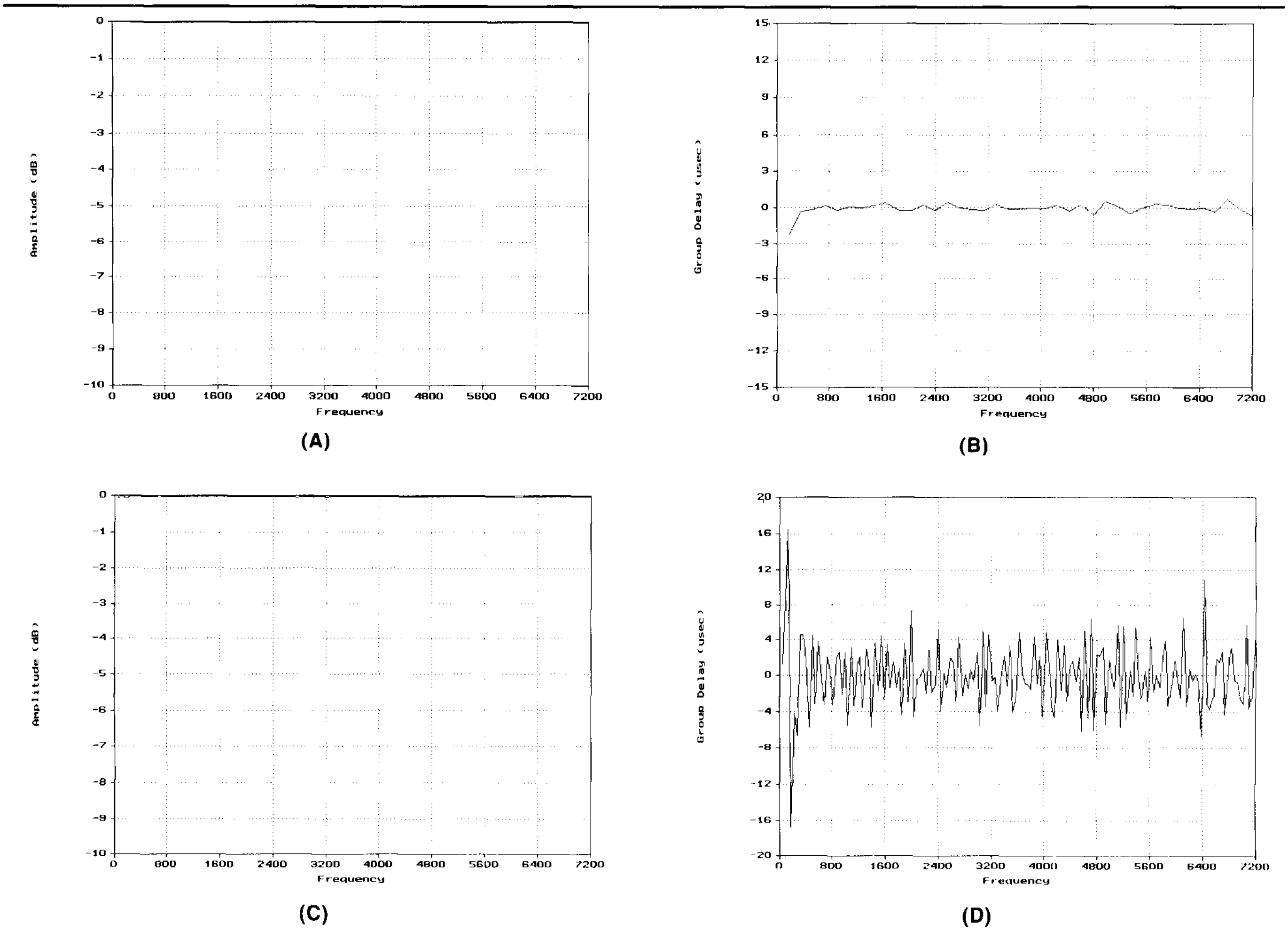


Fig 7—Averaging can improve the measurement accuracy. Here, the measurements of Fig 6 are repeated using an averaging factor of 10. Substantial improvement is noticeable.

do this by applying the DSP output signal directly to its input—without the device under test in the system. Then when we measure $H_x(f)$ with the device in the system, we can simply divide the complex number at each frequency of $H_x(f)$ by the measured and computed value of $H_t(f)H_r(f)$ for that frequency to get $H_d(f)$. This procedure not only accounts for the delays in generating and sampling the signals, it also accounts for the sin*x*/*x* roll-off.

A more complete system description is shown in Fig 5. The calibration measurement is shown in Fig 5A, the DUT measurement in Fig 5B. Note that the calibration procedure does not take into account the system response of the generator. Hopefully, the variation in amplitude and phase response of the generator will be negligible. It must, however, add at least some delay. If this delay value is constant and known, it can be subtracted from the measured delay of the receiver to get an absolute result. If the generator delay is constant but unknown, it won't affect the variation in group delay, which is our primary interest, but the absolute delay values will be in error. But if the delay of the generator is *not* constant, both the phase and group-delay measurements will be affected.

DSK Application Software

RESP, the program that runs on the DSK, is shown in Listing 1. This program is downloaded to the DSK and executed from the host PC by the DSKL utility. On exiting, the DSKL program leaves the DTR line of the serial port high. This is important, because if DTR goes low the DSK will be reset. From the time DSKL loads the program until we are finished using it, DTR must remain high at all times.

After initializing the processor and I/O ports, RESP enters a loop waiting for a command from the host computer via the serial port. Each command consists of an ASCII character. The supported commands allow the host to set the number of samples to transmit, the number to receive and the start-up delay. The list of sample values that make up the transmitted waveform is also sent by the host. Table 1 shows the commands and their parameters.

After setting the appropriate parameters, the host issues the G command, which tells RESP to begin signal processing. RESP then enables interrupts and enters an idling loop. At each sample clock, an interrupt is generated. RESP performs all of its signal processing during the servicing of these interrupts. At each sample interrupt, RESP outputs the next sample in the transmit sample list that was downloaded from the host computer. When it reaches the end of the list, it starts over, resulting in a repetitive waveform.

While sending the waveform, RESP also counts down the start-up delay, once count per sample. When the count reaches zero, RESP begins filling the receive sample buffer with incoming samples. Once the specified number of samples have been received and placed into the buffer, RESP sets the "resp" flag to indicate that the buffer is ready to be sent to the host.

The idle loop is quite simple. It simply waits for each interrupt to occur. Once the interrupt has been processed, the idle loop checks to see if the "resp" flag has been set. If so, interrupts are disabled and the received sample data is sent serially to the host computer. If "resp" is not set, the idle loop checks the state of the BIO pin of the processor. This is the serial-data input signal. If it's low, there is input coming from the host, so interrupts are disabled and RESP returns to the command-input mode.

This scheme allows RESP to be interrupted by the host during sampling, but it raises a potential problem. Since the serial input line is only tested once each sample clock, and since the serial input is done via software timing loops, by the time RESP realizes there is data coming in, part of the data has already been sent. RESP cannot properly receive the incoming data since it has missed part of it. For that reason, commands are sent via a three-step process. First, the host sends a character to "wake up" RESP. RESP responds by sending an R character to the host. Finally, the host sends the command character followed by any needed parameters. Not only does this ensure that RESP's software timing loop is synchronized with the host data, it also provides a positive response to the host so it can be sure RESP is responding to commands.

