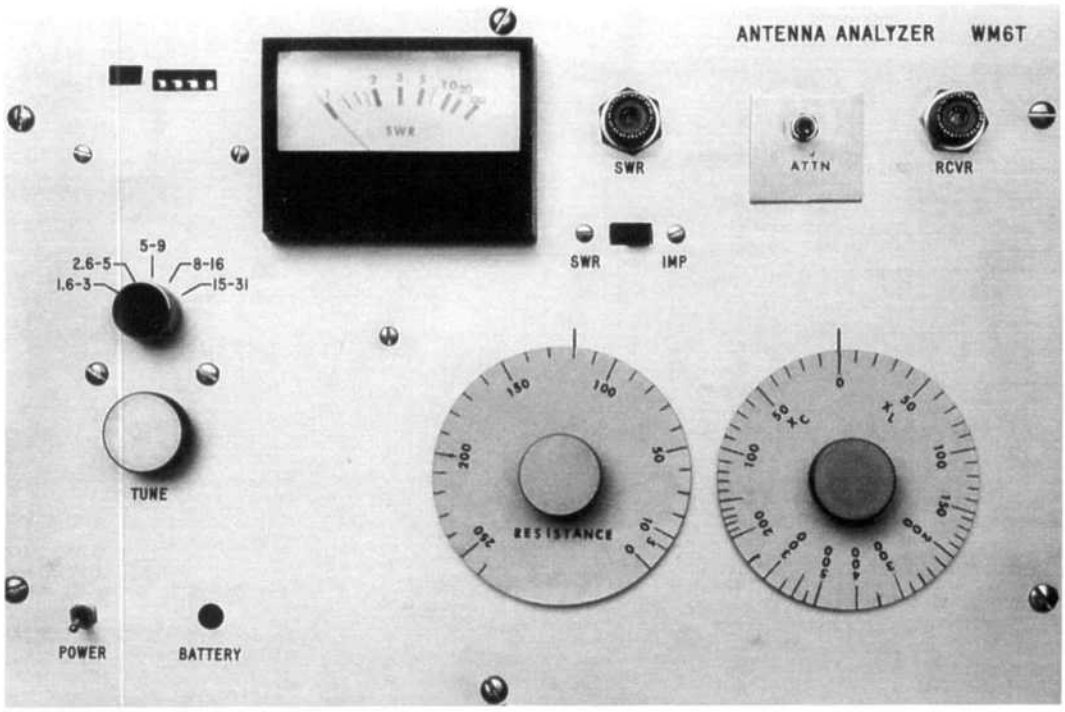
$1.75

QEX

ARRL Experimenter's Exchange

August 1994



The Comprehensive Antenna Analyzer

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

QEX

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 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:

Put as many antenna measurement tools in one unit as possible; that's the approach WM6T took for this antenna analyzer.

ISSUE NO. 150

Features

Columns

August 1994 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

Depressing Isn't the Word

Harold Price is depressed. In this month's "Digital Communications" column, Harold grumbles about the lack of progress in amateur packet radio over the last decade or so. Is he being too hard on packeteers? Not really.

A few days ago, we received a press release from the International Telecommunications Union (ITU), which says that the V.34 telephone modem standard has been approved by ITU Study Group 14. What this means is that V.34 will probably become an official standard within a few months. By mid-1995, the market will be awash in V.34 modems. These modems will support data rates of up to 28.8 kbit/s.

Contrast the advancement in telephone modem technology with that of amateur packet modem technology:

Year	*Typical Telephone Modem*	*Typical Packet Modem*
1979	300 baud	1200 baud
1995	28,800 baud	1200 baud

Oh, sure, there are such things as 9600-bit/s packet modems; even 56-kbit/s packet modems exist. But they aren't in widespread use because they don't plug into the microphone jack of a standard voice-grade radio. Most packet communication still takes place over 1200-bit/s, half-duplex, hidden-terminal-filled networks. That's more than depressing; it's pathetic.

And telephone modems aren't even a particularly high standard of comparison. Squeezing such high-speed signals into the narrow bandwidth of a telephone channel requires designing systems that operate near the Shannon bound. For higher-speed packet, we don't need to do anything nearly so esoteric because we aren't limited to such a narrow bandwidth.

The thing is, network throughput and delay have a large impact on the usability of the network. It's not just a case of being able to send our present short text messages coast-to-coast in a few minutes, as opposed to the hours (or days) it takes now. There are many potential network applications we can't do *at all* at present, but that we could do with a better network. Obvious examples are voice and image communication, as well as real-time interactive computer applications.

So why is packet lagging behind? The answer is not a technological one. We know how to send data faster and more reliably. As we said, higher-speed packet technologies do exist. But they are not available off-the-shelf. And that's the key. Like it or not—and many experimenters do not—a robust, efficient packet network will not get built from homebrew parts. There just aren't enough builders available to make it happen. For a network to be built, we need to be able to dial an 800 number and order the network hardware—all of it.

How we get to that point is the question. It's largely an economic problem. Can some group or company develop the needed pieces and make them available at prices that are viable? Never say never, but it hasn't happened in the 15 years packet has been around. And that's *really* depressing.

This Month in *QEX*

"A Comprehensive Antenna Analyzer" is just the thing for those upcoming fall antenna projects. Aubra E. Tilley, WM6T, provides some good ideas for one, embodied in a working design.

A low-cost, accessible DSP development platform is one of the most-requested items by experimenters these days. "Programming a DSP Sound Card for Amateur Radio" provides just that, as Johan B. Forrer, KC7WW, shows us.

John A. Magliacane, KD2BD, wants you on the Pacsats! And he's put his design where his mouth is with "The KD2BD Pacsat Modem."

Along with grumbling about packet (non)advancements, Harold Price, NK6K, provides some sage advice about virus avoidance in this month's "Digital Communications" column.—*KE3Z, email: jbloom@arrl.org.*

A Comprehensive Antenna Analyzer

Putting a signal source, impedance bridge and SWR bridge in one unit makes an effective measurement tool.

By Aubra E. Tilley, WM6T

Our interests in antennas include knowledge of their radiation patterns, which affect propagation, and their electrical properties, which affect impedance matching to transmitting and receiving apparatus and associated transmission lines. The antenna analyzer described here directly measures the electrical properties. However, it has adequate power output to make it a useful instrument for radiation pattern measurements.

The analyzer incorporates an analog signal generator with a digital counter readout, a precision impedance bridge and an SWR meter. Omitted from my initial ambitious goal is a built-in detector for the impedance bridge. Presently, an external communication receiver is used for this purpose.

The analyzer comprises three principal parts: a signal generator (oscillator and amplifier); bridges (impedance bridge and SWR bridge); and the power supply (+5-V regulator, –5-V inverter, 12-V battery and battery charger).

Notes appear on page 8.

PO Box 429
Pauma Valley, CA 92061

Signal Generator

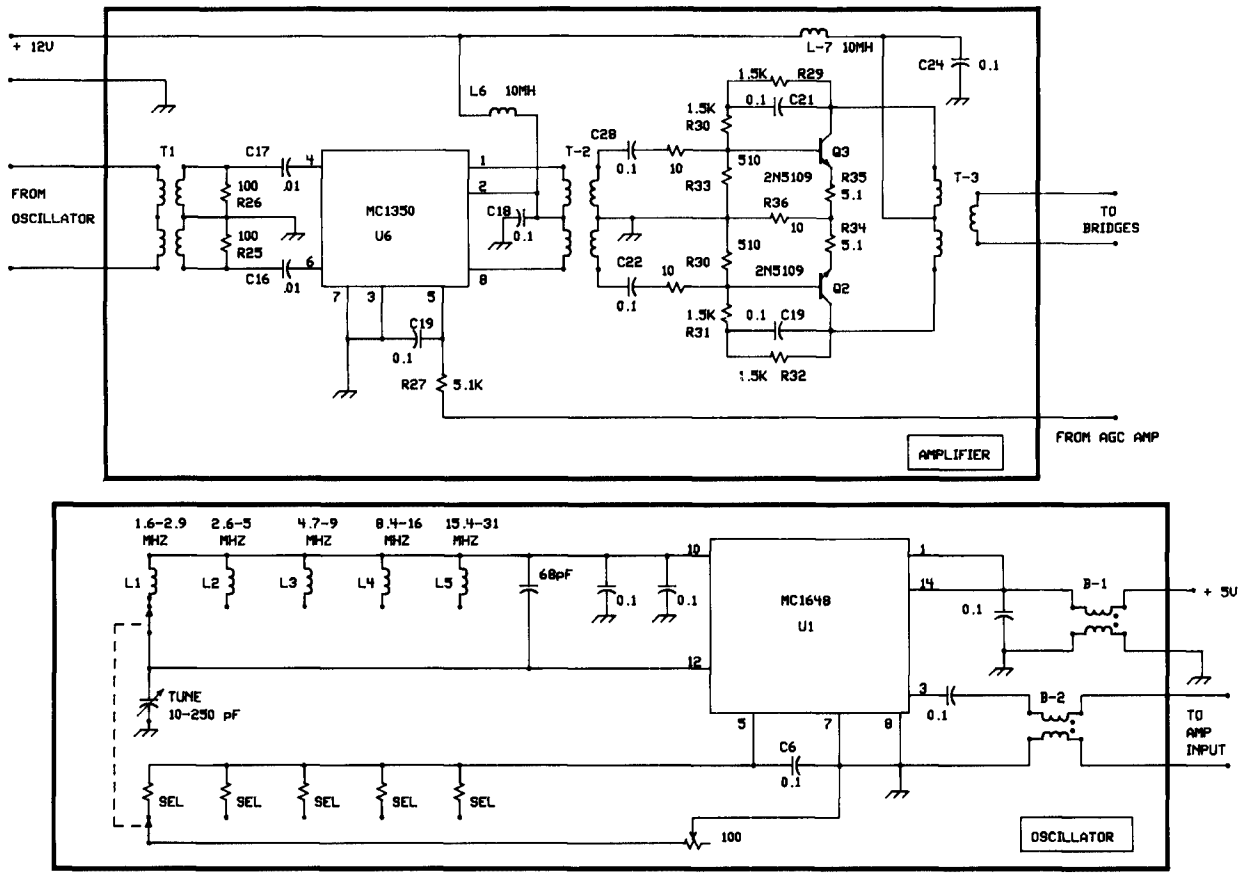
Housed in an aluminum enclosure, the oscillator provides frequencies from 1.6 to 30 MHz. The oscillator circuit uses a Motorola MC 1648 IC. This is an ideal component for a low-distortion oscillator with buffered output. As noted on the schematic of Fig 1, selected resistors are inserted between pin 5 and ground to improve the wave forms. Balun transformers B1 and B2 isolate the unit from power supply and ground thus reducing feedback effects from circulating currents in the chassis and power leads.

A power amplifier is included to improve SWR measurement accuracy and to provide useful power for radiation pattern measurements. An MC1350 IC is used as a driver for push-pull output transistors. This integrated circuit was designed for use as a video amplifier. Its broadband characteristics make it an ideal driver. Also, its excellent AGC characteristics are used to maintain a constant drive voltage to the bridge circuits. This feature permits making accurate SWR measurements without adjusting bridge amplitude levels.

Impedance Bridge

The impedance bridge, shown in Fig 2, is of the series differential type. The advantages of the series differential R-C bridge accrue from symmetrical and nearly ideal dial scale factors. An unlimited variety of possibilities are available through the selection of the value of the differential capacitors in combination with fixed-value shunts. Fig 3 shows the computed reactance values versus the rotational angle of the differential capacitor at 10 MHz for the instrument being described. Figs 4, 5, 6 and 7 illustrate other possibilities using different capacitor combinations. Comparisons with other R-C bridge types are covered in a previous *QEX* article.[1]

Schematically, the series differential bridge is very similar to the common series R-C bridge. A simplified schematic is shown in Fig 8, where it can be noted that a variable capacitor is substituted for the usual reference capacitor. This variable is differentially ganged to the calibration capacitor. The two capacitor sections must be insulated from each other. In another configuration a conventional differen-



TUNING INDUCTORS			
IND	TURNS	WIRE	CORE
L1	100	28	T80-6
L2	40	28	T68-6
L3	28	22	T50-6
L4	12	20	T50-6
L5	9	20	T50-6
NOTE: ALL CORES AMIDON			

TRANSFORMERS			
	PRIMARY	SECONDARY	CORE
T-1	6 TURNS NO 28 BIFILAR	3 TURNS NO 28 BIFILAR	AMIDON BN-43-202
T-2	7 TURNS BIFILAR	2 TURNS NO 28 BIFILAR	AMIDON BN-43-202
T-3	6 TURNS NO 28 BIFILAR	3 TURNS NO 28	AMIDON BN-43-202
B-1, B-2	6 TURNS NO 28 TWISTED 5 TURNS PER IN		AMIDON BN-43-202

Fig 1—Schematic of the oscillator/amplifier signal source for the antenna analyzer. The oscillator circuitry is enclosed in an aluminum box.

tial capacitor (one having a common conductive shaft connecting the rotors) could be used, but there are attendant problems related to undesired capacitance effects caused by the windings in the balun.

In my instrument, the differential capacitor was fabricated from two 176-pF single-gang variable capacitors using an insulated flexible-shaft coupler. It is important to minimize the capacitance to ground of each section. I mounted the capacitors on ¼-inch Plexiglas using plastic angle brackets. Dial calibrations at 10 MHz are shown in the photograph, Fig 9.

Shunting the 250-Ω variable-reference resistor with a capacitance equal to the combined capacitance of the circuit and the "unknown" connector is desirable from an accuracy standpoint.

Balun design and construction are most important. In order to maintain balanced voltages at the input terminals of the bridge throughout the useful frequency range, it is imperative that capacitive coupling between the primary and secondary be minimized. In my design, this is accomplished by using the outer braid of miniature coax cable as an electrostatic shield. Construction details for the balun can be observed from the schematic of Fig 2 and the photograph of Fig 10. This balun gives excellent results, among the best that I have used. The schematic also shows a two-winding balun (B-3) feeding the bridge balun. My instrument performs well without this extra balun.

SWR Bridge

The SWR bridge circuit shown in Fig 2 is the familiar SWR-resistance bridge. Meter scale-factor and zero-set functions are adjustable, respectively, by R23 and R18.

Power Supply

The power supply for the analyzer is shown in Fig 11. Primary power is supplied by a nominal 12-V NiCd battery. A –5-V suppy is developed by the

ICL7662 inverter. The battery charger was designed to prevent overcharging and includes current limiting. A battery monitor circuit is included to indicate a low-battery condition.

General Construction Notes

In noting the construction of my instrument, the reader is counseled to accept the final outcome as the result of a migration of ideas beginning with some rather ambitious concepts. While it is an effective unit having reasonable accuracy and is friendly to use, there are many opportunities for the experimenter to apply his own ingenuity and skills. My original plans included a phase-locked-loop signal generator, which was later abandoned. One would especially want to optimize the component arrangement of the impedance bridge. I had to compromise the location of the unknown (impedance) connector by placing it on the back panel in order to reduce lead lengths.

The digital frequency readout, origi-

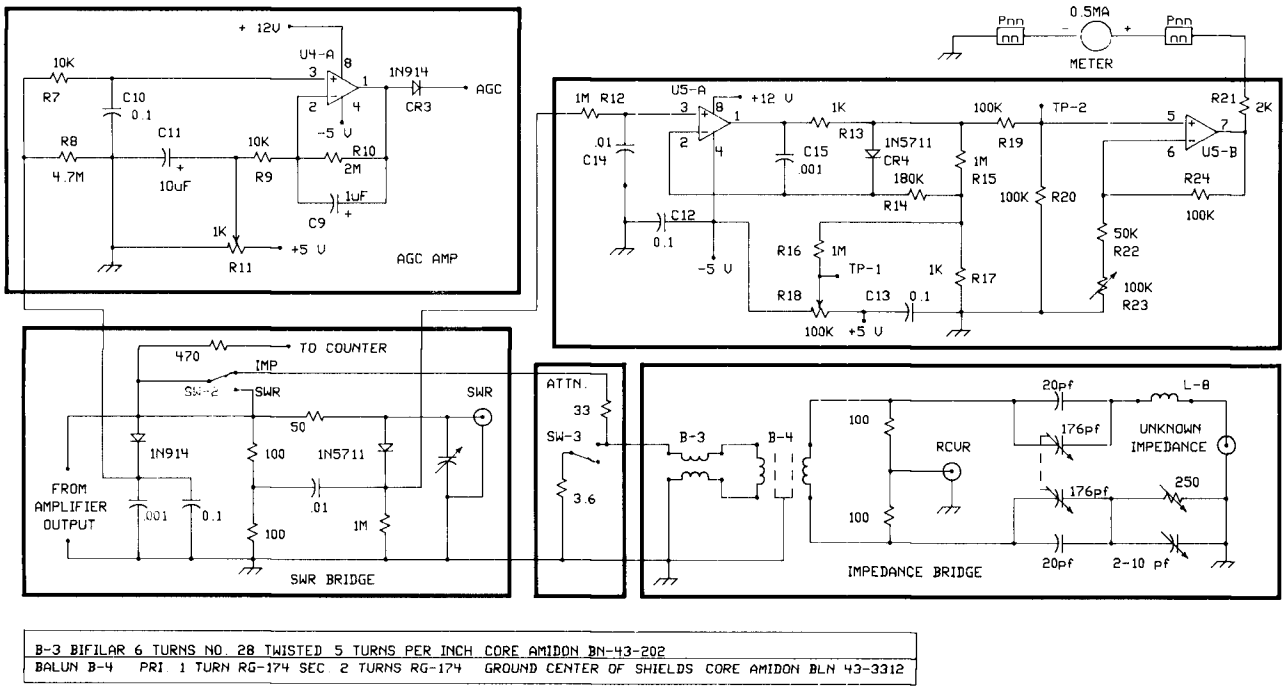


Fig 2—Schematic diagram of the measurement bridges and the AGC amp used to keep the applied signal level constant. U4 and U5 are LF412 dual op amp ICs.

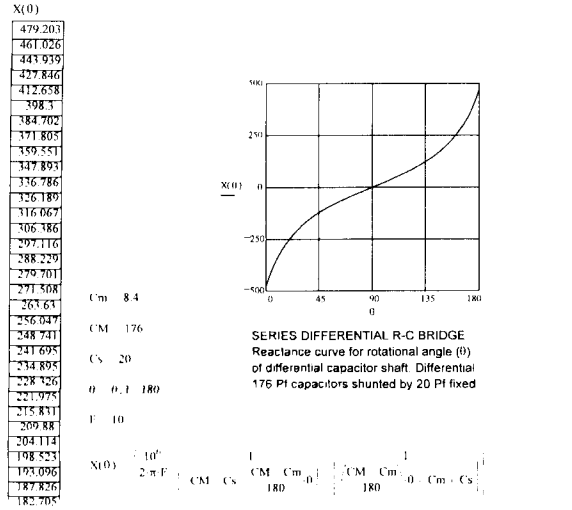


Fig 3

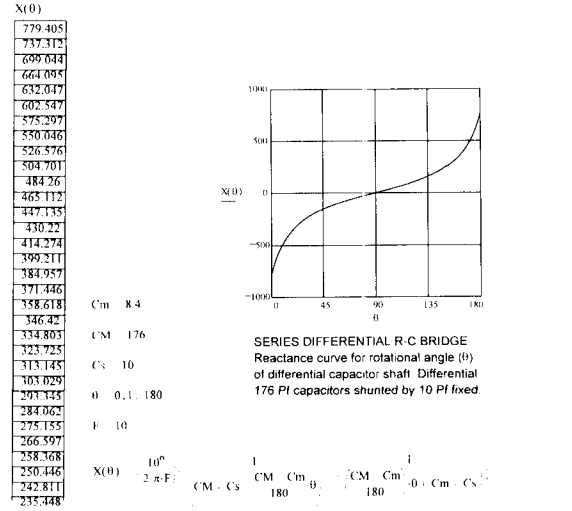


Fig 4

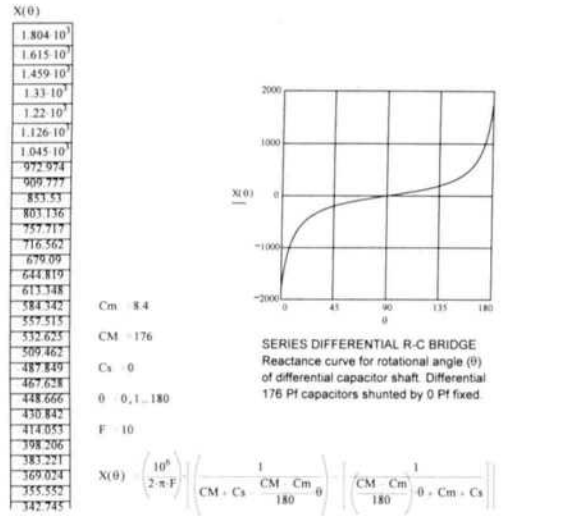


Fig 5

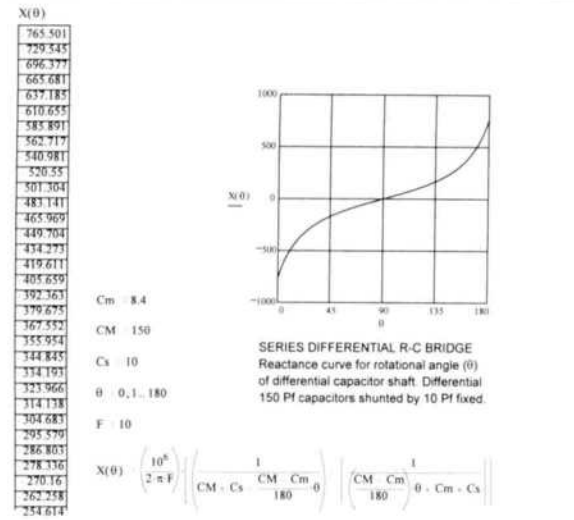


Fig 6

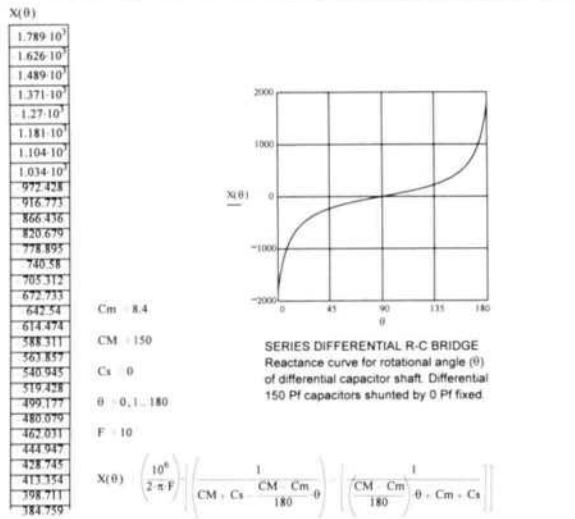


Fig 7

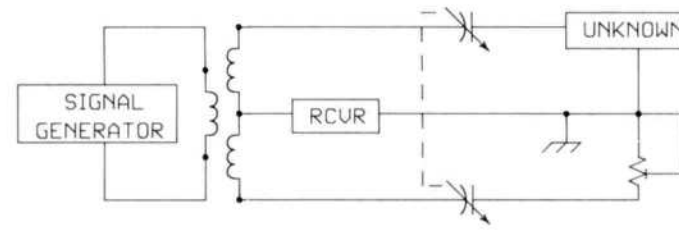


Fig 8—Simplified schematic of a series differential R-C bridge.

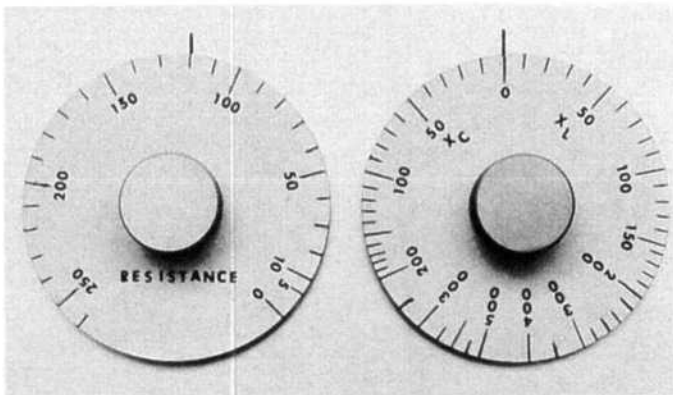


Fig 9—Close-up view of the impedance dial. The reactance dial was calibrated at 10 MHz. Scaling of the reactance values is necessary for measurements at other frequencies.

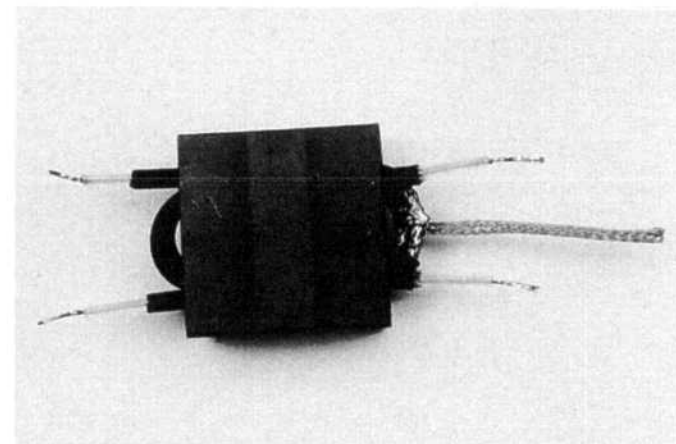


Fig 10—Construction details for the baluns used in the analyzer.

nally designed by Bainbridge, was purchased from A&A Engineering.[2] If one does not wish to design his own, other small counters are also available on the market.

I employed speed-reducing drives (Jackson Brothers) on both the resistance and reactance dials. In addition, a 2:1 gear drive (Jackson Brothers) drives the differential capacitor. This permits expanding the reactance dial to 360°. The torque requirement of the home-made differential capacitor is somewhat excessive, and the anti-backlash gearing is not too effective.

With the exception of the oscillator circuit, all of the normal circuit-board components, including the power sup-

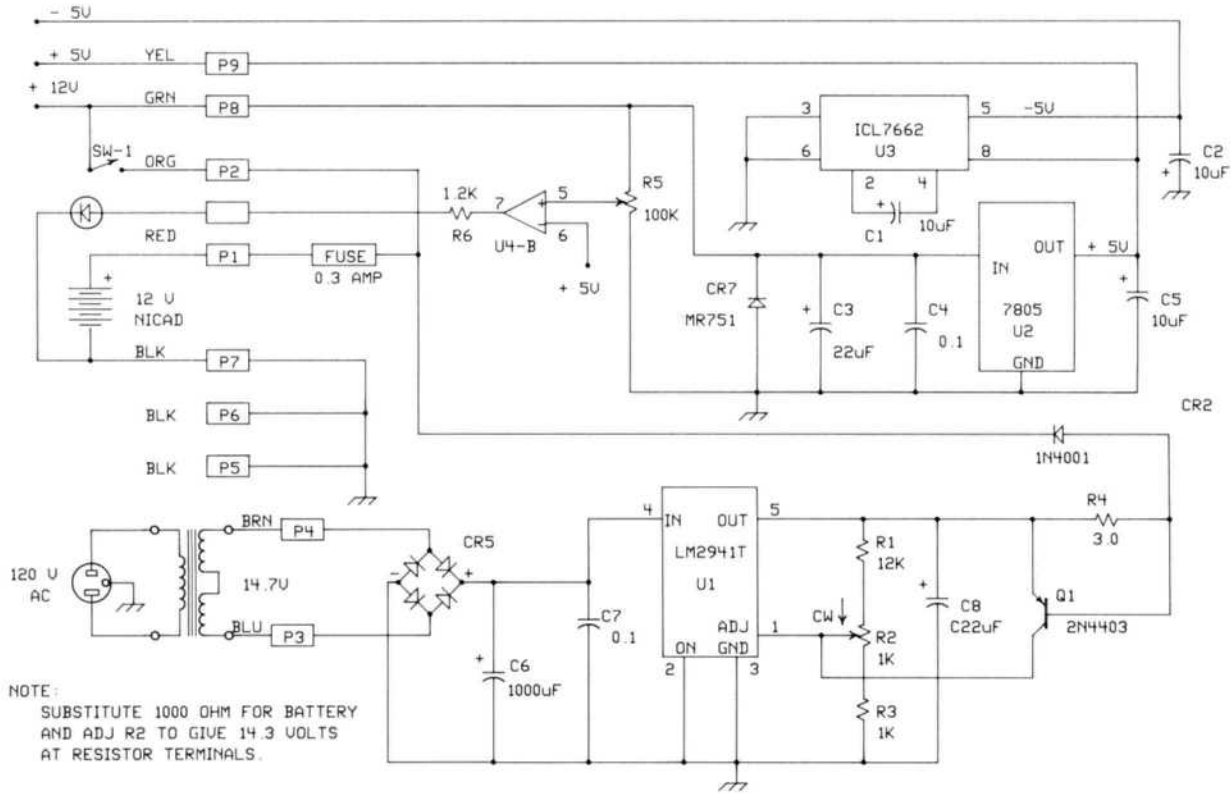


Fig 11—Schematic diagram of the analyzer power supply.

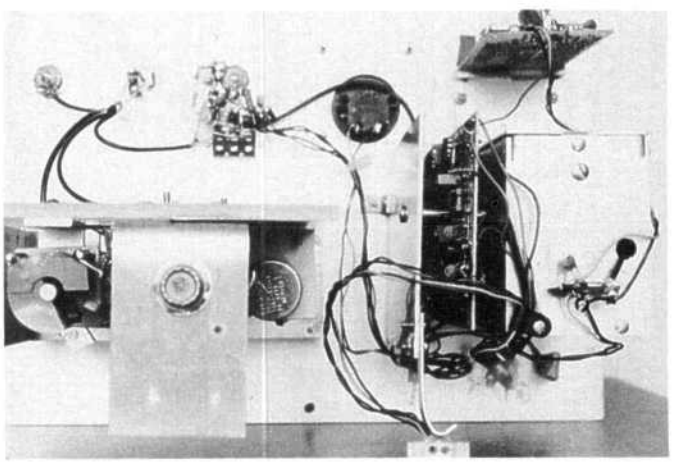


Fig 12—The analyzer interior as viewed from the rear. The oscillator is contained in the aluminum box at the lower right and the printed-circuit board assembly is to the left of the oscillator box. The frequency counter is at the top right.

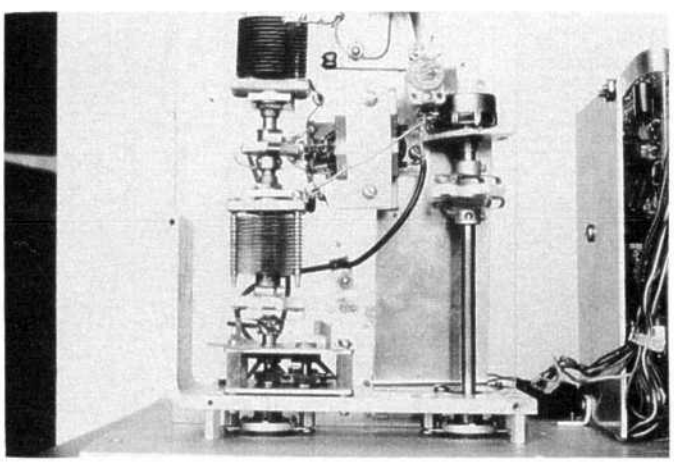


Fig 13—The impedance bridge circuit viewed from the bottom. Note the coupling of two single-gang variable capacitors to form a differential capacitor. The printed-circuit board assembly is at the right edge.

ply, are mounted on one circuit board. A&A Engineering etched the board from my artwork. Should someone be interested, I would be pleased to release A&A Engineering to furnish the boards.

Calibration

Resistance dials are easily calibrated using a digital VOM. It is most convenient to calibrate the reactance dial at one frequency and use a multiplier to convert the dial readings to the correct value at the measured frequency. My dial was calibrated at 10 MHz. The reactance dial can, of course, be calibrated using high-Q reactances of known values, but selecting or fabricating values to satisfy all the desired calibration points is not practical. A reasonably effective method using a calibrated length of coax has been described by Grebenkemper, and is also covered in the *ARRL Handbook*.[3,4] The low-loss 52-Ω cable designated may not be available. A table that allows substituting low-loss 50-Ω cable is included in a previous *QEX* article (see Note 1).

The effects of parasitic circuit capacitance can be ignored since in practice it mainly results in a minor shift in the dial scale. However, parasitic inductance must have compensation. This can be accomplished by inserting a small loop or length of wire in one arm of the bridge. My instrument required adding a small loop, shown as L8 in Fig 2. Many important details regarding parasitic reactances and other general considerations related to bridges are covered excellently by Grebenkemper and Popodi.[5]

Summary

In presenting this paper it has not been my intent to encourage duplication of the instrument, but to present ideas the reader can improve on to build similar instruments. One should consider the utility of a comprehensive instrument. And applications of the MC1648 oscillator IC could be of interest to some. But most of all, I'd like you to recognize the potential of the series differential R-C bridge for use in other impedance-measuring instruments such as noise bridges.

An ideal comprehensive antenna analyzer would include a built-in detector. This requires either a very pure signal generator or highly frequency-selective circuitry in the detector. In making antenna measurements at frequencies where there are nearby active signals, selectivity is most important.

Notes

[1]Tilley, Aubra E., "Characteristics of R-C Impedance Bridges," *QEX*, March 1994, pp 9-16.

[2]Bainbridge, Douglas. "A Low-Cost Frequency Counter," *QST*, February 1989, pp 21-26.

[3]*ARRL Handbook*, 1992 pp 25-35

[4]Grebenkemper, John, "Improving and Using R-X Noise Bridges," *QST*, August 1989, p 27.

[5]Popodi, A. E., "Building the Perfect Noise Bridge," *Communications Quarterly*, Spring 1993.

Programming a DSP Sound Card for Amateur Radio

Low-cost PC sound cards with DSP offer a great way to do signal processing—if you can figure out how to program them. That's just what KC7WW has done!

By Johan B. Forrer, KC7WW

This article gives a brief description of hardware and software development tools for one family of PC sound cards using a programmable digital signal processor (DSP). Such low-cost programmable DSP hardware is of great interest to those experimenting with DSP algorithms. Examples of possible applications in Amateur Radio include: audio signal processing for noise/interference cancellation, CW filters, EME weak-signal processing, DTMF decoding and repeater controllers, modulation and demodulation of coherent CW, AM, FSK, PSK, or QAM signals, and active demodulators in radio receiving equipment.

The DSP in a sound card provides a flexible and highly efficient all-digital front-end, while the PC plays an important supporting role for protocol implementations, mass-storage, and the user interface. These applications require a highly efficient (fast, consistent and reliable) communication path between the DSP front end and the PC application software. The main requirements for development of applications using a sound card include the availability of development software, low-level system utilities, and good PC debugging tools.

The approach taken in this article relies on known DOS programming techniques that most PC programmers are familiar with. Since the user interface is written in C++, it is also possible to port it for use with other operating systems such as Linux.[1] The objectives of this project are to provide:

- a set of re-usable system utilities (templates) to load code and communicate with the sound card's DSP;
- software development tools to assemble source code and produce DSP loader compatible output; and
- a simple example application to illustrate use of the software tools.

The Development of Sound Cards

Sound cards, like CD players and high-resolution color monitors, are often added to PCs for multimedia applications. Not only do these provide support for popular computer games, but they also may be used in audio/visual software, such as embedded sound clips, voice recognition and computer conferencing.

PC sound cards have experienced dramatic development over a relatively short period of time. The first generation of 8-bit cards proved, with their low sampling rates, to be limited in quality and capabilities. These were soon superseded by second-generation sound cards using 16-bit technology—matching CD audio quality. Such sound cards, at a minimum, are capable of producing 16-bit stereo sound at sampling rates of at least 44 kilosamples per second (ksps).

Such quality does not come without a price. The demand for processing of high-speed, 16-bit sampling may strain an average PC's resources. Even a modest application such as simple acquisition and reproduction of a signal can consume megabytes of storage and require a PC with a fast processor. Such volume and processor speed constraints imply that very little opportunity remains for performing much innovative signal processing on the PC. However, this has not prevented some experimenters from developing innovative amateur-radio applications. An example is François Jalbert's CW decoder.[2] And,

26553 Priceview Drive
Monroe, OR 97456

[1] Notes appear on page 15.

as shown in *QEX* recently, low-cost FFT tools based on simple sound cards can also be of great use.[3] Besides these amateur-radio applications, there literally are hundreds of software programs available for recording and playback of audio signals.

To escape the limitations of having the PC do all of the processing, third-generation sound cards now include DSP hardware. The DSP functions as a coprocessor that is independent of the PC's processor and memory. Not only does the DSP system on the sound card run independently of the PC's CPU, it is optimized for executing the kinds of algorithms used to process sampled signals. The result is a vast increase in signal-processing capability over older systems. There still are many cases, however, where the PC needs to handle the sampled data, such as for storage and playback. So the sound card must provide efficient interprocessor communications paths to handle large amounts of data at high rates. Such architectures allow efficient signal processing, such as real-time data compression and decompression, and the ability to emulate the ever-evolving set of industry-standard signal recording formats.

Choices and Options for DSP Sound Cards

Successful development of applications on a DSP by the amateur depends greatly on the ease of accessibility of the hardware. Until recently, this has been a major problem because of the lack of available technical information, especially in cases where proprietary hardware elements, such as application-specific integrated circuits (ASIC) or special analog-to-digital converter (A/D) chips, are used.

Besides hardware, special real-time software is required, such as device drivers or, in some instances, real-time DSP operating system software when multitasking DSP applications are involved. Software development toolkits are often available from the sound card manufacturer, but at prices that put this option out of reach of the average amateur.

These factors carry a lot of weight when selecting a card—often more weight than the performance capabilities of the DSP chip. There are several DSP sound card architectures on the market now, and I learned about several possible options through discussions on the Usenet news groups on Internet. A full discussion of all the various offerings is beyond the scope of this article, which will concentrate on the particular architecture I chose to explore. I concluded that one particular card, the Cardinal Digital Sound Pro-16, met most of the objectives.[4] But I realized that, as would be the case for most of the DSP sound cards, further development was to be without much assistance from the card manufacturer. I would have to figure out the system for myself. This proved to be a rather interesting experience as the details unfolded. Many hours were spent studying the hardware and conducting trial-and-error experiments. Fortunately, the folks at Analog Devices DSP Applications provided pointers in the right direction.[5] I owe them a debt of gratitude.