While the number of samples in the transmit waveform is usually the same as the number of receive samples, RESP doesn't require that they be the same. The only restrictions are that each of these counts is limited to a maximum of 512, and the transmit count must be at least 1. (The receive count may be zero, in which case the G command causes the transmit waveform to be sent continually until the host interrupts RESP by sending another command.) RESP could therefore be used for other purposes than that described here. For example, a simple sine-wave generator could be implemented in which the samples of a cycle of the sine wave are downloaded to RESP, the receive count is set to 0 and the G command is sent to generate the sine wave.

One notable feature of RESP is that it switches out the TLC32040's input antialiasing filter. It does so because this is a band-pass filter which, if left in line, would distort the measured response at low frequencies. For most measurements, the absence of an antialiasing filter is not a concern; the signal output from the DSK contains no frequencies that would alias into the baseband spectrum when sampled. But if the DUT output contains high-frequency noise or spurious signals, an external filter should be added to remove these unwanted signals. If this is done, calibration of the

Table 1—RESP Program Commands

Command	*Function and Parameters*
S	Set the start-up delay, sent as a single parameter.
T	Download the transmit sample list. The first parameter sent is the number of samples (1 to 512), followed by that many sample values.
R	Set the receive sample count (0 to 512), a single parameter.
G	Go—begins the generation/sampling process.

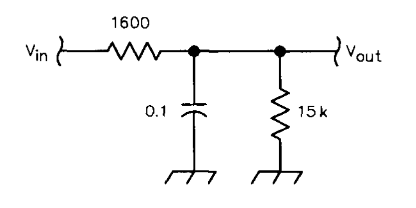


Fig 8—Schematic diagram of a simple test circuit used to evaluate the measurement system.

measurement system should be performed with this filter in place, to account for its amplitude and phase responses.

Host Application Software

Host software for this application has been written to run on an IBM PC or compatible. The host software, CEVAL, manages the DSK, computes the response of the received waveform and displays the response graphically on a CGA, EGA, VGA or Hercules monochrome display. The software is written in C and has been compiled using Borland C++, versions 3.1 and 4.0, as well as Turbo C 2.0. Since the graphics routines are specific to the Borland compilers, porting the application to another compiler or host computer will require some work. The full source code for the host and DSK software, and compiled executable versions of the two programs, can be downloaded from the ARRL BBS (203-666-0578) or via the Internet using anonymous FTP from ftp.cs.bufflalo.edu in the /pub/ham-radio directory. The file name is QEXRESP.ZIP.

Calibration and Averaging

CEVAL handles calibration by allowing the user to take a measurement with the DSK output connected to its input. To calibrate, the host takes one measurement of N samples, then finds the maximum-amplitude sample in the received data. It uses this information to adjust the start-up delay value it subsequently sends to the DSK in order to place the maximum-value sample (the center of the transmitted pulse waveform) in the middle of the received sample window. CEVAL then gets another set of samples from the DSK, now using the adjusted start-up delay. It computes the FFT of this set of samples and saves it in the "calf" array.

The reason for adjusting the start-up delay is that in order to properly determine the absolute phase of a measured signal relative to the calibration signal, the measured signal must appear later in the sample window than does the calibration signal. If we did not make this adjustment, the calibration signal pulse might appear near the end of the sample window, depending on the chosen start-up delay and any delays in the DSK system. When this pulse is further delayed by the DUT, it will appear to "wrap around" to the front of the window. To the FFT, this looks as though the DUT signal occurs *before* the calibration signal, which is clearly impossible and will lead to invalid phase results (although the relative group delay will still be correct). By adjusting the system timing to put the calibration pulse in the middle of the window, we allow the DUT to delay the signal by $N/2$ samples before wrap-around occurs. This allows adequate time to handle the delay of most systems, particularly when N is large.

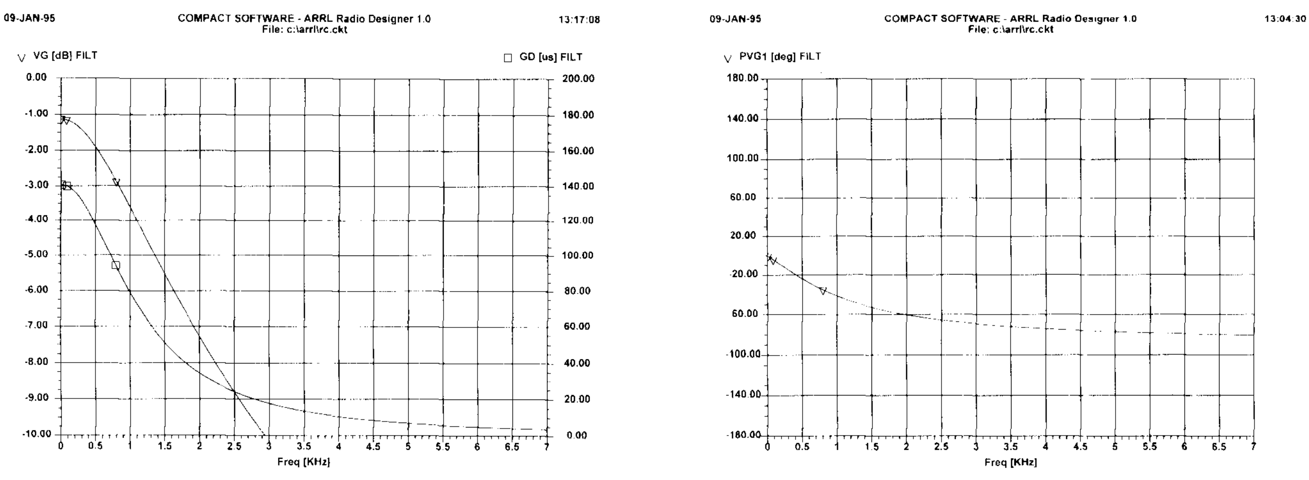
As mentioned, one of the limitations of this technique arises from the relatively small amount of energy present at each of the measurement frequencies, resulting in low signal-to-noise ratios. We can ameliorate this problem somewhat if we take multiple measurements and average the results. Each measurement adds its signal energy to the sum, while the noise, being random, tends to average out. The host software supports this approach by letting the user set an averaging factor. This tells the host how many measurements to take and applies to both calibration and measurement sampling.

The host also lets the user specify the number of samples to use, from 16 to 512 in powers of 2. Whenever the user changes either the number of samples or the averaging factor, the host discards any existing calibration data; the user must recalibrate if calibrated measurements are to be made.

Processing the Measurements

When a set of measurement samples is taken, the host displays the input waveform on the screen. This allows the user to check to ensure that the input signal isn't overflowing the DSK's analog input. This should be done before calibration, to ensure the calibration is performed using a valid input signal, as well as during DUT measurements.

The user can command CEVAL to display the amplitude response of the DUT (in linear, 1-dB/division or 5-dB/division plots), the phase response or the group-delay response. Each of these plots first requires that the FFT



(A) (B)

Fig 9—*ARRL Radio Designer* predictions of the amplitude and group-delay response (A) and the phase response (B) of the circuit of Fig 8.

of the input samples be computed, which CEVAL will do automatically as needed. If a valid calibration has been performed, the computed FFT output values are divided by the calibration values in the "calf" array to produce a calibrated result. Note that the amplitude display is relative. That is, the maximum amplitude in the display is always at 0 dB, regardless of the actual gain or loss of the DUT. Absolute gain values, if desired, can be measured using single-frequency signal techniques.