Besides the DSP system hardware, the chosen hardware includes a standard Microsoft *Windows* "sound port," MIDI music interface, CD-ROM interface, and appears to be compatible with most industry sound formats. Besides, it is reasonably priced and readily available though several consumer electronics dealers as an off-the-shelf item.

The Analog Devices PSA Chip Set

The Cardinal Digital Sound Pro-16 is designed around the Analog Devices Personal Sound Architecture (PSA) chip set. The PSA chip set includes three chips from Analog Devices: the ESC614 ISA (PC) bus ASIC, AD1848 stereo codec and ADSP-2115 DSP. This chip set is also used in other sound cards.[6,7] Some cards, however, such as the Orchid Soundwave 32, use different input-source selection mechanisms or numbers of wait states. The example programs described here won't work correctly on these cards, although the needed changes should be minor.

The heart of the PSA chipset is the ESC614 ASIC. This chip controls several key components of the sound card. A simplified functional block diagram of this ASIC is shown in Fig 1. Communication between the PC and the ESC614 is through PC I/O ports with addresses in the range 220H (alternatively 240H, by external jumper selection). Six registers are used for configuring the functionality of the DSP, *Windows* sound system (WSS), Sound Blaster emulation, CD-ROM, and MIDI.[8,9] For the purposes of this article, only the DSP configuration requires special attention. The DSP configuration register allows software selection of interrupts and DMA channels that the DSP may use. Four 16-bit registers in the address range

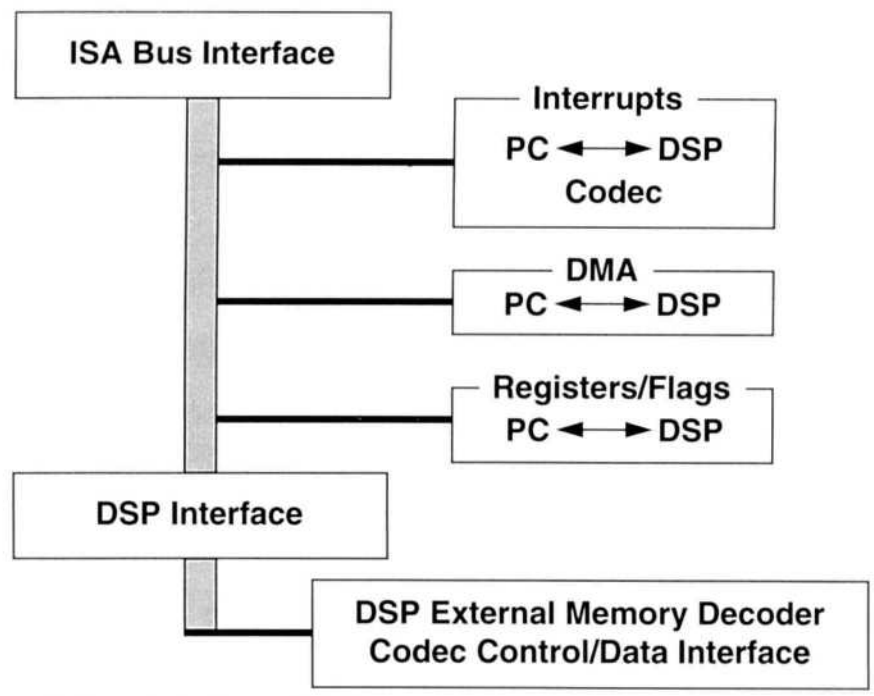


Fig 1—A simplified functional block diagram of the ESC614 ASIC. The ASIC greatly simplifies hardware and software complexity. It allows easy access to the ISA bus and takes care of the DSP's interface with external memory and the codec.

220H to 23AH (alternative 240H to 25AH) allow the PC to communicate with the DSP through the ASIC.

The AD1848 stereo codec is the interface to the analog world. It contains dual 16-bit analog-to-digital (A/D) and digital-to-analog (D/A) channels that are capable of sampling rates up to 48 ksps. Several programmable sampling rates are available, each with automatic scaling anti-alias and reconstruction filters. This codec chip was developed to be fully compatible with the Microsoft *Windows* Sound System WSS. A 28-page data sheet describing the AD1848 is available from Analog Devices and is a prerequisite for interpreting the software described later in this article.[10]

Through the use of the ESC614 ASIC, the codec can be connected directly to the ISA bus, thus bypassing the DSP. This allows for a high degree of WSS compatibility. But the codec also may be placed under DSP control for efficient, low-overhead data acquisition. This is the method used in the software described here.

The third element of the PSA chip set, the ADSP-2115 DSP, is a member of the Analog Devices 21xx family of fixed-point chips. The Cardinal card DSP runs at 16.7 MHz (60 ns ticks). Some other PSA sound cards that use a more recent 2115, the revision 1-1 chip, run at 20 MHz (50 ns ticks).[11] Technical specifications, hardware and software examples, and the programming model for the ADSP-2115 are described in the ADSP-2100 Family User's manual (see note 5).

The ADSP-2115 contains 1024 24-bit words of on-chip program memory and 512 16-bit words of on-chip data memory. The sound card provides for an additional 15-kword, 24-bit off-chip zero-wait-state program RAM and for two 8-kword, 16-bit bank-switched zero-wait-state data RAMs. However, external RAM requires additional fetch cycles due to the multiplexed bus I/O scheme used for off-chip memory accesses. This calls for careful selection of on-chip and off-chip memory use. However, this amount of additional available memory could easily provide for implementation of quite challenging applications.

The ADSP-2115 provides programmable wait-state generation for external devices. In the case of the AD1848, two wait states are required. Off-chip memory and the ESC614 ASIC all run using zero wait states.

DSP Development Software

Developing DSP applications for the sound card requires some familiarity with DSP algorithms, programming experience with the DSP, familiarity with low-level PC fundamentals, and programming the PC, preferably in some high-level language. The software tools described in this article cover some of the low-level PC requirements. Some of this detail will become more clear when the enclosed examples are analyzed. DSP algorithms are found in many technical and academically inclined sources and are outside the scope of this article.

The set of development tools created for this project consists of an assembler for the ADSP-2115 and a bootstrap formatter. The example programs include a simple DSP bootstrap monitor, a DSP application module and a user interface application written in C. An overview of the specifics of each of these follows.

The SpAsm21 *Assembler*

The assembler is a port of James Brundell's program, *SpAsm*, a public-domain 2105 assembler.[12] Although this assembler was originally written for the Macintosh, I've ported it to the PC for this project. My thanks go to James Brundell for making his work available.

This simple assembler has been used for quite ambitious projects and has proven to generate reliable code. But users should be aware that several limitations exist. For example, *SpAsm21* only implements a subset of Analog Devices' assembler directives. The omission of memory allocation, for instance, makes that the responsibility of the programmer—an especially tedious task when several circular buffers of various lengths are required. The usefulness of this assembler could be further improved, especially its error-reporting and listing format. But as public-domain software, the assembler is quite useful, especially since I found no other free assemblers available for the ADSP-2115!

SpAsm21 takes as input an ASCII source file and produces two output files: a .CDE object file and a .LST listing file.

The Bootstrap Formatter: CLOAD

A programmable DSP needs some means of loading DSP application software into the DSP memory, and the ADSP-2115 includes special ROM code designed for this purpose. Loading is a two-step process: first, a small program is loaded into on-chip DSP memory using the special DSP ROM code, then the loaded bootstrap program is used to load and execute the DSP application code. The loading of the bootstrap program is controlled by the ESC614 ASIC. The method used requires that the bootstrap code be organized and formatted in a particular way. *CLOAD* was written to convert the output of the *SpAsm21* assembler to the required bootstrap format. It takes a .CDE file as input and produces the .LD format file required by the ADSP-2115 ROM loader.

Loading and Executing the Application

After the DSP bootstrap process is completed, control passes to a section of the bootstrapped code where it awaits further commands from the PC host. Communication between the DSP and the PC is accomplished by the DSP code polling a special memory-mapped register, the DSP status register. Similarly, the DSP can signal the PC via a special memory-mapped register. This call-and-answer handshaking method allows the implementation of several functions such as reading and writing program and data memory locations and initializing hardware, such as the timer and A/D. These memory-mapped registers are part of the ESC614 ASIC. As far as the DSP is concerned, these registers appear as 16-bit data memory; on the PC side, they appear as a 16-bit I/O port.

The user interface program running on the PC, as one of its initialization tasks, loads the DSP application code into the DSP's memory. It takes a .CDE input file and derives DSP program target addresses and instructions from the contents of the file. Each input line in the .CDE-format file contains a 14-bit address followed by a 24-bit instruction that is to be loaded at the specified address. The actual transfer of the address and instruction data to the DSP is through call-and-answer functions of the bootstrap code. The last function that the loader program on the PC performs is to call a bootstrap function that transfers control to the just-loaded DSP application. This often is set to start at a fixed location in memory.

Register Summary

The DSP interface appears to the PC

as a number of 16-bit I/O registers, located at port addresses from 220H to 23AH, or alternatively 240H to 25AH, depending on a jumper setting on the card. The ASIC further provides for the use of PC interrupts—the selection of which ones is programmable.

ID Version Register (224H)

One of the first responsibilities of the PC interface is to determine whether communications with the DSP is possible. This is accomplished by reading the ASIC ID at port 224H. Success is indicated by reading an ASCII "E" (45H) in the upper byte, while the least-significant byte contains the ASIC version number, ie, 01H for version 1.

Status/Control Register (222H)

Once the presence of the DSP card is established, it most often is necessary to perform a boot of the DSP system. A hard reset may be asserted by toggling bit 13 in the control register. The DSP is also forced into the reset state by a PC hardware reset. This is a known state for the DSP; it is ready to perform a bootstrap.

Data I/O Register (220H) and Status Register (222H)

Data between the DSP and PC may be passed via a 16-bit register located at port 220H. During the bootstrap process, the DSP ROM requires data to be presented in a fixed format and order. After the last bootstrap word has been written, the bootstrap ROM will transfer control to the bootstrap code according to the startup vector that was loaded as part of the code. The purpose of the bootstrap code is to extend the functionality of the DSP, especially to prepare for the loading of further DSP code. At this level, port I/O is done using polling.

The PC performs this polling by reading the status register at port 222H and testing bits 14 and 15. Bit 15 (WE) indicates that the DSP has read the last data written by the PC to 220H. Bit 14 (RF) indicates that the DSP has written new data to 220H. Bit 13 (IRQ) is used when the DSP is programmed to generate a PC interrupt.

From the DSP, this register appears at memory address 3000H.

Configuration Register (230H)

When the PC's interrupt mechanism is to be used to service the DSP, the configuration register first must be set up. The actual interrupt used is programmed by writing one of the codes of Table 1 to the configuration register at port 230H.

Interrupt Acknowledge Register (224H)

The PC interrupt service routine has to clear any DSP-initiated interrupts by writing to the interrupt acknowledge register at port 224H in addition to writing the usual end-of-interrupt (EOI) instruction, code 020H, to the PC's programmable interrupt controller (PIC) at port 020H. It is of course necessary to set up the appropriate PC interrupt vector and arm the interrupt in the PIC before DSP interrupts can be recognized. The programmer should also be aware that interrupts above IRQ7 use the PC's chained PICs, requiring that an EOI be written to port 0AH as well.

WSS Emulation Register (232H)

This register specifies the address through which the codec is to be accessed from the ISA bus. If any configuration of the codec is to take place from the PC side, the port address of the (WSS) emulation register, located at 232H, must be written first. Once this is done, the identity and status of the codec can be then be established. The reason for this is that it often is easier to configure the codec's operating mode and sample rate from the PC side before transferring control to the DSP. Once the DSP gains control of the codec (by writing a control code to a special memory-mapped register), the codec is effectively disconnected from the ISA bus.

PC port address 530H is typically used for WSS emulation. Once the

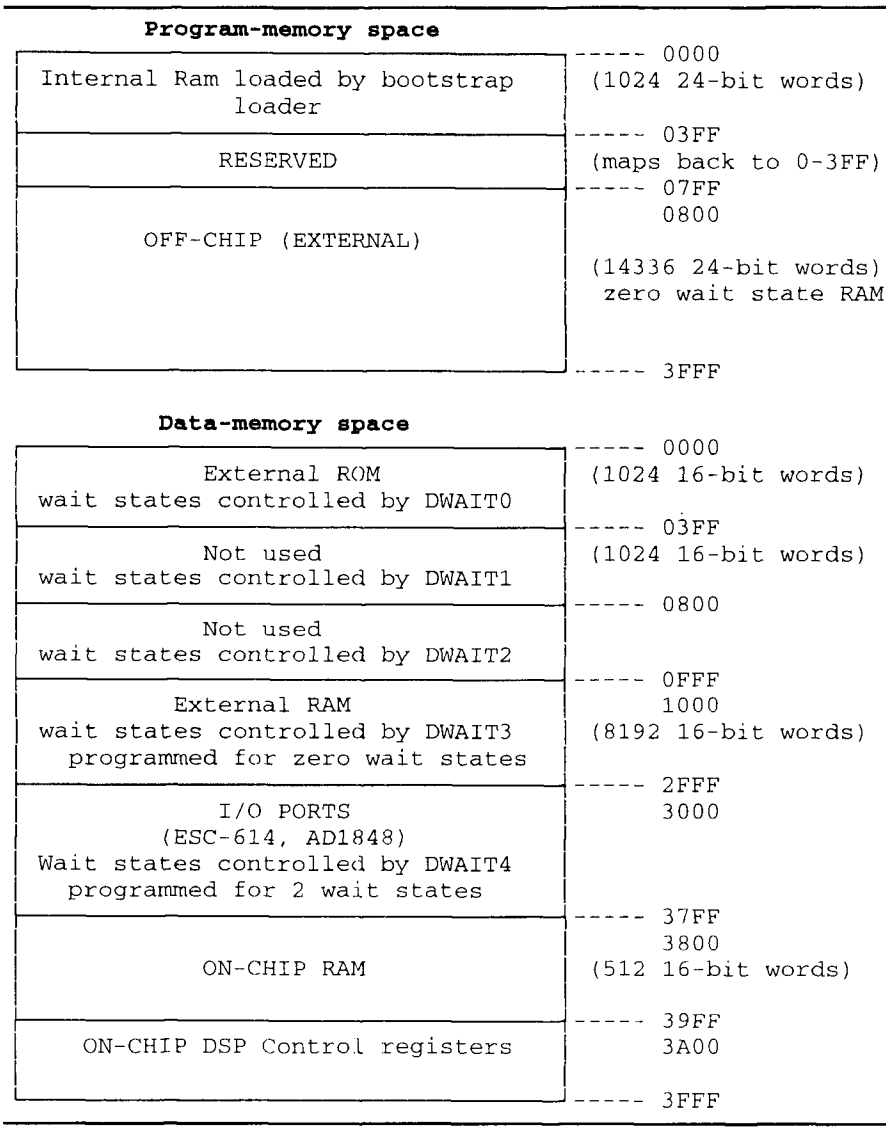


Fig 2—Program and data memory layout for a PSA sound card.

WSS emulation address is set, a series of registers starting at that port address form a "window" into the codec. The use of WSS registers is discussed at length in the AD1848 codec data sheet, available from Analog Devices (see note 10).

DSP Memory-Mapped Registers

The DSP uses memory-mapped I/O to access the off-chip peripherals such as the AD1848 codec and the ASIC-emulated I/O data ports. It must be noted that not all these peripherals are capable of running at zero wait states. The ADSP-2115, however, allows programmable wait states to be associated with certain blocks of its external memory space. A block of addresses are associated with the ASIC and codec. These are required to run at two wait states, while the external static RAMs are programmed to run at zero wait states. The actual layouts for program and data memory are shown in Fig 2.

Interrupts

After a reset, the DSP and ASIC start in a known state with no interrupts set up. Communication with the DSP is then possible after the bootstrap process only via polling, as described above. Such a polling approach is effective for simple applications requiring a small amount of processing overhead. For more advanced applications, such as HF/VHF modems, minimal service latency is crucial to meet timing demands and must be performed via the programmable interrupt mechanism. When very large data blocks need to be transferred at high rates, the programmable DMA mechanism, in conjunction with the interrupt mechanism, must be used. DMA usage is not discussed in this article.

Since there are two processors involved—the DSP and the PC—there actually are two different interrupt mechanisms involved as well.

DSP System Interrupts

DSP system interrupts are generated by devices connected directly to the DSP, such as the AD1848 codec, the DSP's internal timer, and the ESC614 ASIC, which generates interrupts initiated by the PC. The DSP handles these interrupts by providing a series of interrupt vectors located at the start of program memory. The DSP is made aware of which interrupts are being used by setting up the interrupt control (ICNTL) and interrupt mask (IMASK) registers (see the ADSP-2100 Family User's Manual for further details).

Table 1—PC Interrupt Configuration

Code	*Interrupt*
00H	Interrupts disabled
08H	IRQ3
10H	IRQ5
18H	IRQ7
20H	IRQ9
28H	IRQ10
30H	IRQ11
38H	IRQ12

Table 2—ASIC IRQ2 Enable Register (31C0H)

Bit	*Interrupt source*
15	PC has written new data
14	PC output data buffer empty
13	AD1848 service request

All off-chip interrupt sources are handled by the ESC614 ASIC and are vectored via DSP interrupt IRQ2. (Note: this is the DSP chip's IRQ2, *not* the PC's IRQ2.) Any or all of the three possible interrupt sources can be enabled by writing a binary 1 to the appropriate bits in the IRQ Enable register, at memory address 31C0H. The layout for this register is shown in Table 2.

When an IRQ2 interrupt occurs, the source of the interrupt is determined by reading the register located at memory address 31C0H; each interrupt source corresponds to a bit shown in Table 2. Interrupts due to bit 15 are cleared by *reading* the I/O register at memory location 3000H, while interrupts due to bit 14 are cleared by *writing* to the I/O register at that address. Interrupts due to codec service requests must be cleared by writing to the right channel codec DMA register. The procedure for doing this is shown in the programming example.

When the DSP wishes to interrupt the PC, it does so by setting PCIRQ, bit 12, in the DSP control register at memory address 3008H. This bit must then be cleared by the PC, as will be described. (Also see the programming example.)

The main interrupting sources of the DSP system for amateur-radio applications are the timer and the AD1848 codec. Usually, the timer will be programmed for an interrupt rate equal to a modem's symbol rate, while the codec is programmed to interrupt at the sample rate. For efficient A/D and D/A operations, the codec is configured to run in DMA mode under DSP control. There is no actual DMA involved in the data transfer between the DSP and the codec, due to the speed of the DSP. Rather, D/A and A/D operations take place during each interrupt. This way, the codec is capable of 16-bit, full-duplex stereo (2 simultaneous channels being read and written to at each interrupt) at 48 ksps.

The AD1848 is a complex device to program. It is often easiest to perform all codec initialization while the codec is connected to the PC. Then, when the application is started, the codec is switched from the ISA bus to the DSP, where it runs under the DSP's interrupt control.

An important point to remember here is that the AD1848's registers are 8-bit registers. The DSP, however, uses 16-bit I/O. Therefore, all data must be left-justified: passed in the most-significant byte of the DSP's 16-bit words. For further details on setting up the AD1848 codec, see the programming example as well as the data sheet.

PC Interrupts

Handling interrupts from the DSP to the PC requires that the PC's interrupt system be programmed appropriately. This is a fairly routine low-level PC programming task, but it needs careful attention to detail as one small error usually has catastrophic consequences, the least of which would be a frozen system.

The first step is to decide which PC interrupt the DSP system is to assert. This may be difficult, as spare interrupts are often a scarce commodity. The range of PC interrupts the DSP system can access is shown in Table 1. Once the configuration register is programmed to select the PC interrupt, the appropriate PC interrupt vector must be prepared to point to the user's interrupt service routine (ISR) and the PC's programmable interrupt

Table 3—Status Register (port 222H)

Bit	*Use*
15	DSP has read data
14	DSP has written data
13	DSP generated interrupt

controller (PIC) must be programmed to arm the appropriate interrupt source. All this initialization, of course, must be performed while the PC's interrupt system is disabled. For further details please see the programming example.

When the DSP signals an interrupt to the PC, the PC's PIC will interrupt the PC, and control will vector to the user's ISR. The ISR will then need to read the status register at port 222H, the layout of which is given in Table 3.

The PC ISR is the workhorse of the user's application. It is where all timing-related processing takes place. Besides handling timing-critical functions, it also forms the interface to the user-interface code. Entry into the user's ISR is via the PC's vectored interrupt system. Before resuming normal processing, the PC acknowledges the interrupt by writing to the IRQ acknowledge register at port 224H. This is required in addition to writing the end-of-interrupt (EOI) to the PIC (that is, writing 20H to port 20H, and also to port 0AH for IRQs above 7).

DSP Application Example: A 250-Hz CW Filter

The use and programming of the Cardinal sound card are further illustrated by a simple example: a 250-Hz-wide, 127-tap FIR filter, centered at 400 Hz and using a sample rate of 5512.5 sps. To aid in this illustration, the timer is programmed to interrupt the PC approximately once per second. The PC acknowledges the interrupt and prints a period to indicate the event.

The main components for this application include the bootstrap monitor, the DSP application code, and the PC user-interface code and ISR.

Bootstrap Monitor: CWMON.DSP

The purpose of a bootstrap program is to serve as a means of getting a small piece of code loaded that, once executed, provides a means for loading larger, more extensive program code. Generally, the bootstrap program provides just the bare minimum of functionality. The bootstrap monitor for the example application follows these guidelines. The source code is contained in module CWMON.DSP. This module must first be assembled by *SpAsm21* to obtain a .CDE object module, then converted to an .LD loadable format by the *CLOAD* program.

The DSP program code usually starts with memory allocation, followed by variable and buffer naming. For the ADSP-2115, the interrupt vector space forms the first actual executable code. In the sample application, the DSP startup vector, IRQ2, and the timer interrupts are to be used and thus must be set up.

The example bootstrap program is relatively straightforward. It consists of DSP initialization, two ISR's, and the call-and-answer idle loop.

During initialization of the DSP hardware, there are a few important steps that need careful attention. In particular, the behavior of the interrupt system, wait-state generation, and enabling the appropriate interrupts.

The PC and DSP uses a call-and-answer mechanism for asynchronous communications. A number of function codes are defined for this purpose. These include functions to read and write data memory, to read and write program memory, and to control the state of different applications. During program development, this set of codes would need to be further expanded for additional functionality.

In order to use the timer under interrupt control, the timer's counter registers need to be loaded with appropriate constants, its interrupt vector needs to be set up, and the timer needs to be enabled to start counting. In the accompanying example, the interrupt vector points to a simple ISR. This ISR only sets bit 13 in the IRQ control register, to assert an interrupt on the appropriate PC interrupt line. Of further interest is that the alternate register set of the ADSP-2115 is used while ISR code is executing. This provides for rapid context switching. Note that the timer is started and terminated by means of two call-and-answer function codes.

DSP applications rely on sampling a signal at some steady sampling rate. In this example, this is achieved by programming the AD1848 codec for a sample rate of 5512.5 sps. In order to process the sampled digital signal through the FIR filter at this rate, the AD1848 is programmed to interrupt the DSP via its IRQ2 interrupt each time new data samples are ready. The IRQ2 vector thus must point to the first code instruction of the user's DSP application. Note that in the example, the vector actually points to another module outside the bootstrap code space. A hook, an address named *sound_port* is defined in the data definition area for this purpose.

In order for this code to start functioning, the AD1848 must be switched from the ISA bus to DSP control. These functions are also performed by two call-and-answer function codes.

DSP Code for the CW Filter: CW250.DSP

The DSP application is loaded separately from the bootstrap by code in the user interface using read/write call-and-answer function codes. The source code for this module is contained in CW250.DSP. This program must be assembled by *SpAsm21* to obtain a .CDE object file that the user interface program will load into the appropriate DSP program memory space.

The actual DSP application is embedded in the AD1848 ISR. One logical constraint is that the application code must be able to execute to completion before the next interrupt comes along. In the case for the ADSP-2115, a considerable number of processing "ticks" are available, and this example takes a small fraction of the alloted time.

The introductory part of the code for the DSP application follows along similar lines as that of the bootstrap program. The entry point for the actual executable code was provided by the "hook" address in the bootstrap program.

The first order of business for the application code is to service the AD1848. Note that both the output and input ports of the codec require attention in this instance.

The remainder of the application involves the implementation of the FIR filter. In order to obtain the optimum execution speed of the ADSP-2115, multiple instructions are executed in parallel though a hardware pipeline. This requires that data and coefficients reside in different memory banks; the data is located in data memory, while the coefficients are located in program memory. Of further importance is that these memories be on-chip rather than off-chip, as off-chip memory access exacts penalties in the form of additional fetch cycles. These speed considerations, however, may not be crucial when executing other, noncritical parts of the DSP code.

PC User Interface and ISR: CW250.CPP

Most DSP developers agree that the application code of a project is often more involved than the algorithms residing in the DSP. This example proves the point. It is for this reason

that a high-level language is a more appropriate development platform for the PC code. Not only is there a wide choice of compilers, editors and debuggers, but also a greater pool of available software developers to participate in the effort!

Borland C++ 4.0 was used in the example application for a simple user interface. The program source is contained in module CW250.CPP. A header file, PSA.H contains sound card hardware-related definitions.

The first order of business for the user interface program is to find the PSA card and install the access port for the AD1848, or WSS "sound port." Several function calls are provided for this purpose. These function calls are packaged in two precompiled object modules, PSA1.OBJ and PSA2.OBJ.

Once the PSA card has been initialized, the bootstrap loading process is initiated and verified, to show that the DSP actually is up and running with the bootstrap code. This is followed by loading the user's DSP application code before starting the application on the DSP side.

To illustrate the multitasking nature involved in many amateur-radio applications, such as modems, a DSP task must be doing processing at audio sample rates, while the user application is simultaneously operating at symbol rates. In this example, a timer running on the DSP side is programmed to interrupt the user-interface program at regular intervals while the audio filter DSP application is active as well. Thus, this example shows the necessary steps involved to set up and activate interrupts from the DSP side, as well as how to hook and process interrupts on the PC side.

The example program uses IRQ7, the printer interrupt. The content of the IRQ7 vector is first saved, then reprogrammed to point to a new ISR within the user-interface program. The PC's interrupt mask register (IMR) is then saved before arming interrupt 7, in order for the PC to start responding to sound-card interrupts on the IRQ7 line. Once all this has been successfully completed, a message is sent to the DSP to start the timer that will actually initiate the sending of these timer interrupts.

Before starting the user's DSP application, the AD1848 must be programmed, while still connected to the ISA bus. This may of course be done instead in DSP code—it's just a little easier to perform from a high-level language. The new set of parameters for the AD1848 are contained in the array *initstate[18]*. Refer to the AD1848 data sheet for further explanation of these parameters.

Before terminating, the user application must restore all interrupts and the status of the IMR. It also is a good idea to leave the DSP in a state such that it is ready to accept new application code for running subsequent programs. This orderly shut-down procedure is also implemented using call-and-answer functions.

Software Availability

The software described here can be downloaded from the ARRL BBS (203-666-1578), or via Internet from ftp.cs.buffalo.edu, in the pub/ham-radio directory, The file name is QEXSNDCD.ZIP. Source code is provided for CWMON.DSP, CW250.DSP, CW250.CPP and PSA.H, which are listed on the following pages. The modules PSA1.OBJ and PSA2.OBJ, required for linking the user interface, are also provided. Note that the large code model must be used when recompiling the user-interface program.

The assembler, CLOAD and related files are available in the file PSATOOLS.ZIP.

Summary

This article provides some insight into use of a programmable DSP sound card for amateur applications. It is evident that there are a number of hardware and software issues that need to be addressed to sucessfully do such programming. However, as shown, these may be encapsulated to hide some of the complexity, thus allowing DSP experimenters to enjoy what they like doing best: developing DSP applications.

A useful toolkit is provided here, but further functionality could be provided to be appeal to a greater audience. Depending on the level of reader interest, this will receive attention in the future.

Notes

[1]*Linux* is a popular public domain UNIX look-alike for 386/486 personal computers.

[2]FFTMORSE, available on Compuserve and via Internet on oak.oakland.edu.

[3]Price, Harold E., "Digital Communications," *QEX*, December, 1993, pp 13-15.

[4]Cardinal Technologies, Inc. 1827 Freedom Road, Lancaster, PA 17601, USA.

[5]Analog Devices, DSP Applications, One Technology Way, PO Box 9106, Norwood, MA 02062-9106. For literature, contact Analog Devices Literature Center at 617-461-3392. Analog Devices also sells professional software tools: Part no. ADDS-21xx-DSW-PC (assembler, linker, librarian, PROM splitter, system builder); Part no. ADDS-21xx-BUN-PC (above tools plus an ANSI C compiler). Also available is the Personal Sound Architecture Software Development Kit, part number SDSC-SDK1—1.0 (available free of charge). This requires the above assembler, linker and system builder. It allows application development under *Windows*.

[6]Some sound cards that use the PSA chipset include:
- Echo Personal Sound System (Echo Speech Corp)
- Cardinal Pro 16 (plus) (Cardinal Technologies)
- Orchid SoundWave 32 (Orchid Technology)
- Wearnes Beethoven ADSP16 (Wearnes)
- Western Digital Paradise 16-DSP (Western Digital)
- Adaptec AME-1570 (Adaptec)

[7]For further information on the Analog Devices PSA chipset, please see the frequently-asked-questions (FAQ) file obtainable from Analog Devices Signal Computing BBS at 617-461-4258 or via Internet by anonymous FTP from ftp.analog.com in the /pub/dsp/tools/psa-sdk directory.

[8]WSS is the Microsoft *Windows* Sound System specification, obtainable from Microsoft.

[9]Sound Blaster is the Creative Labs sound format.

[10] Data sheets for the AD1848 are available from the Analog Devices Literature Center. See note 5.

[11]An example is the Orchid Soundwave32, manufactured by Orchid Technology, 45365 Northport Loop West, Fremont, CA 94538-6469, tel: 510-683-0300.

[12]*SpAsm*, Copyright 1992 by James Brundell, Physics Dept, University of Otago, PO Box 56, Dunedin, New Zealand, e-mail: james@physics.otago.ac.nz (Internet).

```
Page 1  CWMON.DSP

{-------------------------------------------------------------------}
{     Super Simple DSP Bootstrap monitor for testing the CW filter  }
{     Demo software accompanying the QEX August 1994 article:       }
{     "Programming a DSP PC Sound Card for Amateur Radio"           }
{                 (c) Johan Forrer, KC7WW                           }
{                  26553 Priceview Drive                            }
{                  Monroe, OR 97456                                 }
{-------------------------------------------------------------------}
{  Acknowlegements to:                                              }
{  Joe Ballantyne, Echo Speech Corporation                          }
{  DSP Applications Group, Analog Devices                           }
{-------------------------------------------------------------------}
{                                                                   }
{          Compilation instructions:                                }
{               >qsasm21 cwmon.dsp                                  }
{               >cload   cwmon                                      }
{                                                                   }
{-------------------------------------------------------------------}

                                { Monitor version number            }
                                { returned when monitor is initialized }
.CONST version = 0x0100;        { msb byte represents integer value }
                                { lsb byte represents a decimal value }
                                { from .01 to .99                   }

{-------------------------------------------------------------------}
{   List of implemented call and answer functions                   }
{-------------------------------------------------------------------}
{      CODE     Function                                            }
{      ----     --------                                            }
{      0x00b0 - Start timer                                        }
{      0x00b1 - End timer                                          }
{      0x00b4 - Start application                                  }
{      0x00b5 - End   application                                  }
{      0x00d0 - Read  Dm location                                  }
{      0x00d1 - Write Dm location                                  }
{      0x00d2 - Read  Pm location                                  }
{      0x00d3 - Write Pm location                                  }
{      0x00d4 - Send back "PSA-MON" string                         }
{-------------------------------------------------------------------}

{             The folling are a number of useful addresses          }

.const PSS_data_reg     =0x3000;
.const PSS_control_reg  =0x3008;
.const PSS_status_reg   =0x3008;
.const PSS_dma_reg      =0x3010;
.const ram_bank_reg     =0x3018;
.const ext_mem_latch    =0x3020;

.const addr_1848        =0x3440;
.const data_1848        =0x3448;
.const stat_1848        =0x3450;
.const pio_1848         =0x3458;
.const IRQ_status       =0x31C0;
```

```
Page 2:  CWMON.DSP

.const dmal_1848        =0x1060;
.const dmar_1848        =0x1068;
.const enable_1848      =0x1070;

.const system_control   =0x3FFF;
.const wait_state_ctl   =0x3FFE;
.const timer_period     =0x3FFD;
.const timer_counter    =0x3FFC;
.const timer_prescale   =0x3FFB;


{ Note: delay lines use circular addressing                         }
{       9- 16 requires a " 16*0x0010" boundary                      }
{ length 17- 32 requires a " 32*0x0020" boundary                    }
{       33- 64 requires a " 64*0x0040" boundary                     }
{       65-128 requires a "128*0x0080" boundary                     }
{      129-256 requires a "256*0x0100" boundary                     }

{ Here we define the start address of our DSP application that      }
{ incidentally are embedded in the AD1848 sound port ISR            }
.const sound_port       =0x0100;        { ISR is at 0100            }

{ Define space for the FIR bandpass filter delay line               }
.const BPF1             =0x1800;        { 1800 -:  Input bandpass   }

{-------------------INTERRUPT VECTORS-------------------------------}
        jump main_; nop; nop; nop;      { 0000 restart interrupt}
        jump sound_port; nop; nop; nop; { 0004 IRQ2 interrupt}
        rti; nop; nop; nop;             { 0008 SPORT0 tx interrupt}
        rti; nop; nop; nop;             { 000C SPORT0 rx interrupt}
        rti; nop; nop; nop;             { 0010 IRQ1 or SPORT1 tx}
        rti; nop; nop; nop;             { 0014 pc_irq_ser }
        jump timer_int;nop;nop;nop;     { 0018 Timer interrupt }
{ ***************************************************************** }
main_:
        ax0 = dm(PSS_data_reg);         { dummy read to clear out last boot byte}
        ax1 = version;                  { send version number to PC}
        call put_word_;

        ax0 = 0x0007;                   { no nesting of interrupts }
        ICNTL=ax0;                      { all interupts EDGE sensitive }

        ax0 = 0xFFFF;                   { for max length Timer period }
        dm(timer_period) = ax0;
        ax0 = 0x00FF;
        dm(timer_prescale) = ax0; { Max Timer prescale}

        ax0 = 0x0011;                   { enable timer and IRQ2 interrupts - for DMA}
        IMASK = ax0;

        l0=0;l1=0;l2=0;l3=0;            { all length (modulo) registers }
        l4=0;l5=0;l6=0;l7=0;            { should be set to 0   }

        m0=0;m1=1;m2=2;m3=-1;           { modify registers set ( DAG1 set) }
        m4=0;m5=1;m6=0;m7=-1;           { to various constants ( DAG2 set) }
```

```
Page 3:  CWMON.DSP


        ax0=0x2000;
        dm(wait_state_ctl)=ax0; { set 2 wait states- Sound port }
                                { zero wait states for everything else }
        ax0=dm(system_control); { read the system control register}
        ay0=0xfff8;
        ar=ax0 and ay0;         { make program mem 0 wait state}
        dm(system_control)=ar;

        ax0 = 0x2000;           { set SPWE (bit 13) in IRQ enable register }
        dm(IRQ_status)=ax0;     { to allow DMA interrupts }

{-------------------------------------------------------------------}
{ Start looking for a call and answer command                       }
loop_:
        call get_word_;

        ay0=0x00d0;             { Dm read  }
        ar=ax0 xor ay0;
        if eq jump data_read_;

        ay0=0x00d1;             { Dm write }
        ar=ax0 xor ay0;
        if eq jump data_write_;

        ay0=0x00d2;             { Pm read  }
        ar=ax0 xor ay0;
        if eq jump program_read_;

        ay0=0x00d3;             { Pm write  }
        ar=ax0 xor ay0;
        if eq jump program_write_;

        ay0=0x00d4;             { Send back PSA-MON string }
        ar=ax0 xor ay0;
        if eq jump prog_id_write_;

        ay0=0x00b4;             { Start application }
        ar=ax0 xor ay0;
        if eq jump start_app_;

        ay0=0x00b5;             { End application }
        ar=ax0 xor ay0;
        if eq jump end_app;

        ay0=0x00b0;             { Start timer }
        ar=ax0 xor ay0;
        if eq jump start_timer;

        ay0=0x00b1;             { Stop timer }
        ar=ax0 xor ay0;
        if eq jump stop_timer;

        jump loop_;
```

```
Page 4:  CWMON.DSP

{-------------------------------------------------------------------}
{ Timer ISR                                                         }
{ Assert PC interrupt                                               }
timer_int:
        ena sec_rep;
        ax0=0x1000;
        dm(PSS_control_reg)=ax0;
        dis sec_rep;
        rti;

{-------------------------------------------------------------------}
{ Call and Answer functions                                         }
data_read_:
        call get_word_;
        i4 = ax0;
        ax1 = dm(i4,m6);                { m6=0 }
        call put_word_;
        jump loop_;

data_write_:
        call get_word_;
        i4 = ax0;
        call get_word_;
        dm(i4,m6)=ax0;
        jump loop_;

program_read_:
        call get_word_;
        i4 = ax0;
        ax1 = pm(i4,m6);
        call put_word_;
        ax1 = px;
        call put_word_;
        jump loop_;

program_write_:
        call get_word_;
        i4 = ax0;
        call get_word_;
        px = ax0;
        call get_word_;
        pm(i4,m6)=ax0;
        jump loop_;

prog_id_write_:
        ax1=0x0050;             { P }
        call put_word_;
        ax1=0x0053;             { S }
        call put_word_;
        ax1=0x0041;             { A }
        call put_word_;
        ax1=0x002D;             { - }
        call put_word_;
        ax1=0x004D;             { M }
        call put_word_;
```