If the measurement is made with the DUT not in the system, the resulting signal should be the same as was obtained during calibration. In this case, the amplitude response should be at 0 dB, the phase should be 0 degrees and the group delay should be 0 µs for all frequencies. But system noise will affect these expected results, as shown in Fig 6. Larger values of N will spread the available pulse energy into more frequencies, lowering the signal-to-noise ratios and increasing the uncertainty of the results. Increasing the averaging factor will increase the signal-to-noise ratios, improving the results (Fig 7).

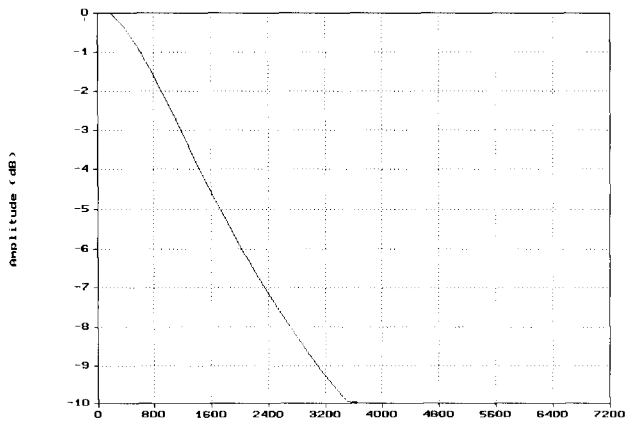
Group-Delay Measurement

As shown in Eq 3, we can estimate group delay from the phase response values at two frequencies, and that is the technique used here. When plotted on the screen, the value shown for any particular frequency is based on the phase response at that frequency and the response at the next higher frequency. In some cases, particularly at lower values of N and at low frequencies, the phase variation between successive frequencies may be larger than 2π. In such cases, an abrupt spike in the group-delay curve will occur. The only fix for this is to redo the measurement using frequencies that are closer together. That's one reason why the system includes two ranges; the lower range, which extends up to 1800 Hz instead of 7200 Hz, places the frequency points closer together for a given N.

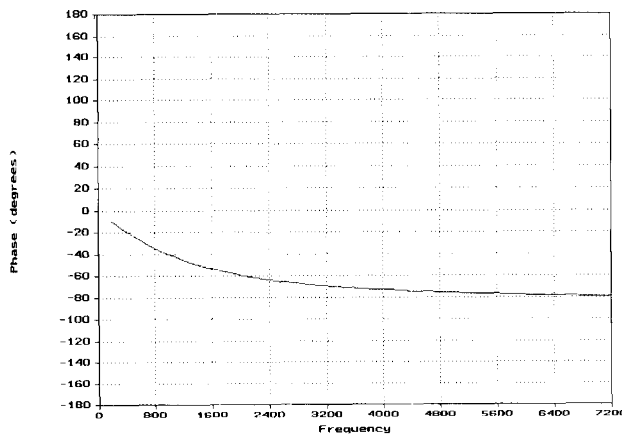
While we usually don't demand amplitude measurement accuracy better than a few tenths of a dB, we want to measure group-delay variations with an accuracy of a few microseconds. What this means is that measurement noise is usually of more concern to group-delay measurement than to amplitude measurement. To get reasonably noise-free group delays, you may need to reduce the number of samples used and/or increase the averaging factor. Since the needed values depend on the system noise, it's hard to give precise values to use. A little experimentation goes a long way in finding effective values to use for these measurements.

DC and AC Coupling

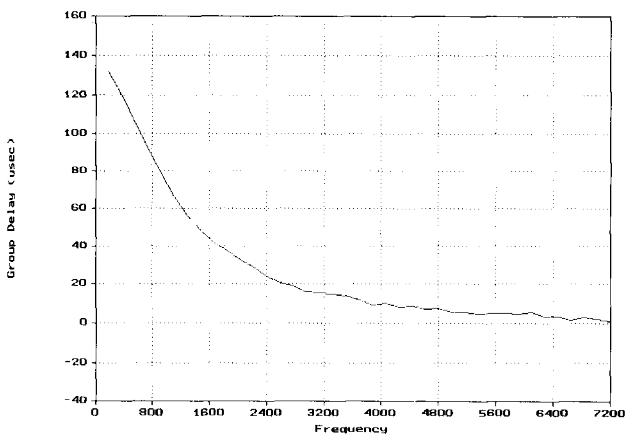
One of the frequencies measured by the system is 0 Hz (dc). This can be useful, but it can also be a problem. Suppose there is a small dc offset in the system hardware, which is likely. That offset is constant regardless of



(A)

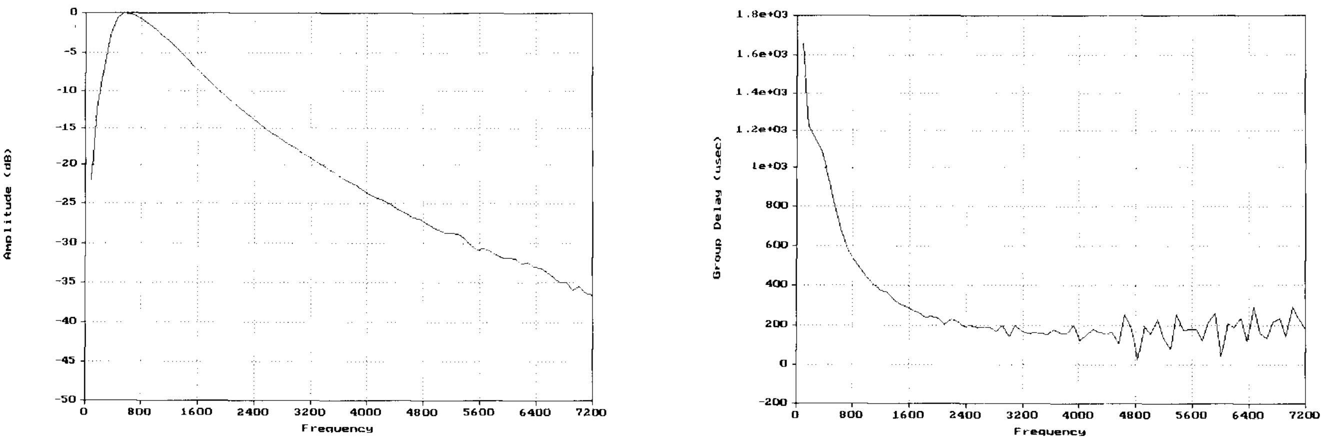


(B)



(C)

Fig 10—The measured response of the circuit of Fig 8. Compare this to the predicted response of Fig 9, noting that Fig 9 includes an overall 1-dB loss not displayed by the measurement system.



(A) (B)

Fig 11—Measured amplitude and group-delay response of an ICOM IC-281H FM receiver (speaker output).

the number of samples in the measured waveform. But as N gets large, the amount of signal energy at each frequency becomes smaller. So, the relative effect of the constant dc offset becomes larger as N increases. This is not a big problem, as the calibration procedure will account for the offset. But system noise will likely creep into the calculated response, making the displayed dc response small, but not zero. And in systems that are ac coupled—which should show a zero response at dc—the amplitude displays may show a drop from some nonzero response at the first frequency above 0 to a zero response at dc. This makes for a somewhat disconcerting display. For these reasons, CEVAL lets the user specify that the system is ac coupled. In this case, CEVAL simply doesn't display the amplitude response at 0; the trace begins at the first nonzero frequency. The phase and group-delay traces always begin above 0—what is the phase of a dc signal, anyway?

Some Real Measurements

To test the measurement system, I constructed a simple R-C circuit (Fig 8). I analyzed this circuit using *ARRL Radio Designer* to predict the amplitude, phase and group-delay responses shown in Fig 9. Then I measured the response of the circuit using the system described here, with the results shown in Fig 10. As you can see, the measured response closely tracks the calculated response. This verifies the basic functionality of the system.

Since the initial purpose of this exercise was to measure the response of receivers, I would be remiss in not including a measured receiver response. Fig 11 shows the measured response of an IC-281H FM receiver.

Measuring SSB Receivers

Earlier we noted that the FFT requires that the input signal be exactly periodic at a frequency of f_s/N. Since an SSB receiver will introduce a frequency shift equal to its tuning error, it would seem that we cannot measure an SSB receiver using this technique. But an inspection of the pulse waveform (Fig 3) shows something interesting. At the beginning and end of the waveform, the amplitude is essentially zero. Frequency-shifting this pulse by a small amount will just expand or contract the pulse width, either allowing a small part of the preceding or following pulse waveform to creep into the sample window or truncating the measured pulse waveform. But since the samples being added or cut off are very near zero amplitude, no significant discontinuity occurs at the ends of the window, and it is this discontinuity that produces spectral leakage. What happens when we process this distorted waveform via the FFT is that the FFT considers the signal it sees to be periodic at f_s/N, whether it is or not. The result in our measurements is a slight error in the displayed frequencies, caused by the slight tuning error of the receiver. So, we still get a perfectly usable result. However, we have lost the ability to do useful averaging, since each time we take a set of samples the values will be different from the previous set of samples due to the changing phase—caused by the frequency error. Therefore, as long as we limit ourselves to an averaging factor of 1, we can successfully measure the response of SSB receivers.

Conclusion

The system presented here is capable of measuring the amplitude response, from 0 to 7200 Hz, of a receiver or other system to within a fraction of a dB. Group-delay response can be measured to within a few microseconds. The system's main limitation is that it cannot make accurate group delay measurements in the presence of significant amounts of noise, but when large signal-to-noise ratios can be ensured, accurate measurements are possible.

Notes

[1] *The ARRL Handbook for Radio Amateurs*, 1995, ARRL, Newington, CT, p 18.6.

[2] Bloom, J., KE3Z, "Negative Frequencies and Complex Signals," *QEX*, September, 1994, pp 22-27.

Listing 1

```
Page 1: resp.lst


DSKA => DSP Starter Kit Assembler Rev 1.00 Thu Jan 19 07:05:06 1995
Copyright (c) 1992-1993 Texas Instruments Incorporated

0001 - ---- ----        *
0002 - ---- ----        * RESP.ASM -- TI DSK program that generates a repetitive waveform
0003 - ---- ----        * and samples a snapshot of the input signal.
0004 - ---- ----        *
0005 - ---- ----        * Serial communication with a host computer is used to control
0006 - ---- ----        * operation of this program. The host sends a set of sample values
0007 - ---- ----        * to be sent repetitively; the DSK sends the set of sample values
0008 - ---- ----        * back to the host after sampling.
0009 - ---- ----        *
0010 - ---- ----        * Copyright (c) 1995, American Radio Relay League, Inc.
0011 - ---- ----        * All Rights Reserved
0012 - ---- ----        *
0013 - ---- ----        * J. Bloom, KE3Z
0014 - ---- ----        * 04 January, 1995
0015 - ---- ----        *
0016 - ---- ----        ;-----------------------------------------------------------;
0017 - ---- ----                .include        "respi.asm"
**********************************************
*     OPENING INCLUDE FILE  respi.asm
**********************************************
0001 - ---- ----        AIC_1   .set    03870h  ;RA =TA = 1C 0011100001110000 = 03870h
0002 - ---- ----        AIC_2   .set    00205h  ;TA'=1 RA'= 1  0000001000000000101 = 00205h
0003 - ---- ----        AIC_3   .set    03e7eh  ;RB =TB =1Fh 0011111001111110 = 03e7eh
0004 - ---- ----        AIC_CMD .set    00083h  ;   COMMAND  0000000010000011 = 00083h
**********************************************
*     CLOSING FILE
**********************************************
0018 - ---- ----        TBUF    .set    0400h           ; TX sample buffer
0019 - ---- ----        RBUF    .set    0600h           ; RX sample buffer
0020 - ---- ----        ; DSK_COMM routines
0021 - ---- ----        RECV16  .set    0fa18h
0022 - ---- ----        RECV    .set    0fa16h
0023 - ---- ----        XMIT    .set    0fa1ah
0024 - ---- ----        ; Data storage in page 0
0025 - ---- ----        func    .set    068h            ; Function command from host
0026 - ---- ----        resp    .set    069h            ; "Need to respond" flag
0027 - ---- ----        sudly   .set    06ah            ; Start-up delay count
0028 - ---- ----        xsamp   .set    06bh            ; output sample
0029 - ---- ----        rsamp   .set    06ch            ; input sample
0030 - ---- ----        txcnt   .set    06dh            ; # of samples to send
0031 - ---- ----        rxcnt   .set    06eh            ; # of samples to receive
0032 - ---- ----
0033 - ---- ----                .include  "mmregs.asm"
**********************************************
*     OPENING INCLUDE FILE  mmregs.asm
**********************************************
0001 - ---- ----        ;-----------------------------------------------------------;
0002 - ---- ----        ; MMREGS.ASM                                                ;
0003 - ---- ----        ; (C) 1992-93                                               ;
0004 - ---- ----        ; Texas Instruments Inc.                                    ;
0005 - ---- ----        ; By: Keith Larson                                          ;
0006 - ---- ----        ;     TMS320Cxx DSP Applications                            ;
0007 - ---- ----        ;                                                           ;
```

```
Page 2: resp.lst

0008 - ---- ----        ; Contains a list of commonly used values                   ;
0009 - ---- ----        ;-----------------------------------------------------------;
0010 - ---- ----        DRR     .set    0
0011 - ---- ----        DXR     .set    1
0012 - ---- ----        TIM     .set    2
0013 - ---- ----        PRD     .set    3
0014 - ---- ----        IMR     .set    4
0015 - ---- ----        GREG    .set    5
0016 - ---- ----        B2_D    .set    060h
0017 - ---- ----        B0_D    .set    0200h
0018 - ---- ----        B1_D    .set    0400h
0019 - ---- ----        B3_D    .set    0600h
0020 - ---- ----        B0_P    .set    0FA00h
0021 - ---- ----        B1_P    .set    0FC00h
0022 - ---- ----        B3_P    .set    0FE00h
0023 - ---- ----        ;  use a comment to end .include files ending with .set
0024 - ---- ----        ;  to avoid an ERROR.  Also, comment must start in column 2
**********************************************
*     CLOSING FILE
**********************************************
0034 - ---- ----
0035 - ---- ----        ; Interrupt vectors
0036 - ---- ----
0037 - ---- ----                .ps     0fa00h
0038 0 fa00 ff80                B       start           ;RS
0039 0 fa02 ff80                B       start           ;INT0
0040 0 fa04 ff80                B       start           ;INT1
0041 0 fa06 ff80                B       start           ;INT2
0042 0 fa08 ff80                B       start           ;TINT
0043 0 fa0a ff80                B       RINT            ;RINT
0044 0 fa0c ce00                eint                    ;XINT
0045 0 fa0d ce26                ret
0046 - ---- ----
0047 - ---- ----        ; Application code
0048 - ---- ----
0049 - ---- ----                .ps     0FB00h
0050 - ---- ----                .entry
>>>>> ENTRY POINT SET TO fb00
0051 0 fb00 c800        start:  ldpk    0               ; All direct addressing is to MMRs and B2
0052 0 fb01 ce0e                fort    0               ; Serial port : 16 bit
0053 0 fb02 ce20                rtxm                    ;             : ext. FSX
0054 0 fb03 ce37                sfsm                    ;             ; burst mode
0055 0 fb04 ca80                lack    080h            ; AIC reset by pulsing /BR (Global Data)
0056 0 fb05 6800                sach    DXR             ; send 0 to DXR (AIC)
0057 0 fb06 6000                sacl    GREG            ; 256 * 100 nS /BR pulse
0058 0 fb07 d000                lrlk    AR0,0FFFFh
0059 0 fb09 cbff                rptk    255             ; read junk from address 0xFFFF
0060 0 fb0a 2088                lac     *,0,AR0
0061 0 fb0b ce3d                conf    1
0062 0 fb0c ce03                sovm                    ; catch accumulator overflows
0063 - ---- ----        ;--------------------------------
0064 0 fb0d ca24        AIC_RS2 lack    024h            ; Turn on XINT
0065 0 fb0e 6000                sacl    IMR
0066 0 fb0f ce1f                idle
0067 0 fb10 d001                lalk    AIC_1           ; Load each AIC configuration word
0068 0 fb12 fe80                call    AIC_2nd         ; and load it into the AIC
```