Page 5: CWMON.DSP

```
        ax1=0x004F;             { O }
        call put_word_;
        ax1=0x004E;             { N }
        call put_word_;
        ax1=0x0000;
        call put_word_;
        jump loop_;

get_word_:
        ay0=0x4000;
        ax0=dm(PSS_status_reg);
gwloop:
        ar=ax0 and ay0;
        ax0=dm(PSS_status_reg);
        if eq jump gwloop;
        ax0=dm(PSS_data_reg);
        rts;

put_word_:
        ax0=dm(PSS_status_reg);
        ay0=0x8000;
        ar=ax0 and ay0;
        if eq jump put_word_;
        dm(PSS_data_reg) = ax1;
        rts;

{ Stop/Start TIMER ------------------------------------------------------ }
start_timer:
        ENA TIMER;
        jump    loop_;
stop_timer:
        DIS TIMER;
        jump    loop_;

{ Start/Stop application  ----------------------------------------------- }
start_app:
        ax0=0xFFFF;
        dm(enable_1848)=ax0;    { Let DSP access 1848 }

        i1=BPF1;                { Pointers and lengths }
        l1=127;

        ax0 = 0x0021;           { enable IRQ2 interrupts - for DMA}
        IMASK = ax0;            { enable TIMER interrupts too }

        jump loop_;

end_app:
        ax0 = 0x0000;           { disable IRQ2 interrupts - for DMA}
        IMASK = ax0;
        ax0=0x0000;             { get ready to return to the PC }
        dm(enable_1848)=ax0;    { Now let PC access 1848 }

        jump loop_;
{------------------------------------------------------------------------}
```

Page 1: CW250.DSP

```
{------------------------------------------------------------------------}
{                PSA super duper 250 Hz CW Filter                        }
{       Demo software accompanying the QEX August 1994 article:          }
{       "Programming a DSP PC Sound Card for Amateur Radio"              }
{                    (c) Johan Forrer, KC7WW                             }
{                     26553 Priceview Drive                              }
{                     Monroe, OR 97456                                   }
{------------------------------------------------------------------------}
{                                                                        }
{                Assembly instructions:                                  }
{                ----------------------                                  }
{                  -epasm21 cw250.dsp                                    }
{                                                                        }
{------------------------------------------------------------------------}

{   DSP hardware addresses used by user's application                    }
.const dmal_1848        =0x3040;
.const dmar_1848        =0x3068;

{ Note: delay lines use circular addressing                              }
{       9- 16 requires a " 16=0x0010" boundary                           }
{ length 17- 32 requires a " 32=0x0020" boundary                         }
{       33- 64 requires a " 64=0x0040" boundary                          }
{       65-128 requires a "128=0x0080" boundary                          }
{      129-256 requires a "256=0x0100" boundary                          }

{ Define space for the FIR bandpass filter delay line                    }
.const BPF1             =0x1800;        { 1800 -:  Input bandpass        }

.const bpf_out          =0x1900;        { Ouput of FIR filter            }
{------------------------------------------------------------------------}
{ This is the user's DSP application code.  It's part of the AD1848 ISR  }
{ It should be kept as efficient as possible.                            }
{------------------------------------------------------------------------}
0x0100  sound_prt:
        ena sec_reg;                    { use alternate registers }

{ The way we have the audio jack connected, only the left line           }
{ channel will carry a signal.                                           }
{ Input data is saved in a circular buffer.                              }

        ar=dm(bpf_out);                 { pick up oldest value }
        dm(dmal_1848)=ar;
        dm(dmar_1848)=ar;               { this also resets interrupt }

        ar=dm(dmal_1848);               { using only left input channel }
        dm(i1,m1)=ar;
        ar=dm(dmar_1848);               { read right input but discard it }

        call bandpass;                  { call bandpass routine }

        dis sec_reg;                    { use normal registers }
        rts;
```

Page 2: CW250.DSP

```
{--FIR FILTER----------------------------------------------------------}
{                                                                      }
{ This FIR filter was designed using the window method.  The window used }
{ was a KAISER-BESSEL type, filter length is 127 taps.                 }

{ Sampling frequency is          5512.500 Hz                           }
{ Lower cut-off frequency is      375.000 Hz                           }
{ Upper cut-off frequency is      525.000 Hz                           }
{ ------------------------Filter coefficients--------------------------- }
dm:     0xFFFD00; {    -3}
        0xFFF900; {    -7}
        0xFFF700; {    -9}
        0xFFF600; {   -10}
        0xFFFA00; {    -6}
        0x000200; {     2}
        0x000D00; {    13}
        0x001800; {    24}
        0x001F00; {    31}
        0x001E00; {    30}
        0x001300; {    19}
        0xFFFF00; {    -1}
        0xFFE600; {   -26}
        0xFFD100; {   -47}
        0xFFC600; {   -58}
        0xFFCA00; {   -54}
        0xFFDF00; {   -33}
        0xFFFE00; {    -2}
        0x001F00; {    31}
        0x003700; {    55}
        0x004000; {    64}
        0x003600; {    54}
        0x001F00; {    31}
        0x000400; {     4}
        0xFFF200; {   -14}
        0xFFEF00; {   -17}
        0xFFFB00; {    -5}
        0x000E00; {    14}
        0x001800; {    24}
        0x000A00; {    10}
        0xFFE000; {   -32}
        0xFF9F00; {   -97}
        0xFF5F00; {  -161}
        0xFF1D00; {  -195}
        0xFF5800; {  -168}
        0xFFBE00; {   -66}
        0x006400; {   100}
        0x012300; {   291}
        0x01C000; {   448}
        0x01FB00; {   507}
        0x01A700; {   423}
        0x00BA00; {   186}
        0xFF5F00; {  -161}
        0xFDEB00; {  -533}
        0xFCCC00; {  -820}
```

Page 3: CW250.DSP

```
        0xFC6900; {  -919}
        0xFD0000; {  -768}
        0xFE8B00; {  -373}
        0x00B800; {   184}
        0x02F900; {   761}
        0x04AA00; {  1194}
        0x053E00; {  1342}
        0x046F00; {  1135}
        0x025500; {   597}
        0xFF6D00; {  -147}
        0xFC7800; {  -904}
        0xFA4700; { -1465}
        0xF98100; { -1663}
        0xFAAE00; { -1426}
        0xFCDF00; {  -801}
        0x003800; {    56}
        0x039600; {   918}
        0x060F00; {  1551}
        0x06F700; {  1783}
        0x060F00; {  1551}
        0x039600; {   918}
        0x003800; {    56}
        0xFCDF00; {  -801}
        0xFAAE00; { -1426}
        0xF98100; { -1663}
        0xFA4700; { -1465}
        0xFC7800; {  -904}
        0xFF6D00; {  -147}
        0x025500; {   597}
        0x046F00; {  1135}
        0x053E00; {  1342}
        0x04AA00; {  1194}
        0x02F900; {   761}
        0x00B800; {   184}
        0xFE8B00; {  -373}
        0xFD0000; {  -768}
        0xFC6900; {  -919}
        0xFCCC00; {  -820}
        0xFDEB00; {  -533}
        0xFF5F00; {  -161}
        0x00BA00; {   186}
        0x01A700; {   423}
        0x01FB00; {   507}
        0x01C000; {   448}
        0x012300; {   291}
        0x006400; {   100}
        0xFFBE00; {   -66}
        0xFF5800; {  -168}
        0xFF1D00; {  -195}
        0xFF5F00; {  -161}
        0xFF9F00; {   -97}
        0xFFE000; {   -32}
        0xFFFB00; {    -5}
        0x000A00; {    10}
        0x001800; {    24}
```

Page 4: CW250.DSP

```
        0x000800, {    14}
        0xFFF800, {    -5}
        0xFFEF00, {   -17}
        0xFFF200, {   -14}
        0x000400, {     4}
        0x001F00, {    31}
        0x003600, {    54}
        0x004000, {    64}
        0x003700, {    55}
        0x001F00, {    31}
        0xFFFE00, {    -2}
        0xFFDF00, {   -33}
        0xFFCA00, {   -54}
        0xFFC600, {   -58}
        0xFFD100, {   -47}
        0xFFE600, {   -26}
        0xFFFF00, {    -1}
        0x001300, {    19}
        0x001E00, {    30}
        0x001F00, {    31}
        0x001800, {    24}
        0x000D00, {    13}
        0x000200, {     2}
        0xFFFA00, {    -6}
        0xFFF600, {   -10}
        0xFFF700, {    -9}
        0xFFF900, {    -7}
        0xFFFD00, {    -3}
} ------------------------------------------------------------------------}
bandpass:
        i4=dr;                          { point to coeffiricnts }
        mr=0, ma0=de(i1,m1), my0=ge(i4,m5);
        cntr=126;                       { length - 1 }
        do sopbp until ce;
sopbp:  mr=mr+ma0*my0(ss), ma0=de(i1,m1), my0=ge(i4,m5);
        mr=mr+ma0*my0(rnd);
        if mv sat mr;
        dm(bpf_out)=mr1;
        rts;
{ -------------END OF USER APPLICATION-----------------------------------}
```

Page 1: CW250.CPP

```
// ------------------------------------------------------------------
//      Super Duper DSP CW filter - Control and User Interface
//      Demo software accompanying the QEX August 1994 article:
//      "Programming a DSP PC Sound Card for Amateur Radio"
//                  (c) Johan Forrer, KC7WW
//                  24553 Priceview Drive
//                  Monroe, OR 97456
// ------------------------------------------------------------------
//
//      Compilation/link instructions for Borland C++ 4.0
//
//      >bcc -c -ml -O2 -Z -G cw250.cpp
//      >tlink /x c:\bc4\lib\c0l cw250 PSA1.obj PSA2.obj, emu mathl cl
//
// ------------------------------------------------------------------
//              This is how the AD1848 is reprogrammed:
// ------------------------------------------------------------------
// Idx=0
// Sta=0
//  0 =+7    select left  line, gain =7
//  1 =+7    select right line, gain =7
//  2 =0x80  mute left  aux#1
//  3 =0x80  mute right aux#1
//  4 =0x80  mute left  aux#2
//  5 =0x80  mute right aux#2
//  6 =0x07  left  DAC with some attenuation
//  7 =0x07  right DAC with some attenuation
//  8 =0x51  16 bit 2s compl, linear PCM, stereo,
//           stall-16.93MHz, 5512.5 kHz,
//  9 =0x08  DMA capture/playback, Autocalibrate after mode change,
//           disable capture/playback, dual DMA mode
// 10 =0     N.A
// 11 =0     N.A
// 12 =0     N.A
// 13 =0     digital mix muted
// 14 =0     base count = 0
// 15 =0     base count = 0
// ------------------------------------------------------------------

#include "psa.h"

// ------------------------------------------------------------------
// Local function prototypes
// ------------------------------------------------------------------
void start_application(char *, unsigned char *);
void terminate_application(void);
void timer_test(void);

int tick;                       // a global tick counter

#define MON "cwmon.ld"          // define name of bootstrap
#define APP "cw250.cde"         // define name of application

void main()
```

Page 2: CW250.CPP

```
{
unsigned int boot_version;
unsigned char initstate[16]={0,0,0x47,0x47,0x80,0x80,0x80,
                0x80,0x57,0x57,0x51,0x08,0,0,0,0,0,0};

// ------------------------------------------------------------------
// Announce who we are and what the program is about
// ------------------------------------------------------------------
        printf("Demo program to set up and communicate with a PSA sound card\n");
        printf("By Johan Forrer, KC7WW\n\n");

// ------------------------------------------------------------------
// Check for presence and initialize DSP card
// ------------------------------------------------------------------
        if (!checkDSPcard(psabase, wssbase)) {
                printf("No PSA card - or WSS address is not available\n");
                exit(0);
                }

// ------------------------------------------------------------------
// Perform the DSP bootstrap next
// and verify that it has completed successfully
// ------------------------------------------------------------------
        boot_version=bootdsp(MON, NULL, (unsigned)0x500);
        checkdspteestprog();
        printf("Monitor version %d.%d reported\n",
                boot_version>>8, boot_version&0xFF);

// ------------------------------------------------------------------
// Finally we are ready to load and run the DSP application
// also start the timer
// ------------------------------------------------------------------
        start_application(APP, initstate);
        timer_test();
        terminate_application();
}

//------------------------------------------------------------------
//                  Some local functions
//------------------------------------------------------------------
//------------------------------------------------------------------
// Start application
//------------------------------------------------------------------
void start_application(char *app_name, unsigned char ad1848_state[])
{

        load_DSP_mem(app_name);         // load the DSP application code

        set_1848state(ad1848_state);    // reprogram CODEC

        outp(wssbase+WSS_CODECINDEXADDR,0x0009);  // Capture/Playback on
        outp(wssbase+WSS_CODECINDEXDATA,0x0003);

        putdspword(0x0004);             // send "start application" code to DSP
```

Page 3: CW250.CPP

```
}

//------------------------------------------------------------------
// Terminate application
//------------------------------------------------------------------
void terminate_application()
{

        putdspword(0x00b5);     // send "stop application" code to DSP

        outp(wssbase+WSS_CODECINDEXDATA,0x0009); // Playback/capture off
        outp(wssbase+WSS_CODECINDEXDATA, 0);

        outp(wssbase+WSS_CODECINDEXADDR,0x0006);  // mute DAC
        outp(wssbase+WSS_CODECINDEXDATA, 0xDF);
        outp(wssbase+WSS_CODECINDEXADDR,0x0007);
        outp(wssbase+WSS_CODECINDEXDATA, 0xDF);

}

//------------------------------------------------------------------
// Timer test using interrupt IRQ7
//------------------------------------------------------------------
void timer_test()
{

// Define ISR prototypes
void interrupt (*oldfunc)(...);         // for storage of old vector
void interrupt DSP_int_serv(...);       // Prototype for new interrupt serv.
unsigned char old_IMR;                  // Save old PIC IMR

        tick=0;
        printf("Timer is now ticking - hit any key to stop application\n ");

        oldfunc=getvect(INT);           // Hook IRQ7 on the PC side
        setvect(INT,DSP_int_serv);

// Set INT7 on DSP side as well
        outpw(psabase+PSA_CONFIG,IRQ7);
        old_IMR=inp(0x21);              // get IMR
        outp(0x21,old_IMR&0x7F);        // Arm PC's IRQ7 in IMR

        putdspword(0x00b0);             // send "start timer" code to DSP

// Show timer ticks, check when user quits
        for(;;) {
                if(kbhit()) break;
                if(tick) {
                        printf("."),
                        tick=0;
                        }
                }
```

```
Page 4:  CW250.CPP

        getch();                        // flush out breakout character

        putdspword(0x00b1);             // send "stop timer" code to DSP

        outpw(psbase+PSS_IRQ_ACK, 0);   // Clear any pending PSA interrupts
        outpw(psbase+PSS_CONFIG,0x0000);  // Remove PSA from IRQ7

        outp(0x21,old_IMR);             // Restore IMR

        setvect(INT, oldfunc);          // Restore INT

        printf("\nApplication terminated\n");
}

//-------------------------------------------------------------------
// The new IRQ7 interrupt handler
//-------------------------------------------------------------------
void interrupt DSP_int_serv(...)
{
        outpw(psbase+PSS_IRQ_ACK, 0);   // Clear the interrupt
        outp(0x20,0x20);                // write EOI to PIC
                                        // wlee bad things happen!
        tick++;
}
```

```
Page 1:  PSA.H

//-------------------------------------------------------------------
//              Header file for use with user interface
//      Demo software accompanying the QEX xxx  1994 article:
//      "Using a Programmable PC Sound Card for Amateur Radio"
//                  (c) Johan Forrer, KC7WW
//                       26553 Priceview Drive
//                    Monroe, OR 97456
//-------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <dos.h>

#define IRQ7            0x0018
#define INT       0x0F          // INT7+0x0F

/* WSS register address offsets from WSS base address. */
#define WSS_STATUS                      0x0000
#define WSS_CODECINDEXADDR              0x0004
#define WSS_CODECINDEXDATA              0x0005
#define WSS_CODECSTATUS                 0x0006
#define WSS_CODECDATA                   0x0007

/* PSS registers' io port address offsets from psbase. */
#define PSS_CONFIG    0x0010

#define PSS_DATA      0x0000
#define PSS_STATUS    0x0002
#define PSS_CONTROL   0x0002
#define PSS_ID_VERS   0x0004
#define PSS_IRQ_ACK   0x0004

// External function prototypes

extern unsigned int read_data(unsigned int);
extern void write_data(unsigned int, unsigned int);
extern unsigned long read_prog(unsigned int);
extern void write_prog(unsigned int, unsigned long);
extern int load_DSP_mem(char *);
extern void checkdsptestprog(void);
extern int checkDSPcard(int, int);
extern int bootdsp(char *, char *, unsigned);
extern void set16Bstate(unsigned char *);
extern int putdspword(unsigned);


//-------------------------------------------------------------------
// NOTE: for this application the card base is assumed at 220h
//       and the  Windows sound system base is assumed at 530h
//-------------------------------------------------------------------
extern unsigned int psbase, wssbase;   // ports
```

The KD2BD Pacsat Modem

This high-performance Pacsat modem you can build will get you up and running on the 1200-bit/s Pacsats.

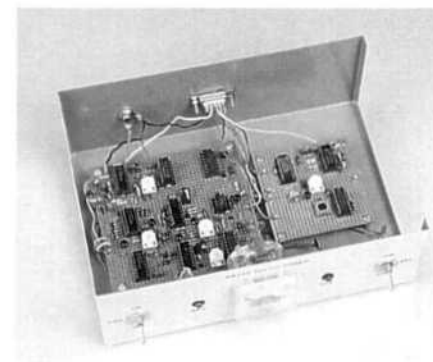
By John A. Magliacane, KD2BD

The KD2BD Pacsat Modem is a low-cost, high-performance 1200-bit/s BPSK modem designed to interface between a packet-radio TNC and an amateur-satellite ground station. It allows full-duplex access to the 1200-baud Pacsat constellation of amateur satellites.

Important features of this modem include:

- high immunity to noise and interference
- "matched filter" demodulator performance
- non-limiting (linear) balanced phase detection
- wide-range automatic gain control
- Doppler-correcting AFC for the downlink receiver
- compatible with current and future Pacsats
- uses readily available components
- interfaces easily with standard TNCs
- operates from a single +12-V dc power supply
- suitable for terrestrial communication
- Provides outstanding performance

1320 Willow Drive
Sea Girt, NJ 08750



The modem consists of a 1200-bit/s BPSK demodulator for receiving Pacsat transmissions and a bi-phase Manchester encoder for generating Pacsat uplink signals. The demodulator uses coherent phase detection and correlation decoding techniques that are capable of demodulating BPSK signals well into the noise level. It also includes an automatic gain control (AGC) system to compensate for signal strength variations and an automatic frequency control (AFC) for tuning the downlink receiver in response to Doppler shift during a satellite pass. The demodulator is very sensitive and is capable of locking into BPSK signals so weak they are barely audible above the ambient receiver noise. The modulator produces low-distortion 1200-bit/s bi-phase Manchester code suitable for generating Manchester-encoded FSK when used with a standard 2-meter FM voice transmitter, or BPSK when used with an SSB transmitter.