Page 3: resp.lst

```
0069 0 fb14 d001           lalk    AIC_2
0070 0 fb16 fe80           call    AIC_2nd
0071 0 fb18 d001           lalk    AIC_3
0072 0 fb1a fe80           call    AIC_2nd
0073 0 fb1c d001           lalk    AIC_CMD
0074 0 fb1e fe80           call    AIC_2nd
0075 - ---- ----   ;---------------------------------------------------------------
0076 0 fb20 ce06           rsxm
0077 - ---- ----
0078 0 fb21 ca00           zac
0079 0 fb22 6000           sacl    IMR             ; Mask all INTs
0080 - ---- ----
0081 0 fb23 ff80           b       waity           ; Wait for command from host
0082 - ---- ----
0083 - ---- ----   ; Receive and respond to new command
0084 - ---- ----   ; Command                     Parameters               Parameter words
0085 - ---- ----   ;   S Set start-up delay count  sudly                  1
0086 - ---- ----   ;   T Set transmit sample list  txcnt; sample values   1 + txcnt
0087 - ---- ----   ;   R Set receive sample count  rxcnt                  1
0088 - ---- ----   ;   G Go--generate and sample   ---                    0
0089 - ---- ----   ;
0090 - ---- ----   ; If rxcnt <> 0, the G command causes the program to begin sending
0091 - ---- ----   ; the tx sample list over and over, sudly sample periods later, the
0092 - ---- ----   ; program begins storing a snapshot of rxcnt samples of the input
0093 - ---- ----   ; signal. Once rxcnt samples are in, they are sent serially to the
0094 - ---- ----   ; host and the program awaits a new command.
0095 - ---- ----   ;
0096 - ---- ----   ; If rxcnt is set to 0, the G command causes the program to send the
0097 - ---- ----   ; tx signal endlessly, until a command is received on the serial port.
0098 - ---- ----
0099 - ---- ----   docmd:
0100 0 fb25 fe80           call    RECV            ; Get function command from host
0101 0 fb27 6000           sacl    func            ; Save function ID
0102 0 fb28 cd53           subk    053h            ; S = Set startup delay
0103 0 fb29 f580           bnz     docmd2
0104 0 fb2b fe80           call    RECV16
0105 0 fb2d 6000           sacl    sudly
0106 - ---- ----   docmd2:
0107 0 fb2e 2000           lac     func
0108 0 fb2f cd54           subk    054h            ; T = tx sample list
0109 0 fb30 f580           bnz     docmd3
0110 0 fb32 fe80           call    RECV16          ; Get TX count
0111 0 fb34 6000           sacl    txcnt
0112 0 fb35 d200           lrlk    AR2,TBUF        ; TX sample buffer pointer
0113 0 fb37 3300           lar     AR3,txcnt
0114 0 fb38 558b           larp    AR3
0115 0 fb39 5590           mar     *-
0116 - ---- ----   tfill:                          ; Get list of TX samples
0117 0 fb3a fe80           call    RECV16
0118 0 fb3c 558a           larp    AR2
0119 0 fb3d 60ab           sacl    *+,0,AR3
0120 0 fb3e fb90           banz    tfill,*-
0121 - ---- ----   docmd3:
0122 0 fb40 2000           lac     func
0123 0 fb41 cd52           subk    052h            ; R = rx count
0124 0 fb42 f580           bnz     docmd4
```

Page 4: resp.lst

```
0125 0 fb44 fe80           call    RECV16
0126 0 fb46 6000           sacl    rxcnt
0127 - ---- ----   docmd4:
0128 0 fb47 2000           lac     func
0129 0 fb48 cd47           subk    47h             ; G = go
0130 0 fb49 f580           bnz     docmd5
0131 0 fb4b 3500           lar     AR5,sudly       ; Start-up delay
0132 0 fb4c d100           lrlk    AR1,TBUF        ; TX data buffer pointer
0133 0 fb4e d200           lrlk    AR2,RBUF        ; RX data buffer pointer
0134 0 fb50 3300           lar     AR3,txcnt       ; TX sample count
0135 0 fb51 558b           larp    AR3
0136 0 fb52 5590           mar     *-
0137 0 fb53 3400           lar     AR4,rxcnt       ; RX sample count
0138 0 fb54 ff80           b       runit
0139 - ---- ----   docmd5:
0140 0 fb56 ff80           b       waity           ; Ignore invalid commands
0141 - ---- ----   runit:
0142 0 fb58 ca00           zac
0143 0 fb59 6000           sacl    resp
0144 - ---- ----
0145 - ---- ----   ; Enable RINTs to start processing
0146 - ---- ----
0147 - ---- ----   stint:
0148 0 fb5a c800           ldpk    0
0149 0 fb5b ca10           lack    010h            ; AIC RINT
0150 0 fb5c 6000           sacl    IMR             ; where INT0 indicates EOC (End Of Conv)
0151 0 fb5d ce00           eint
0152 - ---- ----
0153 - ---- ----   ; Idle loop. Just processes RINTs until BIO goes low (incoming
0154 - ---- ----   ; command from host) or resp flag is set.
0155 - ---- ----
0156 0 fb5e ce1f   AGN:    idle
0157 0 fb5f fa80           bioz    docomm
0158 0 fb61 2000           lac     resp
0159 0 fb62 f680           bz      AGN             ; Loop
0160 - ---- ----
0161 - ---- ----   ; Dump the receive buffer to the host
0162 - ---- ----
0163 - ---- ----   dump:
0164 0 fb64 c800           ldpk    0
0165 0 fb65 ca00           zac
0166 0 fb66 6000           sacl    IMR
0167 0 fb67 6000           sacl    DXR
0168 0 fb68 d200           lrlk    AR2,RBUF
0169 0 fb6a 3400           lar     AR4,rxcnt       ; Buffer count
0170 0 fb6b 558c           larp    AR4
0171 0 fb6c 5590           mar     *-
0172 0 fb6d 558a           larp    AR2
0173 - ---- ----   dumpl:                          ; Send each 16-bit value
0174 0 fb6e 20a0           lac     *+
0175 0 fb6f fe80           call    XMIT16
0176 0 fb71 558c           larp    AR4
0177 0 fb72 fb9a           banz    dumpl,*-,AR2
0178 - ---- ----
0179 - ---- ----   ; Wait for a command
0180 - ---- ----
```