The modem was originally designed to interface with an MFJ 1270B terminal node controller (a TNC-2 clone) and a Yaesu FT-726R multi-mode VHF/UHF communications transceiver. Suitable interfacing modifications should allow proper modem operation with other TNCs and ground-station radio equipment. The specifics for doing so are left to the expertise of the reader.

The Pacsats

Orbital Satellites Carrying Amateur Radio (OSCARs) have been a part of Amateur Radio for over 30 years. With the growing popularity of terrestrial packet-radio communication on the amateur bands, recent OSCAR satellites have been designed to make use of the AX.25 protocol in their communication with ground stations. Some satellites, such as DOVE-OSCAR-17, use the protocol for the transmission of telemetry information through beacon transmitters, while others use the protocol to provide full-duplex communication links with the satellites' transponders. Amateur satellites containing packet radio store-and-forward transponders are known as "Pacsats."

The current roster of satellites making 1200-bit/s NRZI, HDLC, AX.25 protocol compatible packet-radio transmissions on amateur frequencies includes PACSAT-OSCAR-16, DOVE-OSCAR-17, WEBERSAT-OSCAR-18, LUSAT-OSCAR-19, FUJI-OSCAR-20, AMSAT-OSCAR-21 and ITAMSAT-OSCAR-26. Table 1 lists the operating frequencies of each of these OSCAR satellites.

Of these satellites, only DOVE-OSCAR-17 uses audio frequency shift keying (AFSK) modulation on a narrow-band FM carrier, which is compatible with existing terrestrial VHF-FM and UHF-FM packet-radio communication. The remaining satellites use binary phase shift keying (BPSK) modulation. As popular and as widespread as the AFSK-FM method of transmission is for 1200-baud packet-radio communication, AFSK-FM and its popular demodulation methods yield a level of performance that leaves a lot to be desired. Steve Goode, K9NG, has shown through extensive testing that a Tucson Amateur Packet Radio (TAPR) TNC-1 internal Bell 202 modem required a signal level that produced at least 25 dB of FM receiver quieting (25 dBQ) for high communication reliability. Since this is a difficult signal level to achieve from micro satellites operating on UHF frequencies with only several watts of transmitter power, a more robust binary phase shift keying (BPSK) emission was selected for use by 1200-bit/s Pacsat downlink transmitters.

Binary Phase Shift Keying

Coupled with low binary data rates, binary phase shift keying has allowed interplanetary space probes to transmit vast quantities of data to ground stations on Earth from great distances with low transmitter power. BPSK offers a 6-dB signal-to-noise ratio advantage over coherent-CW (CCW) modulation. (CCW has long been considered to be the premium weak-

Table 1—1200-bit/s BPSK Pacsats

PACSAT-OSCAR-16 : NASA Catalog Number 20439

Uplinks	*Downlinks*
Channel #1: 145.900 MHz	437.051 MHz—primary; raised cosine, right hand circular polarization
Channel #2: 145.920 MHz	437.026 MHz—secondary, left hand circular polarization
Channel #3: 145.940 MHz	2401.143 MHz—secondary
Channel #4: 145.960 MHz	

Operates as a file-server in space. Use *PB* terminal software for downloads, *PG* for uploads.

WEBERSAT-OSCAR-18 : NASA Catalog Number 20441

Downlinks

437.102 MHz—primary; raised cosine, right hand circular polarization
437.075 MHz—secondary, left hand circular polarization (this transmitter is bad and is not normally used)

Contains a color CCD Earth-imaging camera and transmits images using the Pacsat Broadcast Protocol. OSCAR-18 also contains a file-server, although it will not be activated until the primary mission objective of taking Earth images is fully exhausted.

LUSAT-OSCAR-19 : NASA Catalog Number 20442

Uplinks	*Downlinks*
Channel #1: 145.840 MHz	437.153 MHz—primary, left hand circular polarization
Channel #2: 145.860 MHz	437.125 MHz—secondary; raised cosine, right hand circular polarization
Channel #3: 145.880 MHz	
Channel #4: 145.900 MHz	

Operates as a file-server in space. Use *PB* terminal software for downloads, *PG* for uploads.

FUJI-OSCAR-20 : NASA Catalog Number 20480

Uplinks	*Downlink*
Channel #1: 145.850 MHz	435.910 MHz
Channel #2: 145.870 MHz	
Channel #3: 145.890 MHz	
Channel #4: 145.910 MHz	

Operates as a packet radio bulletin board in space. No special ground station terminal software is required for access.

AMSAT-OSCAR-21 : NASA Catalog Number 21087

Uplink	*Downlink*
435.016 MHz	145.987 MHz RUDAK-II Mode 1

Offers many features including a packet radio mailbox.

ITAMSAT-OSCAR-26 : NASA Catalog Number 22826

Uplinks	*Downlinks*
Channel #1: 145.875 MHz	435.867 MHz—primary
Channel #2: 145.900 MHz	435.822 MHz—secondary
Channel #3: 145.925 MHz	
Channel #4: 145.950 MHz	

Operates as a file-server in space. Use *PB* terminal software for downloads, *PG* for uploads.

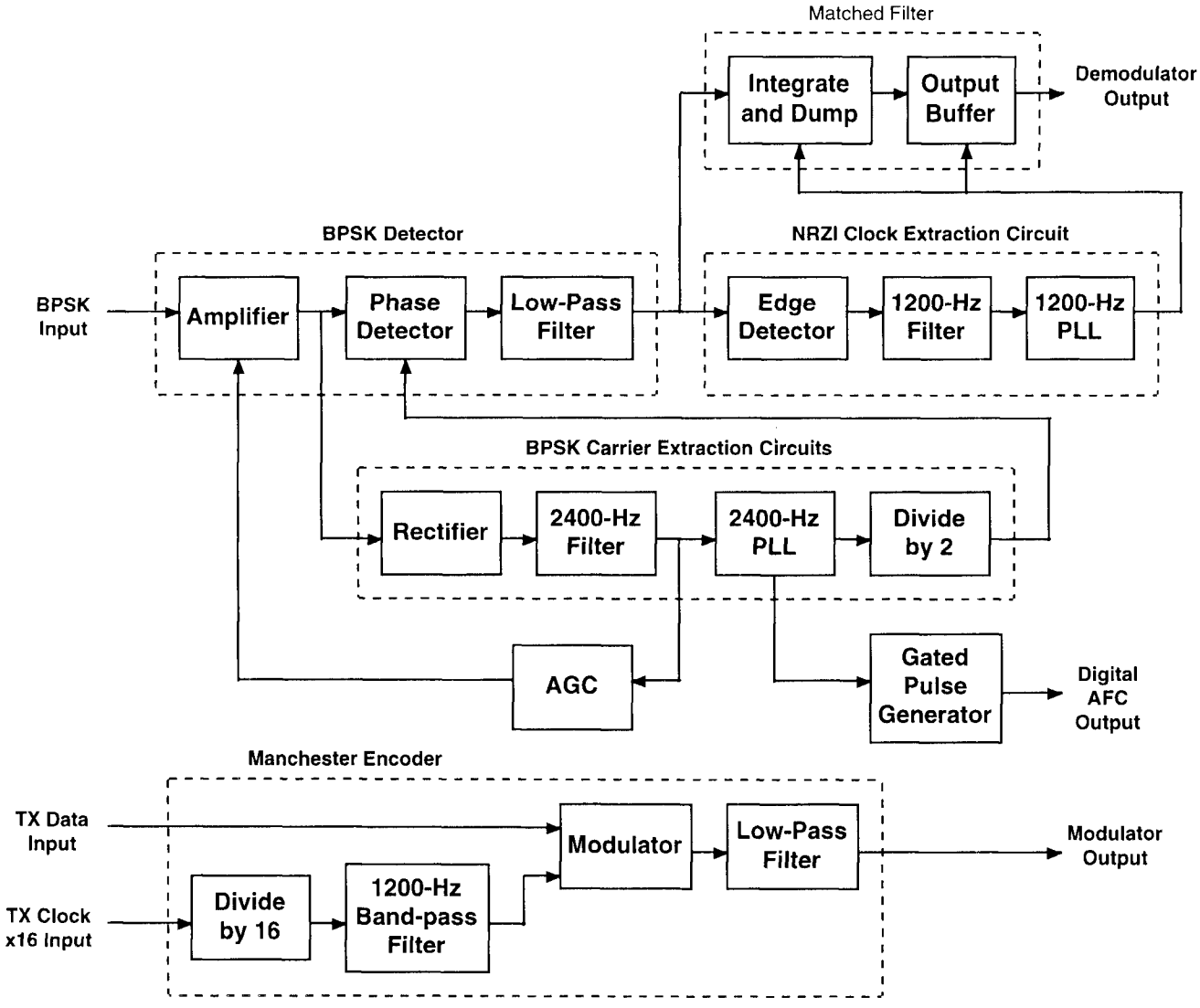


Fig 1—Block diagram of the KD2BD Pacsat Modem.

signal communication mode available to amateurs.)

BPSK is produced by modulating a carrier oscillator with binary modulating information in a balanced modulator. The resulting BPSK emission can be analyzed several ways, depending on whether the analysis is performed in the time domain or the frequency domain. In the time domain, the BPSK signal looks similar to the local carrier introduced to the modulator, except its phase shifts by 180-degree intervals with binary modulation. In the frequency domain, BPSK looks similar to a double-sideband suppressed-carrier AM signal centered about its local suppressed carrier frequency.

BPSK Signal Processing

Fig 1 is a block diagram of the KD2BD Pacsat Modem. The demodulator is designed to operate with an SSB receiver properly tuned to the frequency of a BPSK transmitter. In essence, the SSB receiver merely acts as a frequency converter, translating the RF BPSK signal captured by the ground-station antenna down to the audio-frequency range where it can be easily processed by the demodulator using audio circuitry.

The BPSK signal applied to the input of the modem branches in two separate directions within the demodulator. One path extracts, processes and regenerates the BPSK carrier, while the second performs BPSK signal detection and filtering.

BPSK Carrier Recovery

BPSK signals in this modem design are demodulated synchronously, and synchronous detectors require a reference carrier for phase determination. Since the BPSK transmitter suppresses its carrier in its balanced modulator, there is no clearly defined reference present in a BPSK signal. The demodulator must therefore synthesize a BPSK reference carrier from sideband components present in the composite BPSK signal. If viewed in the time domain, the carrier of a signal whose phase shifts by 180-degree

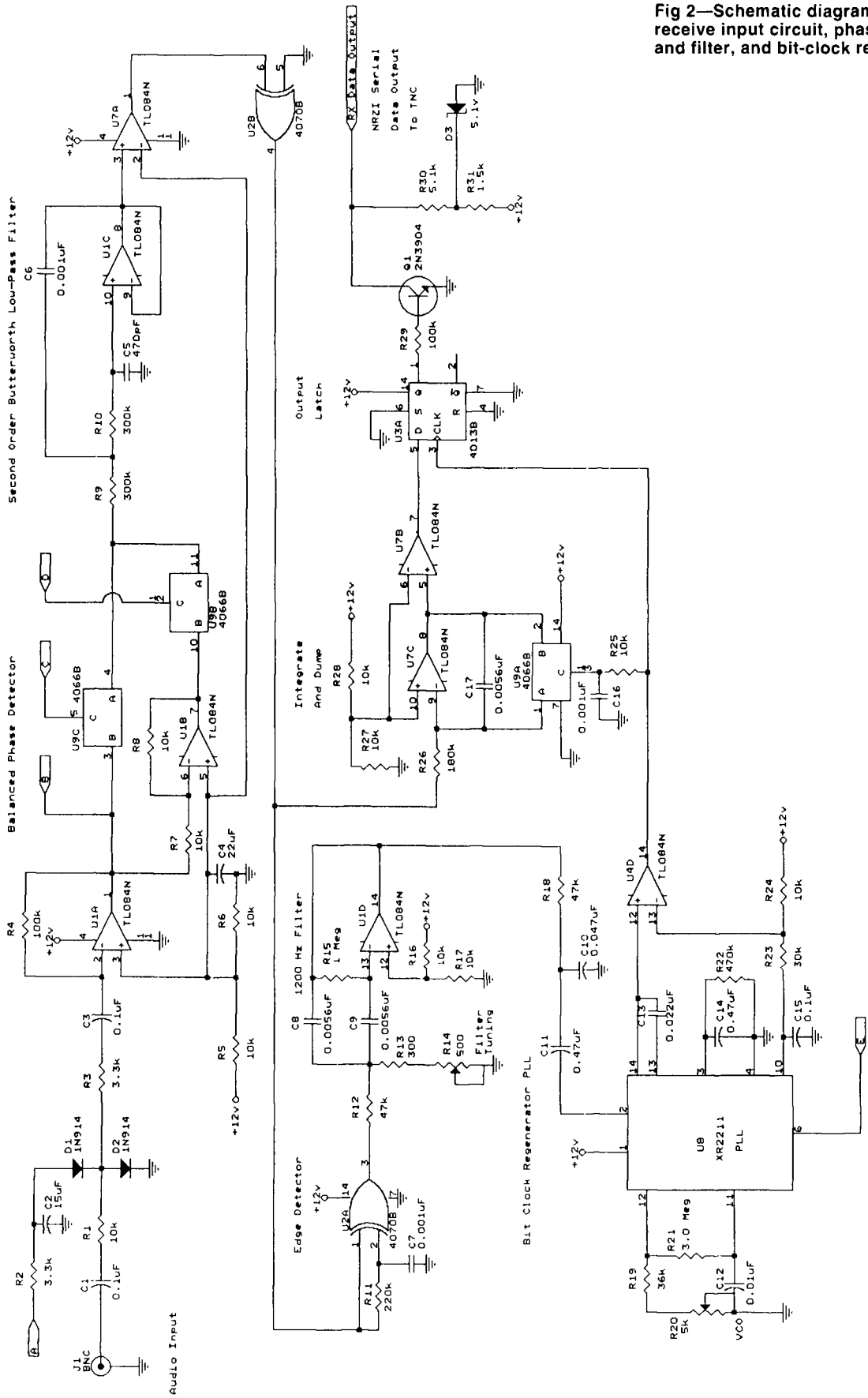


Fig 2—Schematic diagram of the receive input circuit, phase detector and filter, and bit-clock regenerator.

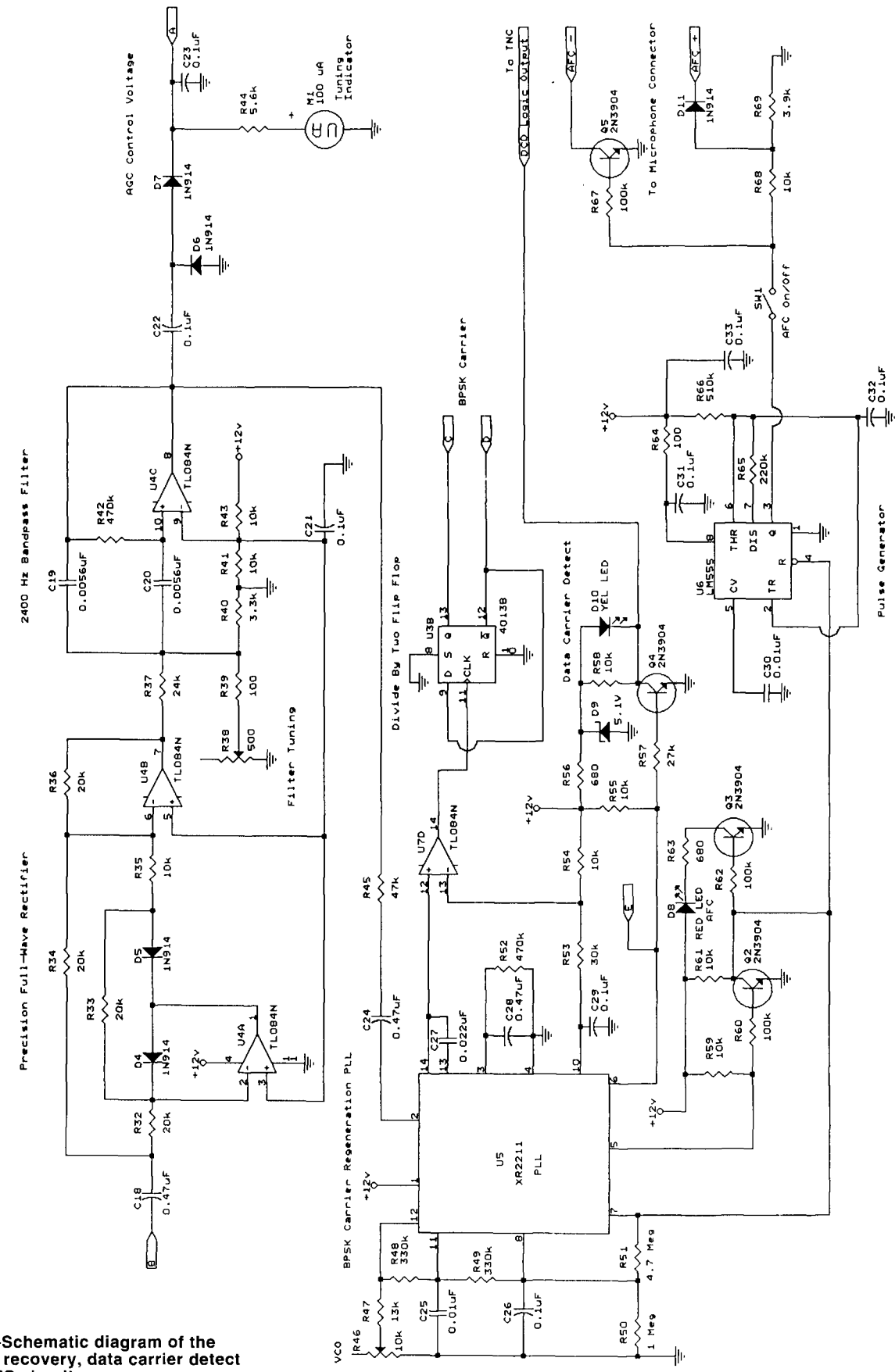


Fig 3—Schematic diagram of the carrier recovery, data carrier detect and AFC circuitry.