Page 5: resp.lst

```
0181 - ---- ----          waity:
0182 0 fb74 fa80                  bioz    docomm          ; Get new command
0183 0 fb76 ff80                  b       waity
0184 - ---- ----
0185 - ---- ----          ; Here when host wants to talk to us. Respond with R(eady)
0186 - ---- ----
0187 - ---- ----          docomm:
0188 0 fb78 c800                  ldpk    0
0189 0 fb79 ca00                  zac                     ;Turn off interrupts
0190 0 fb7a 6000                  sacl    IMR
0191 0 fb7b c128                  lark    AR1,40
0192 0 fb7c 5589                  larp    AR1
0193 0 fb7d cbff          waitx   rptk    255
0194 0 fb7e 5500                  nop
0195 0 fb7f fb90                  banz    waitx,*-
0196 0 fb81 ca52                  lack    052h            ; Send the response char
0197 0 fb82 fe80                  call    XMIT
0198 0 fb84 ff80                  b       docmd
0199 - ---- ----
0200 - ---- ----          RINT:
0201 - ---- ----
0202 - ---- ----          ; I/O the sample values
0203 - ---- ----
0204 0 fb86 c800                  ldpk    0
0205 0 fb87 2000                  lac     xsamp
0206 0 fb88 6000                  sacl    DXR
0207 0 fb89 2000                  lac     DRR
0208 0 fb8a 6000                  sacl    rsamp
0209 - ---- ----
0210 - ---- ----          ; Generate the next output sample
0211 - ---- ----
0212 0 fb8b 5589                  larp    AR1             ; Point to TX buffer
0213 0 fb8c 20ab                  lac     *+,0,AR3
0214 0 fb8d 6000                  sacl    xsamp
0215 0 fb8e fb90                  banz    chksu,*-        ; Not at end of buffer, go do delay
0216 - ---- ----                  ; End of TX buffer--start over
0217 0 fb90 3300                  lar     AR3,txcnt
0218 0 fb91 5590                  mar     *-
0219 0 fb92 d100                  lrlk    AR1,TBUF
0220 - ---- ----
0221 - ---- ----          ; Count down the start-up delay, if any
0222 - ---- ----
0223 - ---- ----          chksu:
0224 0 fb94 558d                  larp    AR5
0225 0 fb95 fb80                  banz    cntsu,*         ; Count down if in start-up delay
0226 0 fb97 ff80                  b       getrx           ; Not in delay, store input samples
0227 - ---- ----          cntsu:
0228 0 fb99 5590                  mar     *-              ; Count down start-up delay
0229 0 fb9a ff80                  b       outcor
0230 - ---- ----
0231 - ---- ----          ; Store the input sample in the RX buffer
0232 - ---- ----
0233 - ---- ----          getrx:
0234 0 fb9c 558c                  larp    AR4             ; Check receive count
0235 0 fb9d fb80                  banz    getrx1,*
0236 0 fb9f ff80                  b       outcor
```

Page 6: resp.lst

```
0237 0 fba1 558a          getrx1  larp    AR2
0238 0 fba2 2000                  lac     rsamp
0239 0 fba3 60ac                  sacl    *+,0,AR4        ; Store the sample
0240 0 fba4 5590                  mar     *-
0241 0 fba5 fb80                  banz    outcor,*        ; Buffer, not full, continue
0242 0 fba7 ca01                  lack    1               ; Flag foreground routine to stop
0243 0 fba8 6000                  sacl    resp
0244 - ---- ----          outcor:
0245 0 fba9 2000                  lac     xsamp
0246 0 fbaa d004          finint: andk    0fffch
0247 0 fbac 6000                  sacl    xsamp
0248 0 fbad ce00                  eint
0249 0 fbae ce26                  ret
0250 - ---- ----          *******************************************************************
0251 0 fbaf df02          AIC_2nd adlk    6,15            ;set ACCU_hi = 3 for secondary XMIT
0252 0 fbb1 ce1f                  idle                    ;Wait for a XINT
0253 0 fbb2 6800                  sach    DXR
0254 0 fbb3 ce1f                  idle                    ;ACCU_hi requests 2nd XMIT
0255 0 fbb4 6000                  sacl    DXR
0256 0 fbb5 ce1f                  idle                    ;ACCU_lo sets up registers
0257 0 fbb6 6200                  sacl    DXR,2           ;close command with LSB = 00
0258 0 fbb7 ce1f                  idle
0259 0 fbb8 ce00                  eint
0260 0 fbb9 ce26                  ret
0261 - ---- ----          ;==================================================================
0262 0 fbba fe80          XMIT16: call    XMIT            ;xmit LO
0263 0 fbbc fe80                  call    XMIT            ;xmit HI
0264 0 fbbe ce26                  ret
>>>>> FINISHED READING ALL FILES

>>>>> ASSEMBLY COMPLETE: ERRORS:0   WARNINGS:0
```

Digital Communications

By Harold E. Price, NK6K

Where Everybody Knows Your Name

All right, cheer up. I continue to get a lot of mail about the previous doom, defeat and despair columns, but no more of that—for a while. This column covers a random collection of other interesting things. One that should interest you is the AMSAT Phase 3D satellite, scheduled for launch in 1996. You should be interested because it will be a heavy user of interesting modulation schemes, will support a variety of interesting communications protocols, and your fellow hams are investing several millions dollars of hard cash (hardware and launch costs) and countless man-hours in the project.

P3D

I discussed P3D in the June 1994 column. This 400-kg satellite will have up to 250 watts PEP of output, or about 60 watts continuous. Of that 60 watts, about 20 watts will be allocated to the digital transponder (called RUDAK-U). The digital RF inputs and output pass through a 10.7-MHz IF matrix, giving us access to a variety of uplink and downlink frequencies. The actual frequency bands are shown in Table 1.

P3D will be flying a plethora of DSP-based modems. While the Surrey Satellite Technology UoSATs were the first to fly DSP modems on amateur spacecraft, P3D will be flying several: eight modulators and eight demodulators. The modems can appear anywhere within the digital subbands described in Table 1.

Most of the information below comes from the RUDAK-U project manager and lead hardware designer, Lyle Johnson, WA7GXD. The design is near-final but still subject to change. The basic configuration of the digital section is shown in Fig 1. There are two CPUs, a V53 and an 80386. Each will have 16 Mbytes of error-detection-and-correction (EDAC) protected program- and file-storage space. Each processor has a connection to the payloads via a serial port, the 1-Mbit/s CAN-bus or both. Each processor has its own set of "low" speed DSP-based modems (up to 56 kbit/s or so). The processors share a single 256-kbit/s modem.

The two main processors are reloadable from the ground or from one another. The DSP modems and payloads are reloadable from the ground via the main processor's onboard storage. The processors/modems complex has 18 programmable processors.

The 9600-baud modems are hardwired and will be used for initial loading and as backups to the DSP modems. They will be on fixed frequencies in the digital subband. At least one of the frequencies will not be published, to provide a contention-free command and loading channel.

The DSP modulators and demodulators use the ADSP2171 CPU. Each modulator and demodulator chain has a separate processor, allowing the full power of the DSP chip to be used for a single half-duplex link. This will allow high rates, up to 56 kbit/s, or very heavily coded low baud rates. Each DSP has 10 kbytes of internal, non-EDAC-protected memory. The DSPs will perform an internal CRC on pro-

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

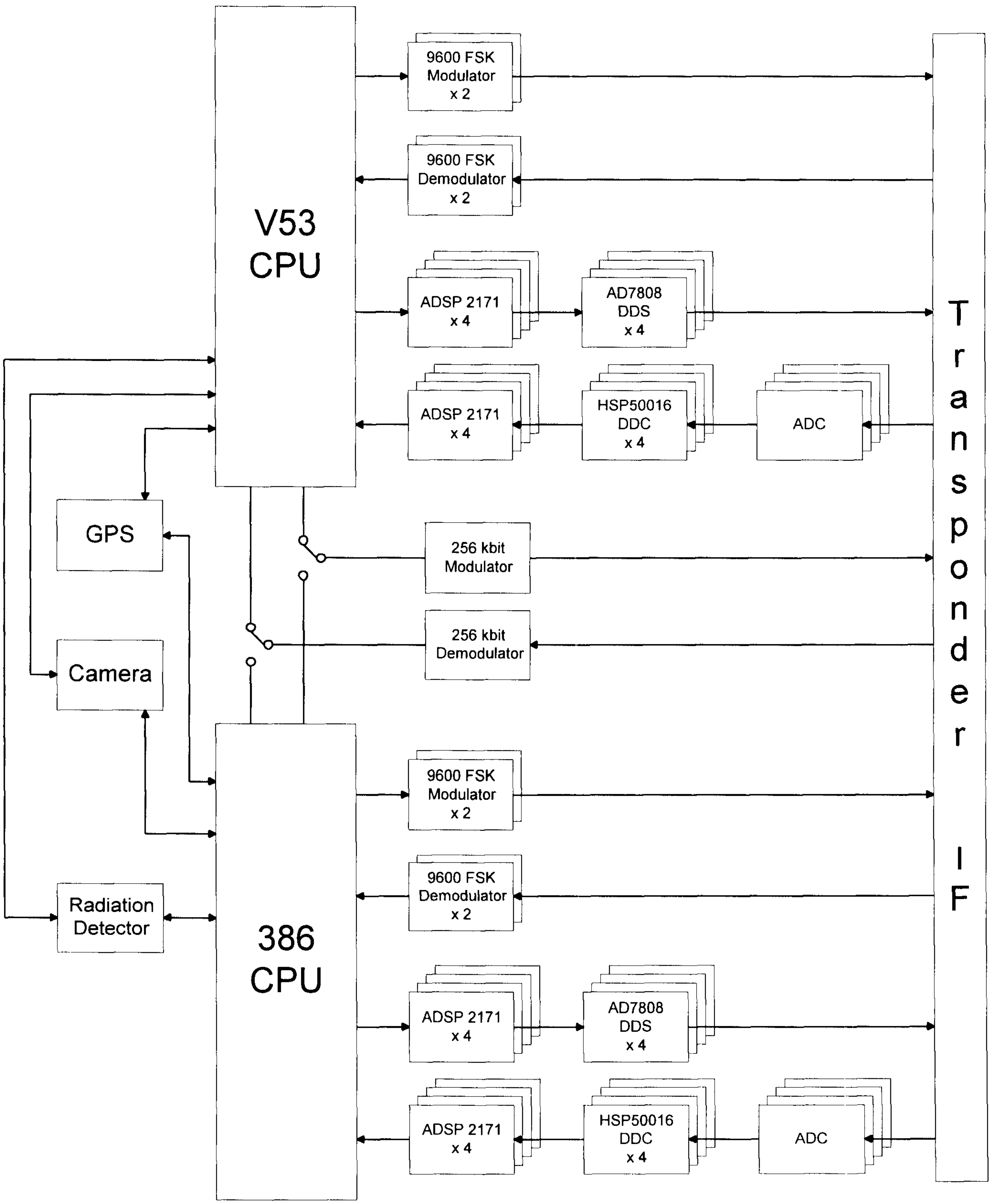


Fig 1—Block diagram of the P3D digital electronics package.

gram memory every second and report back to the main processor, which will reboot the DSP on a bad CRC or timeout.

A demodulator is fed from a high-speed video ADC. Though each demodulator could use its own ADC, each is seeing the same IF and could instead share an ADC with one or more other demodulators. The decision, as always, is one of redundancy versus complexity and has not yet been made. Each demodulator uses its own HSP50016 Digital Down Converter (DDC) to convert the digitized 10.7-MHz IF to a frequency range more suitable for digital processing.[1] This also allows the uplink frequency to be anywhere in the passband, under software control. Likewise, the modulators use the AD7008 Direct Digital Synthesizer (DDS) to generate 10.7-MHz IF signals. This allows the modulators to also appear anywhere in the passband, under software control. The DDC and DDS allow the processor to specify a phase increment to an internally generated sine wave. The DDS also allows an amplitude to be specified. Since the DSP need not generate each point on the sine wave, the input and output frequencies can be much higher than the DSP chip alone could handle.

[1]The DDC has been mentioned previously in *QEX*, see Anderson, P. T., *"A Simple SSB Receiver Using a Digital Down Converter,"* *QEX*, March 1994, for an overview.

The DDS and DDC can be used to allow each ADSP2171 to process more than one low-bit-rate signal at a time, giving us more than 16 uplinks—perhaps as many as 32 or 48 low-bit-rate uplinks. What could we do with such a thing? Why, that's the fun part, of course. Readers are invited to send in application ideas for all this horsepower. Before we get too carried away with thoughts of competing with Qualcomm and Orbcomm, let's review the link budget.

TANSTAAFL

The above is an acronym for "There ain't no such thing as a free lunch," a phrase I was first exposed to as a child in Robert Heinlein's *The Moon is a Harsh Mistress* and which I've encountered in real life ever since. At an altitude of 43,000 km, the P3D orbit has a few advantages over the low-Earth-orbit (LEO) spacecraft (800-1300 km) that digital users are used to. The spacecraft is much further away, meaning it moves more slowly and is visible for longer periods of time. It is also larger, meaning it can generate much more power, for louder downlinks.

Here is the TANSTAAFL part.

Since P3D is much farther away than an LEO satellite like KO-23, the path loss is also much greater. The increased path loss, in fact, just about wipes out the advantage of the higher power. For example, consider the case of a typical ground user of the 9600-baud UoSAT spacecraft, UO-22, KO-23 and KO-25. Most of these use what is called an "OSCAR-10" class station, meaning tracking antennas with about 10 dB of gain and 10 to 100 watts of uplink power. For the LEO satellites, which transmit about two watts on the downlink, this gives plenty of link margin.

For P3D, a similar amount of downlink power results in −0.1 dB of link margin. We have a rule of thumb in the amateur satellite world that says the link margin on paper needs to be about 10 dB to give adequate performance at a typical user station. This 10 dB is usually labeled "implementation loss" and, in my case, equates to the inability to correctly place an N-type connector on a piece of coax. Some link margins for P3D, and the data you need to compute your own, are in Table 2.

I want to be very clear about this. Much of the PR the P3D campaign sends out, and I'm guilty as well, has been talking about 250-watt transmitters and much-improved link performance for current users. The 250-watt figure and the improvements are from the point of view of current AO-10 and AO-13 *voice* users, not the current LEO 9600-baud *data* users.

Still, the goal of the RUDAK-U module is to service the current digital satellite user community. To do this, we'll need to assign a substantial amount of our downlink power budget to a single 9600-baud downlink in the 70-cm band. This leaves us with the challenge of finding interesting applications for lower power, but presumably more heavily DSP-processed, modulation schemes. We'll need matching DSP modems on the ground as well. In fact, the P3D project could

Table 1—P3D Frequencies

These are the final P3D frequencies (crystals have been ordered) and have been coordinated with IARU bandplans. Data is from the AMSAT News Service.