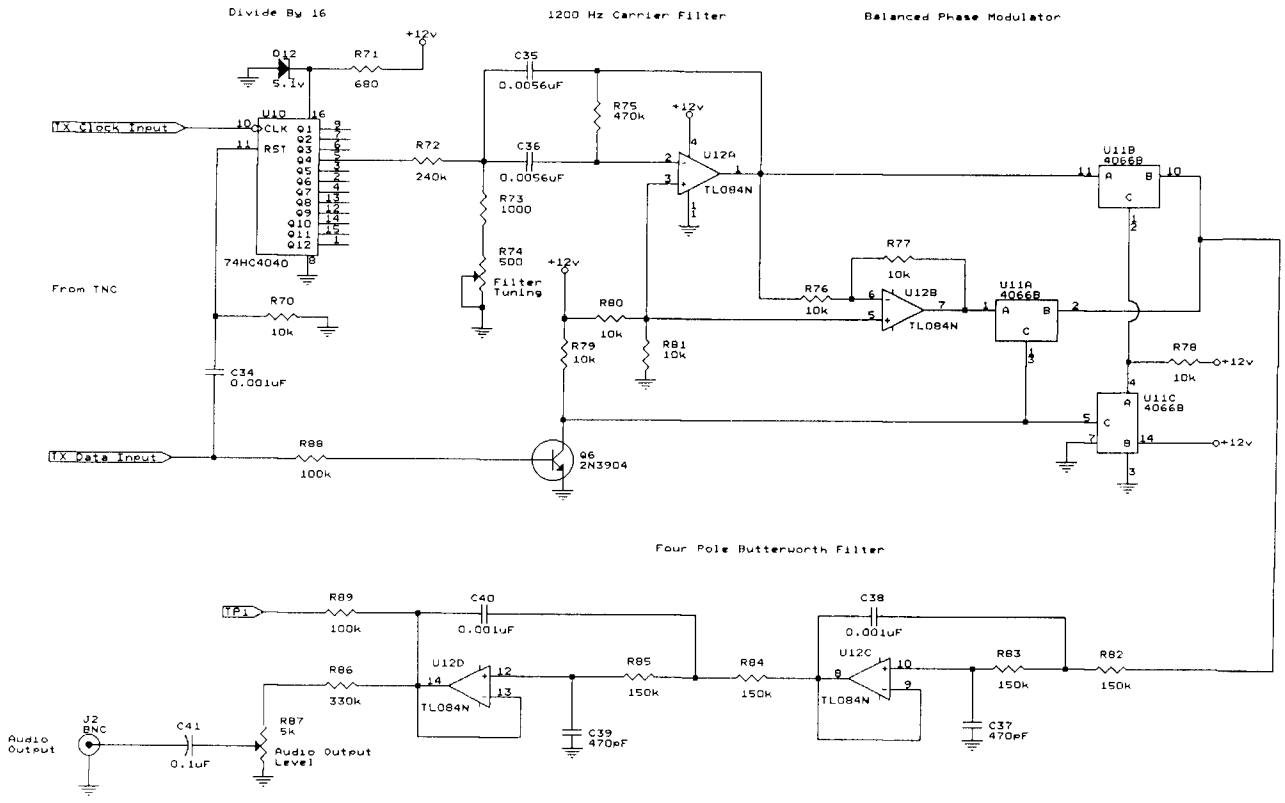


Fig 4—Schematic diagram of the transmit encoder and filter.

intervals may be extracted by taking the absolute value of the BPSK waveform voltage and filtering the result. A full-wave rectifier can be used to perform the absolute value function. The rectification process yields a waveform of constant phase at a frequency twice that of the BPSK suppressed carrier. In the frequency domain, the rectifier may be thought of as being a nonlinear circuit that mixes the upper and lower BPSK sidebands, producing the algebraic sum of those sidebands. Dividing the frequency of this product by two produces a local carrier of constant phase whose frequency equals that of the suppressed BPSK carrier.

Referring to the schematic diagram of the KD2BD Pacsat Modem shown in Figs 2, 3 and 4, the received BPSK signal is first passed through a current-variable input attenuator. Resistor R1 acts as the series element of an "L" attenuator. Diodes D1 and D2 act as current-variable resistances to form the shunt element. The AGC voltage that drives the input attenuator is derived from the filtered BPSK carrier.

The level-controlled BPSK signal is passed through input amplifier U1A, after which it is split between the balanced phase detector and the carrier-extraction circuits. The amplified BPSK signal is processed through a precision full-wave rectifier circuit built around U4A and U4B, two sections of a TL084N quad-biFET operational amplifier, as shown in Fig 3. The output voltage of the full-wave rectifier is the absolute value of the incoming BPSK signal. If the downlink receiver is tuned such that the BPSK suppressed carrier is at a frequency of 1200 Hz, the output of the rectifier will contain a strong product at twice this frequency, 2400 Hz. Not only does the rectifier mix the BPSK sidebands together, it mixes noise components together, creating even more noise energy at the rectifier output. In an effort to remove the undesired noise, the rectifier output is filtered through a narrow-bandwidth bandpass filter built around op amp U4C. This filter has a center frequency of 2400 Hz, a bandwidth of 120 Hz and a Q of 20. The result of this processing is shown in Fig 5. The filtered carrier is further processed through U5, an XR2211 phase-locked-loop (PLL) FSK demodulator/tone-decoder circuit with a loop bandwidth of 120 Hz. The XR2211 offers outstanding frequency stability and provides several output signals and other features used for receiver automatic-frequency control (AFC) and data carrier detection (DCD). The phase-locked loop regenerates the 2400-Hz carrier, producing a noise-free waveform of constant amplitude locked in frequency and in phase with that of the BPSK carrier.

The phase-locked loop's oscillator signal is extracted from pin 14, shaped to a pulse waveform through op amp U7D, and applied to U3B, a "D" flip-flop configured as a frequency divider. Toggling on the rising edge of each input pulse, the flip-flop divides the frequency of the PLL oscillator by two, reproducing the 1200-Hz BPSK carrier with a 50% duty cycle, as required

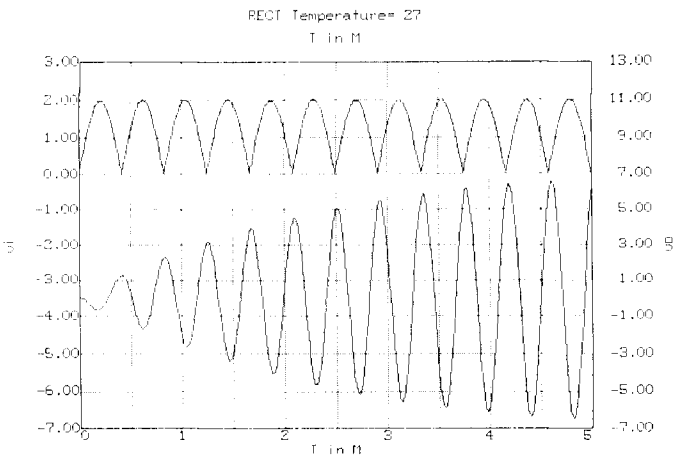


Fig 5—MicroCAP simulation of the filtered output of the rectified received signal. The upper trace is the output of the rectifier, while the bottom trace is the filter output (U4C pin 8). The increase in amplitude occurs because the simulation begins with no previous input signal present.

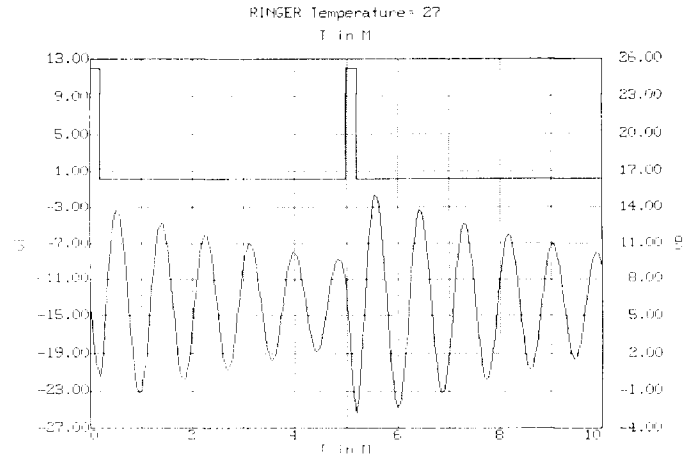


Fig 6—MicroCAP simulation of the output of the 1200-Hz bit-clock filter. The upper trace is the output of the edge detector, at U2A pin 3. The lower trace is the filter output, U1D pin 14. The edge detector output shows HDLC flags being received.

for proper operation of the phase detector.

The output of the 2400-Hz bandpass filter is also passed through a half-wave peak voltage doubler and filter to generate a dc-control voltage for the automatic gain control system. Deriving the control voltage from the filtered BPSK carrier results in an AGC that is virtually immune to noise and interference that could otherwise affect the AGC and lead to demodulator desense. The AGC control voltage also drives a front panel tuning and signal-strength meter.

BPSK Signal Detection

The second path the received BPSK signal takes is the one that actually extracts intelligence from the BPSK signal. A nonlimiting balanced linear phase detector, composed of operational amplifier U1B and CMOS switches U9B and U9C, is used to translate the BPSK input signal down to baseband. The phase detector is driven by the locally generated BPSK carrier supplied by U3B and offers high local carrier suppression without distorting the wave shape of the BPSK signal. It also helps to remove noise and other interference not in phase with the desired BPSK signal.

In this design, linear AGC is used rather than hard limiting to maintain a constant output voltage from the detector. As a result, strong input noise does not corrupt weak BPSK signals since there is no limiter and associated "capture effect" to suppress the weaker signal. Since the phase detector switches at the zero-crossing point of the sinusoidal BPSK waveform, the detrimental effects of phase jitter and transmitted phase noise are minimized.

The output of the phase detector is gently filtered through a second-order Butterworth low-pass filter to remove higher-order products generated by the coherent demodulation process. The frequency response of the filter loosely approximates that of the transmitted BPSK spectrum, and its smoothed output voltage is a function of how well the phase of the BPSK signal correlates with that of the locally generated carrier. A perfect correlation produces a maximum output voltage of one polarity, while a correlation of opposite phase produces an output voltage of opposite polarity.

NRZI Data Encoding

The AX.25 packet radio communication protocol uses a data encoding technique known as NRZI: *N*on *R*eturn to *Z*ero *I*nverted. NRZI synchronous data encoding packs both data and clock information into one binary serial data stream. An NRZI receiver is edge triggered rather than level triggered, and must therefore be sensitive to logic-level transitions to extract clock and data information from those transitions. With NRZI, a 0 data bit is encoded as a bit-level transition, while a 1 is encoded as no transition. The AX.25 protocol uses a process called zero insertion or "bit stuffing" that ensures that no more than five 1s can occur sequentially except when flag bytes are transmitted. Flags are used to identify the beginning and ending of each packet frame.

Zero insertion in combination with NRZI encoding guarantees that a logic-level transition occurs at least once every five bit periods. These frequent level transitions are necessary to allow the modem and TNC to synchronize with the transmitting TNC's clock.

Post Detection Filtering

In order to achieve the maximum demodulator performance possible, the detected NRZI encoded serial data stream must be filtered to remove as much noise as possible without introducing distortion that could lead to data corruption. The detected baseband NRZI serial data stream present after the Butterworth filter branches in two separate paths. One branch extracts and regenerates the 1200-Hz bit clock from the NRZI encoded data stream, while the other filters the NRZI data through an *integrate and dump* processor that forms a matched output filter.

Bit Clock Regeneration

The unprocessed NRZI-encoded serial data stream from the detector is shaped to a square wave of constant amplitude through op amp U7A and further buffered through exclusive-OR gate U2B. The NRZI data is then fed through an edge detector to extract clock pulses from the detected waveform. The edge detector acts as a frequency doubler by multiplying the

NRZI serial data stream by itself delayed by one half of a bit period. The R-C network composed of R11 and C7 provides a one-half-bit delay, while exclusive-OR gate U2A performs the multiplication. The edge detector produces an output pulse for every NRZI logic-level transition received. These pulses are in phase with the embedded NRZI clock signal and are filtered by a 1200-Hz narrow bandpass filter, U1D. This filter receives excitation from the edge detector and produces a damped sine wave by virtue of its high 'Q' and the flywheel effect. Its purpose is to use stored energy to fill in the gaps during periods when NRZI bit transitions do not occur and clock pulses cannot be recovered from the edge detector. This effect is shown in Fig 6. The damped sine-wave output of this filter feeds U8, an XR2211 PLL with a center frequency of 1200 Hz and a loop bandwidth of 15 Hz. The phase-locked loop follows the average phase and frequency of the filtered NRZI clock pulses providing the precise timing signals required for operating the integrate and dump processor.

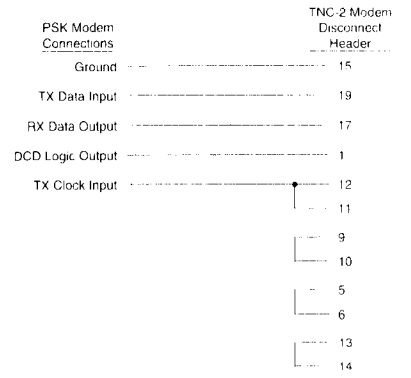


Fig 7—Connections between the PSK modem and a TAPR TNC-2 or clone. Consult your TNC manual to determine connections for other TNCs (see text).

Integrate and Dump

The maximum received signal-to-noise ratio (SNR) is achieved in a communications system when the bandpass of a receiver exactly matches the bandwidth of the transmitted signal. In order to achieve maximum signal-to-noise ratio, the post-detection filtering in this modem is performed with a filter matched to the AX.25 binary data rate of 1200 bit/s. Unlike simple R-C or L-C filters, the integrate-and-dump processor operates in the time domain and is implemented with a resettable integrator and an output latch driven by 1200-Hz clock pulses recovered from the 1200-bit/s NRZI-encoded data stream. With a matched filter, the demodulator's output signal-to-noise ratio is dependent not on the received signal-to-noise ratio, but rather on the ratio of the signal energy to the power spectral density of the noise at the filter's input. A matched filter allows the successful recovery of weak signals buried deep in wideband noise and offers the smallest error probability that can be achieved and the best bit-error-rate performance possible over an additive white Gaussian noise channel.

The unprocessed NRZI serial data stream from the phase detector is integrated over each bit interval and the result is sampled by an output latch. During each bit interval, random noise energy will accumulate no charge across integration capacitor C17, while an input signal coherent with the demodulator's internal BPSK reference carrier will accumulate a charge either above or below this point, depending on its phase. After approximately 90 percent of the bit interval has passed, the voltage integrated over this period is sampled and evaluated by comparing it to a reference voltage equal to half the supply voltage. Operational amplifier U7B, operating as a voltage comparator, performs this evaluation. The result is sampled and latched by U3A, the CD4013B output buffer. If the integrated voltage is above the $+1/2\text{-}V_{cc}$ reference, the output buffer latches to a logic level 1. If it is below this level, the CD4013B buffer outputs a 0.

Shortly after the output has latched, the integrator capacitor is discharged and the process repeats for the next data bit. This time-averaging process of integrating, sampling, latching, and dumping produces a well-filtered serial data stream retimed to the recovered NRZI clock signal. The output of the matched filter is then converted to TTL levels of 0 and +5 V through transistor Q1 and made available for processing by the terminal node controller.

Handshaking Controls

U5, the XR2211 phase-locked loop used for BPSK carrier regeneration contains an in-phase (I channel) detector to indicate whether or not the PLL is locked in frequency and phase with its input signal. Since the PLL's input in this case is a product of the BPSK carrier, the in-phase detector indicates the presence or absence of a valid BPSK carrier on the modem's input. U8, the XR2211 PLL used for bit clock regeneration, contains a similar in-phase detector. In this case, the detector indicates the presence of valid 1200-bit/s clock pulses extracted from the BPSK input signal. The active-low, open-collector outputs of both of these detectors are combined by connecting their outputs together. This combined output is pulled to ground level when no carrier and no clock pulses are detected, and to +5 V when both a carrier and a 1200-bit/s clock stream are present. The output is buffered by Q4 to drive an LED indicator, D10, and is made available to the TNC as a data carrier detect (DCD) control signal on the output of the modem. The DCD line connects to the TNC serial I/O circuits and provides on-the-air flow control and protocol timing. The use of combined carrier and clock detection results in a DCD that is very immune to noise and triggering from false signals.

Automatic Frequency Control

Radio links between satellites and ground stations experience Doppler effects due to satellite motion and the Earth's rotation. For an amateur-radio spacecraft in a low-Earth orbit of about 1000 km, transmitting on UHF, the amount of Doppler shift can be as much as 20 kHz, with a maximum rate of change of 40 Hz per second at the time of closest approach on an overhead pass. With the BPSK demodulator designed for a signal properly tuned to within a few tens of Hertz, it is necessary to use an automatic-frequency-control system to keep the downlink receiver properly tuned to a BPSK transmission during a satellite pass.

An internal voltage comparator in U5 that is normally used for FSK-decoder applications of the XR2211 controls the logic of the automatic frequency control circuitry. The comparator compares the PLL error voltage to a reference within the chip. If the error voltage exceeds limits dictated by the PLL's loop bandwidth, it is an indication that the BPSK signal is not properly in tune and the AFC circuitry is activated to effect a correction. If the signal is tuned too high in frequency, the comparator output voltage at pin 7 is pulled low, but only if a valid BPSK carrier is detected on the input of the modem. If the voltage

at pin 7 goes low, it allows the LM555-gated oscillator to run, stepping the downlink receiver lower in frequency to compensate for the Doppler shift. The receiver frequency is controlled through a connection available on the transceiver's microphone connector. Once the receiver frequency has been corrected, the comparator output voltage at pin 7 returns high, disabling the LM555 oscillator and keeping the receiver's frequency fixed until the next correction is required.

Note that the AFC circuit in this modem can tune the receiver in one direction only. Since the Doppler effect causes the signal received from a satellite in Earth orbit to drift lower in frequency during a pass and never higher, it is only necessary to have the receiver tune lower in frequency to compensate for the motion of the satellite. Front panel switch SW1 allows the operator to disable the AFC feature of the modem during manual receiver tuning.

Two digital AFC pulse polarities are produced by the modem. Output "AFC +" produces a positive output voltage with respect to ground every time the ground station receiver must be tuned. This polarity is consistent with that required for Yaesu FT-726R transceiver tuning. Others require a switch to ground, as provided by the "AFC –" output.

The ground-station receiver must be capable of tuning in 20-Hz increments or less. If this is not possible, it will be necessary to use an analog AFC approach whereby the modem controls the voltage applied across a varactor diode associated with the receiver's VFO.

Data Modulation

Full access to the digital transponders on the Pacsat satellites requires that ground stations use 1200 bit/s, 3.5-kHz deviation, bi-phase Manchester encoded frequency shift keying (FSK) for their uplink transmissions. This is produced by feeding Manchester-encoded binary data into the microphone connector of a standard 2-meter narrowband FM voice transmitter. The KD2BD Pacsat Modem produces Manchester code by modulating the 1200-Hz clock derived from the TNC with the 1200-bit/s transmit data in a balanced phase modulator. The output is then filtered to produce a clean output waveform that is low in harmonic distortion.

The TX clock available from the TNC modem-disconnect header is at 16 times the transmitted data rate, or 19,200 Hz for 1200-bit/s data. The modulator divides this clock signal by 16 and combines it with the transmit data in a phase modulator to produce a Manchester-encoded data stream. U10, a 74HC4040 ripple counter, provides the necessary frequency division. The divider is reset on the rising edge of the transmit data waveform through R70 and C34, a differentiator network. This synchronization keeps the divided clock waveform in proper phase with the transmit data and keeps the modulation switching transients at the zero-crossing points of the carrier waveform for minimum harmonic distortion. The 1200-Hz square wave from the divider is then filtered through a 1200-Hz bandpass filter designed around operational amplifier U12A to produce a low-distortion sinusoidal waveform. The 1200-Hz carrier is modulated by the transmit data from the TNC and the result is passed through a fourth-order Butterworth low-pass filter. The purpose of the low-pass filter is to reduce the sideband components of the output spectrum that are a result of BPSK modulation. The Butterworth filter provides an almost constant group delay across the entire modulator bandwidth, and this results in an output waveform having minimal zero-crossing point dispersion and phase jitter with modulation.

Alignment and Testing

Initial alignment of the KD2BD Pacsat Modem requires the use of a high-impedance voltmeter and an oscilloscope. The modem should be connected to the host TNC so an audio loopback test between modulator and demodulator sections can be performed.