Uplinks

Band	*Digital (MHz)*	*Analog (MHz)*	*Center (MHz)*
15 m	N/A	21.210 - 21.250	21.230
2 m	145.800 - 145.840	145.840 - 145.990	145.915
70 cm	435.300 - 435.550	435.550 - 435.800	435.675
23 cm(1)	1269.000-1269.250	1269.250-1269.500	1269.375
23 cm(2)	1268.075-1268.325	1268.325-1268.575	1268.450
13 cm(1)	2400.100-2400.350	2400.350-2400.600	2400.475
13 cm(2)	2446.200-2446.450	2446.450-2446.700	2446.575
6 cm	5668.350-5668.550	5668.550-5668.800	5668.675

Downlinks

Band	*Digital (MHz)*	*Analog (MHz)*	*Center (MHz)*
10 m	29.330 MHz +/−5 KHz	(To be used for digitized voice bulletins)	
12 m	145.955 - 145.990	145.805 - 145.955	145.880
70 cm	435.900 - 436.200	435.475 - 435.725	435.600
13 cm	2400.650 - 2400.950	2400.225 - 2400.475	2400.350
3 cm	10451.450-10451.750	10451.025-10451.275	10451.150
1.5 cm	24048.450-24048.750	24048.025-24048.275	24048.150

All downlink passbands are inverted from the uplink passbands.

Beacons

Band	*Beacon-1*	*Beacon-2*
2 m	N/A	N/A
70 cm	435.450	435.850
13 cm	2400.200	2400.600
3 cm	10451.000	10451.400
1.5 cm	24048.000	24048.400

Table 2—P3D Link Margins from WA7GXD

Downlink, 9600-baud FSK

Frequency, MHz	436
Spacecraft	
Transmitter power, dBm	43
Line Losses, dB	0.7
Antenna Gain, dBic	13
Path	
Sat. Altitude, km	43800
Max. distance, km	49765
Polarization Loss, dB	0
Atmospheric Loss, dB	0.3
Ground Station	
Isotropic Signal at Ground, dBm	−134.1
Antenna Gain, dBic	13
Sky Temperature, K	50
Feedline Loss, dB	1
Receiver NF, dB	0.5
Bandwidth, kHz	15
Data Rate, symbols/sec	9600
Receiver Noise Temperature, K	169.6
Receiver Noise Power, dBm	−134.5
User S/N, dB	13.4
Link Margin, dB	9.9

The spacecraft antenna gain at 146 MHz is 10 dBic; at 2400 MHz it is 18 dBic.

Uplink, 9600-baud FSK

Frequency, MHz	146	1270
Ground Station		
Transmitter power, dBm	40	40
Line Losses, dB	1	1
Antenna Gain, dBic	12	24
Path		
Sat. Altitude, km	43800	43800
Max. distance, km	49765	49765
Polarization Loss, dB	0	0
Atmospheric Loss, dB	1	0.3
Spacecraft		
Isotropic Signal at Spacecraft, dBm	−119.6	−125.7
Antenna Gain, dBic	10	15
Sky Temperature, K	300	300
Feedline Loss, dB	0.7	0.7
Receiver NF, dB	1	1
Bandwidth, kHz	15	15
Data Rate, symbols/sec	9600	9600
Receiver Noise Temperature, K	438.9	438.9
Receiver Noise Power, dBm	−130.4	−130.4
User S/N, dB	20.8	19.7
Link Margin, dB	7.3	6.2

The 1270-MHz ground antenna gain is for a 1-meter dish.

well exceed the current amateur capacity to generate modem software. Volunteers?

There are other factors to keep in mind. RUDAK can transmit on more than one downlink band at a time. For example, we expect the spacecraft will often be in a mode allowing both the 435-MHz and 2400-MHz downlinks to be used. While we are servicing old-style users on 435 MHz at 9600 baud, we could be handling gateways with big antennas and higher data rates at 2400 MHz. Back in the TANSTAAFL category, while each user will have a longer access time—several hours instead of ten minutes per pass—more users can see the satellite at the same time. Will this lead to more contention, or will the long access times lead to less contention, since users aren't all trying to download in the same ten-minute interval?

All in all, we believe that we can provide access to P3D from current UoSAT 9600-baud FSK ground stations. We may also provide access to Microsat 1200-baud PSK users. And we can simultaneously, on the same or other bands, provide access with new modulation schemes and new protocols. There will be fun for protocol developers and well as software modem designers on this mission.

Who Watches the Watchers?

Perhaps Orwell, in *1984*, wouldn't have been quite so worried had he known we would be able to use the viewscreen to watch the watchers. I'm talking about the FCC's being on the Internet. Through the FTP site ftp.fcc.gov, or the much easier to use World Wide Web site http:/www.fcc.gov/, we have access to at least the public pronouncements of our licensing authority. Included in the collection of files is a list of the phone numbers and office locations of several hundred FCC employees. There is also a WAIS searchable index for the documents. A search on "KARN" produced the following:

> Report No. DC-2700 ACTION IN DOCKET CASE December 29, 1994 COMMISSION DENIES RECONSIDERATION OF ORDER CONCERNING MESSAGE FORWARDING SYSTEMS IN THE AMATEUR SERVICE (PR DOCKET NO. 93-85)
>
> The Commission has denied Phil Karn reconsideration of its decision concerning message forwarding systems in the Amateur Service. Karn sought reconsideration of the Com-

mission's decision that requires the licensee of the forwarding station to either authenticate the identity of the station from which its accepts communications on behalf of the system, or accept accountability for the content of the message.

On March 30, 1994, the FCC adopted an Order which provided that in contemporary message forwarding systems, the control operators of intermediate forwarding stations, other than the first forwarding station, would not be held accountable when their stations retransmitted improper communications inadvertently. The purpose of the Order was to relax the amateur service rules to enable these systems to operate at high speed while retaining the minimum safeguards necessary to prevent misuse.

Denying reconsideration, the Commission said the Order did not address, nor was it intended to address, what accommodations should be made for message forwarding systems that may be developed in the future. This issue appeared to be the main concern of Karn's request for reconsideration. The Commission said that if the present accommodation becomes unworkable in a system using a different architecture, the managers of that system can request necessary rule changes at the appropriate time.Action by the Commission December 23, 1994, by Memorandum Opinion and Order (FCC 94-344). Chairman Hundt, Commissioners Quello, Barrett, Ness, and Chong.

If we could get scanned images of the public's submitted documents, we'd really be in business. I've mentioned all of this for a few reasons. The first is to again encourage the WWW impaired among you to make the effort to get on the web. The second is to remind you that the FCC did codify something many of us had been talking about for a long time, the concept of a "pipe" where only the people at the ends of the pipe were responsible for the content, not those stations who make up the pipe. As Phil pointed out, there is still room for improvement in those rules. And no matter how automated we get, it is still our responsibility to police our own network. Let's be careful out there.

A Little Grumping

It is nice, from time to time, to get confirmation that I'm not totally out in left field. Walter G. Piotrowski, WB1ERE, writes that he has an "appliance machine" and a radio machine, as I described in Gloom III. He also laments the lack of progress in some areas by asking "Where's the $250 spread-spectrum radio? Where's the $250 radio that my computer can tune, and the routing protocol for my computer that makes intelligent use of it? Where are the ideas that make intelligent use of all the frequencies that we have available—instead of the recycled, everybody is on the same wire, Ethernet/Internet thing that we seem stuck with?" Good questions. See you next time.