Set all potentiometers to their center positions. Configure the TNC for a radio data rate of 1200 baud, and connect the TX clock and ground from the TNC to the modem. Apply power to both the modem and the TNC. Connect the oscilloscope to the modulator's high-level audio output, found at test point 1 (TP1), and verify the existence of a 1200-Hz sinusoidal pattern on the oscilloscope. Adjust R74 (500 Ω), associated with the bandpass filter in the modulator, for maximum sine-wave amplitude.

Connect TP1 to the audio input of the demodulator. Adjust the 2400-Hz filter tuning adjustment potentiometer R38 (500 Ω) until the tuning meter, M1, achieves maximum upscale deflection. With a dc voltmeter connected between pins 10 and 11 of U5 (XR2211), adjust R46 (10 kΩ) until the voltmeter reads zero volts. A 12-V peak-to-peak, 1200-Hz square wave should be present on pins 12 and 13 of U3B (CD4013B).

Attach the TX data line from the TNC to the modem. The TNC produces a series of AX.25 "flags" in its idle state that is sufficient for testing the clock extraction and regeneration circuitry of the modem. Using the oscilloscope, verify the presence of a pulse train on pin 3 of U2A (CD4070B). A damped sine wave should be present on U1D pin 14 (TL084N). Adjust potentiometer R14 (500 Ω) for maximum sinewave amplitude as seen on the oscilloscope. With a dc voltmeter connected between pins 10 and 11 of U5 (XR2211), adjust potentiometer R2 (5 kΩ) for a reading of zero volts. At this point, the DCD indicator should be on and the AFC indicator should be off. Trigger the oscilloscope's internal horizontal sweep to the recovered clock pulses present on pin 14 of U4D (TL084N). Monitor the waveform present on pin 8 of U7C (TL084N) and readjust potentiometer R14 for the tallest and cleanest pattern of right triangles seen on the oscilloscope. An "eye diagram" can be viewed by monitoring the waveform on pin 8 of U1C (TL084N) while triggering on U4D pin 14.

At this point, the modem is fully aligned and is ready for operation. You should be able to establish a packet connection with yourself at the keyboard as a verification that both modulator and demodulator sections of the modem are functioning properly.

Modem Operation

The KD2BD Pacsat Modem requires several connections to the host TNC as well as the ground-station radio equipment. Connections to the TNC's internal modem must first be broken so the following connections to the Pacsat modem can be made. Received data from the modem is directed to the TNC via the modem disconnect header available on many terminal node controllers. RX Data from the modem is connected to pin 17. DCD logic from the modem connects to pin 1. Transmit data from the TNC connects to the modem via pin 19, and the transmitter clock (x16) from the TNC attaches to the modem through pin 12. Fig 7 shows the connections for a TAPR TNC-2 or TNC-2 clone. Consult your owner's manual for the specifics of

connecting an external modem to your TNC.

Connections must also be made so the TNC can key the uplink transmitter. Modulator audio from the modem connects to the microphone connector of the uplink transmitter. AFC pulses from the modem trigger the downlink receiver tuning, to simulate a user pressing the microphone's "down" frequency button. Audio from the receiver connects to the modem's audio input. As with the TNC, consult the owner's manual of your ground-station radio equipment before making any connections to the modem.

Once connections between the modem, TNC and ground-station radio equipment have been made, on-the-air operation can begin. With the modem's AFC switch in the off position and the receiver in the USB mode, tune in a satellite transmitting 1200-bit/s BPSK. Slowly tune across the BPSK signal until the tuning meter achieves maximum upscale deflection. The tuning meter will indicate several peaks when tuning across a BPSK signal. Correct tuning is achieved when the receiver is tuned to the center of the highest peak. When properly done, the yellow DCD indicator should be on and you should be able to copy packets from the satellite on your computer terminal. Switch the AFC on to activate the modem's automatic frequency control. As the red AFC indicator comes on, the receiver should tune lower in frequency in compensation for Doppler shift. The modem is very sensitive and will successfully track BPSK signals barely audible through receiver noise.

Potentiometer R87 (5 kΩ) should be adjusted for a modem output audio level that produces approximately 3.5-kHz peak-carrier deviation of the uplink transmitter. Greater levels of modulation will cause the uplink signal to deviate right out of the 15-kHz-wide Pacsat uplink receiver passband.

To communicate with the satellite, adjust your uplink transmitter to one of the transponder uplink frequencies and set your TNC for full-duplex communication (FULLDUP ON). Sending a connect request should result in a connection to the satellite. The actual procedure for communicating with the satellite's mailbox will depend on the Pacsat being accessed and the ground station terminal software required for access. FO-20 operates in a fashion similar to a typical terrestrial packet bulletin board running PRMBS software and can be accessed without the need of Pacsat terminal software, such as *PB*.

Reception of WEBERSAT-OSCAR-18's CCD Earth images does not require the use of an uplink transmitter or the modulator portion of this modem, but does require the use of Microsat ground-station software and WEBERSAT image display software. The *Pacsat Beginner's Guide*, containing *Microsat Ground Station* software, and *Weberware* for use with OSCAR-18 are available from:

AMSAT-NA
850 Sligo Avenue
Silver Spring, MD 20910 USA

In Summary

The KD2BD Pacsat Modem was designed independently without the luxury of having seen other previously published Pacsat modem designs. It is the result of countless hours of research, experimentation and testing. Many different modem configurations were attempted over the design period with the one described here providing the best overall performance. One of the major design goals of this modem was to produce a modem capable of demodulating very weak signals. The performance demonstrated with this modem shows these design goals have clearly been met.

So, whether your interests are in viewing the world through the eyes of WEBERSAT, setting up an electronic-mail gateway with PACSAT, or just reading the latest issue of *SpaceNews* on OSCAR-20, you will find the KD2BD Pacsat Modem is a valuable accessory for your TNC and a welcomed addition to your OSCAR satellite ground station.

See you on the birds!

Bibliography

Goode, Steve, "BER Performance of TAPR TNC Modem", *QEX*, August 1983, pp 3-4.

Jordan, Edward C., *Reference Data for Engineers: Radio, Electronics, Computer and Communications*, 7th Edition, 1986, pp 24-5 to 24-6, 24-12 to 24-15.

Schwartz, Mischa, *Information Transmission, Modulation, and Noise*, 1970, pp 431.

Carter, Max, "Super Narrow-Band Techniques Equalize Power Inequity On 1750 Meters", *Communications Quarterly*, November 1990, pp 99-113.

Miller, James, "Data Decoder for UoSAT", *Wireless World*, May 1983, pp 28-33.

Digital Communications

Harold E. Price, NK6K

Deep Dark Depression, Excessive Misery

I've been reading the various packet-radio related newsgroups and mail lists lately, and I've been getting more and more depressed. Adrian Godwin on the TCP-GROUP mail list says it well: "In searching for some ancient history in the tcp-group mail archives, I was horrified at the short distance we've come since 1987. Pretty well all the significant topics of this year came up in 1987/88, and we seem to be no closer to putting many of the ideas into practice."

Since I've been in active in amateur packet radio for a long time, I can say that this is true all the way back to 1982. Others can probably push that back to 1979. Is there a rut, and are we stuck in it? Here is what I've seen in the last few weeks, and my proposed "solutions." If you have an opposing viewpoint, write in.

Operating Systems

In the old days, the argument was CP/M vs Apples. Now that you can run Linux (for free, $40 anyway) on a clone PC, the argument has resurfaced as UNIX vs anything else. As Phil Karn, KA9Q, proved with NET/NOS, don't waste time or bandwidth arguing that your pet system is the best. If you want someone to use it, write a viable application for it and let the marketplace decide. In the meantime, shut up and get to work. In my experience, operating systems don't sell applications, applications sell operating systems.

By the way, if you aren't enough of a propeller head to be directly involved in the UNIX vs everything else war, I recommend *The UNIX-Haters Handbook* by Simson Garfinkel, Daniel Weise and Steven Strassmann, published by IDG Books Worldwide, Inc. It should be in a bookstore near you. It will give you some insight into the issues involved.

Modems

Back in ancient times, I wrote an article called, "Where is My High Speed RF?" It is still true today. Although high-speed (9600 and above) is in use in some locations, it isn't widespread. I really thought we'd be getting into some pure-fun applications by now, and we're not. Why can't I see the coffee pot in the ARRL lab? Why can't I see if it is raining outside Phil Karn's window? Why can't I see the clean room where the next ham satellite is being built? Why can't I get a sound file containing the weekly Amateur Radio news? Why can't I see a graphic display of the number of packets per second on the fifteen packet channels used in southern California? You can see these things (the nonham equivalents anyway: the coffee pot at Cambridge, a window at Cal State Hayward, a lab in Japan, the geek of the week show on the Internet, the Caltrans real-time freeway speed map) with Mosaic on the World-Wide Web (WWW) on Internet.

None of these have any particular value (unless you're are stuck on the freeway) other than that writing them and using them are excellent learning experiences. See FCC part 97. How does this relate to high-speed modems? Maybe if we invent a killer app that begs for more speed, they will come. It worked in *Field of Dreams*, anyway.

External Versus Internal

This is the old TNC vs TSR (or driver, or kernel, or DLL) argument. In the TNC case, the network interface and software is placed in a box that is external to your main computing engine. All of the device control software and some or all of the network software runs in the external box. In the internal case, the network interface is directly attached to the local bus, and some or all of the device control software as well as the network software runs as part of the local operating system.

In the old days (1978-1984), hams didn't have affordable tools like a widely available hardware standard (such as the IBM PC "ISA" bus). We didn't have real operating systems, meaning ones that supplied standardized high-level services, such as network, GUI, or device control services. What most hams had was a real or glass TTY. The lucky few had a computer (S100 bus, Z80 CPU, 64k memory). The TNC was designed for the lowest common denominator, that is, just a TTY, no software at all. It is no surprise, then, that the first imple-

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

mentation to catch on was the external TNC. Here was a device for $249 that allowed you to talk with a radio at 1200 baud. At the time, telephone modems for 1200 baud were in the same price range; two-wire 2400 baud was major bucks.

Now, of course, a 14.4-kb phone modem, with fax, is $99. Operating systems can be "easily" extended to directly support packet networks. Yet, much of hamdom is still stuck at 1200 baud, with a device pretending it is talking to a glass TTY. The solution? We need an inexpensive device to convert analog to digital, and to get those bits inside our modern machines at high speed. Maybe it's an external box with a high-speed modem on one side and an Ethernet connector on the other. There are lots of cheap Ethernet boards for every type of computer. I don't like the direct bus-connect option because we'd need one for each type of computer, but there are still more ISA boxes out there than anything else. We don't need any new work on 1200-baud plug-in cards, that's for sure.

Chat Versus Network Applications

Every few weeks, someone makes the suggestion on a newsgroup or mail list that the group should be split into "us" (forward-thinking high-speed network users) and "them" (fat fingered packet/RTTY-chatting PBBS-using neo-Luddites). Recently, a new round of complaints has surfaced about the "standard" PBBS interface. It is slow, it is clunky, it is mired in 1984, but it has served a large part of the user base very well over the years. Part of the problem of advancing is the user-base problem. The perception is that a character interface (and a terse one at that) is all that the user community's hardware can support. That was true in 1984, it was less true in 1988, and it may be totally false now. The only remaining issue is the requirement for terseness. At 1200 baud (on a shared channel), there is little patience for fancy menus. Still, look at Mosaic. Look at a packet PBBS. There must be some middle ground.

Installed User Base

Upgrade or die. There, I've said it. Part 97 requires technical training, improvement, advancement. It does not mandate a common-carrier service. Those who want to use old equipment will still have room to play. We're still using AM on the ham bands, after all. Ten years is long enough to be tied down. Implementors should not feel fettered by the old ways and should stop wasting bandwidth worrying about it.

Opposing Viewpoints are Solicited

If you disagree (or agree) with any of the above, please write to my email or US mail addresses. If you have an interesting application that is running on the packet network, let me know. Aside from some character-based white pages, I haven't seen much. We seem to be working hard, but I'm not sure we're having any fun.

Virus

And now for something completely different. A virus was passed around the AMSAT mail list on the Internet in late June. Someone captured a short audio bite from DOVE, converted into .WAV format, zipped it, uuencoded it and sent it in an email message. Unfortunately, he also included a copy of a program to play .WAV files on DOS through a PC speaker. That program had a virus. Several people were infected. (At least, their machines were.) The virus was quickly noticed as one of its side effects was to keep some TSRs from loading properly.

This virus caught the people who never use virus protection, of course. It also caught people who do use it, but hadn't upgraded their virus protection lately. The standard McAfee virus scanner's March version did not detect the virus, though the newer June version did.

Here are my tips on virus avoidance. First, always run a virus scanner on any executable files you download (or that show up in your in box). There are many available. A good shareware version is McAfee SCAN, available on Compuserve (go index, search for virus), on Internet (mcafee.com) or via their BBS at 408-988-5190. Next, keep that scanner up to date. Most virus scanners look for sequences of bytes that are unique to the virus. These ID strings must be updated each time someone comes up with a new virus.

Even better, don't use downloaded executable files. This is how I avoided getting the virus. Even though I decoded and unzipped the file, I already had a program to play .WAV files. I didn't run the infected executables and immediately deleted them. Even running executables from a trusted friend can give you a virus, as that friend may have also been caught using an old scanner.

Finally, friends don't let friends send executables. The only thing more embarrassing than getting a virus is passing one on. In the most recent case, there was little need to send out the executable to play the sound file. The sound was in a standard format, and players are readily available from other locations. While you can get a virus from Compuserve, an Internet archive, or other on-line service (and you must scan their files, too), viruses are quickly found and reported due to the large number of users on the big services. If you must send an executable, think about what you are doing, and scan with the latest programs, even if you "know" your system is clean. In the case of the AMSAT-BB file, people were infected on four continents in 24 hours. Let's be careful out there. □□

Upcoming Technical Conferences

1994 ARRL Digital Communications Conference

August 19-21, 1994, Thunderbird Hotel and Conference Center, Bloomington, Minnesota. Sponsored by the TwinsLAN ARC.

Contact: Paul Ramey, WGØG, 16266 Finland Avenue, Rosemount, MN 55068, tel: 612-432-1149 evenings and weekends. Or, Carl Estey, WAØCQG, 276 Walnut Lane, Apple Valley, MN 55124, tel: 612-432-0699. Internet: estey@skyler.mavd. honeywell.com; packet: WAØCQG@WAØCQG.#MSP .MN .USA.NA.

Events: Friday afternoon, registration, Hospitality Suite and informal demonstration. Saturday, presentation of technical papers, buffet luncheon (included), "birds-of-a-feather" forums, informal demonstrations, optional buffet dinner, technical showcase—TAPR special interest groups and ADRS DSP presentation. Sunday, informal demonstration and conference wrap-up. For those not attending the Conference, outings of interest to all family members are being arranged.

Registration: Conference registration is $45 and includes Saturday's luncheon and a copy of the Conference Proceedings. Registration deadline is August 12, 1994. Saturday evening's buffet is an additional $20. Please call Paul Ramey, WGØG, for registration forms.

Hotel/Etc.: Best Western Thunderbird Hotel and Convention Center, 2201 East 78th Street, Interstate 1-494 at the 24th Avenue exit, Bloomington, MN. Free shuttle service is available to Minneapolis/St. Paul international Airport and Mall of America. You can call the DCC Hotline at 800-726-6715 to make reservations and get more information on area events. Be sure to ask for Cathy Thomas.

Eastern VHF/UHF/SHF Conference

August 26-28, 1994, Quality Inn and Conference Center, Vernon, Connecticut.

Contact: Byron Blanchard, N1EVK, 16 Round Hill Road, Lexington, MA 02173, tel: 617-862-1380 (evenings) or Stan Hilinsky, KA1ZE, 17 Pilgrim Drive, Tolland, CT 06084, tel: 203-649-3258 (W), 203- 872-6197 (H).

Events: Friday, Hospitality room. Saturday, registration, formal talks and bandsessions, preamp noise figure measurement competition, banquet (separate registration, $21), and VHF/microwave Trivia Quiz. Sunday, swap meet and antenna measuring. Shopping, movie theater and amusement area is on-site.

Registration: Preregistration, before August 19, is $20, Sunday only is $5. Registration at the door is $25.

Hotel/Etc: Quality Inn and Conference Center, 51 Hartford Turnpike, Vernon, CT 06066. Call Lori Tozier at 800-235-4667 and mention the Eastern VHF/UHF/SHF Conference. Rates are $55 for single and $64 for double and include a continental breakfast.

Microwave Update '94

September 22-24, The Inn at Estes Park, Estes Park, Colorado.

Contact: William McCaa, KØRZ, PO Box 3214, Boulder, CO 80307-3214, tel: 303-441-3069.

Papers: All information on amateur microwave activity on 902 MHz and up is of interest. If you're interested in preparing a paper, please contact Al Ward, WB5LUA, at 214-699-4369.

Events: Thursday evening, informal get-togethers and registration. Friday and Saturday morning, technical sessions; Friday evening, swapfest, microwave band measurements of noise figure measurements, scalar transmission and reflection, spectrum and power. Everyone is encouraged to bring along your microwave equipment for testing. Saturday afternoon, free time to enjoy the Colorado mountains. Saturday evening, dinner at the *Barleen Family country Music Dinner Theatre.* Following dinner, we will be returning to the Inn at Estes Part for a "Visit My Station" session (bring slides and VHF video tapes for all to see.

Although no spouse program has been planned, the Estes Parks area offers many shops and interests for spouses. A list of spouses based on preregistration will be available to facilitate potential gatherings.

Registration: Preregistration cost is $35 and will be accepted through August 15, 1994. After August 15, all registrations will be $45. Walk-ins are welcome.

Hotel/Etc: The Inn at Estes, 1701 Big Thompson Avenue (Hwy 34), PO Box 140, Estes Park, CO 80517, tel 303-586-5363 (in Colorado), 800-458-1182. They are offering a special rate of $62/night. Rooms are limited and the closing date for reservations is September 1, 1994. Please be to mention the Microwave Update 1994 Conference to receive the special rate.

Pack Rats Conference

A date has not been set at this time, although it's usually the first Saturday in October.

Contact: John Sorter, KB3XG, 5290 Stump Road, Pipersville, PA 18947.

The 1994 AMSAT-NA Annual Meeting and Space Symposium

October 7-9, 1994, in Orlando, Florida.

Contact: Steve Park, WB9OEP, 12122 99th Ave N, Seminole, FL 34642, tel: 813-391-7515. Internet: SKPA@QMGATE.ECI-ESYST.COM; Am Pkt Radio: WB9OEP @ W4DPH.

Call for papers: A call for papers and presentors has been announced. Come help us celebrate 25 years of AMSAT in Space. Share your experience with the amateur satellite community. We need your help to be a success. Advance your reputation. Editing, formatting, graphics and even typing can be provided. Even if you cannot attend, consider a paper for publication. Topics for all amateur satellite disciplines are sought. Author, title and a short abstract are needed as soon as possible. Final drafts are requested on or about August 26. Direct inputs and inquiries to Steve Park, WB9OEP, at the above address. Join us in Orlando, FL, for the fun and festivities.

(Have an upcoming technical event? Drop us a note with the all the details and we'll include it in Upcoming Technical Conferences.) □□