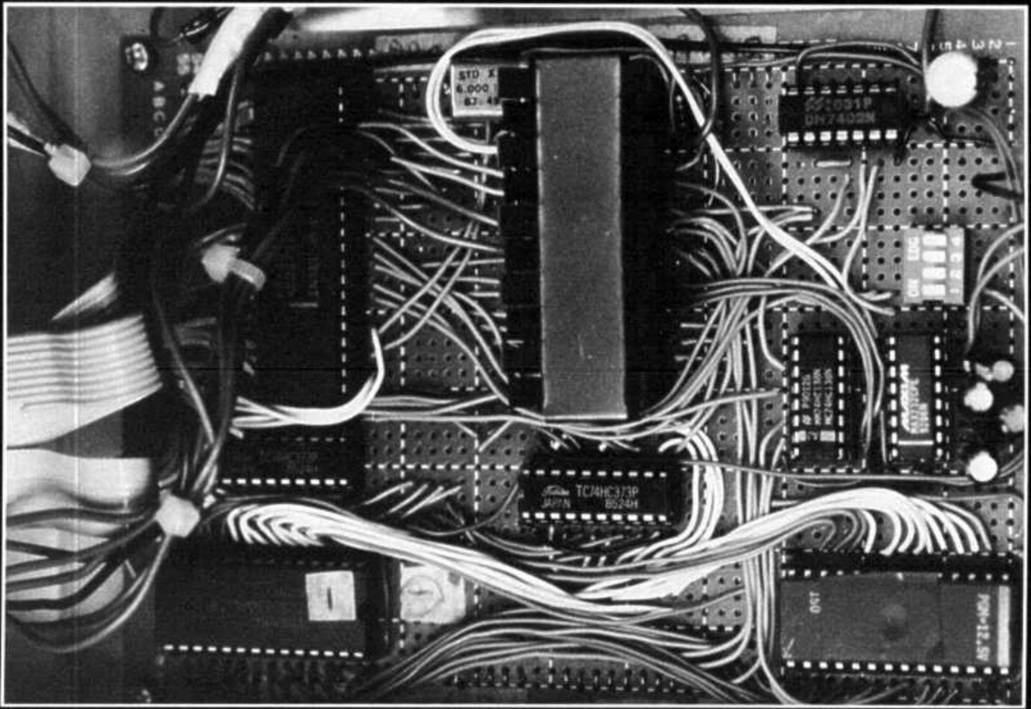
QEX

$1.75

ARRL Experimenter's Exchange

November 1993



Build this 8085 Microcomputer
for your next project!

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

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

Application to mail at second-class postage rates is pending at Hartford, CT.

David Sumner, K1ZZ
Publisher

Jon Bloom, KE3Z
Editor

Lori Weinberg
Assistant Editor

Harold Price, NK6K
Zack Lau, KH6CP
Contributing Editors

Production Department

Mark J. Wilson, AA2Z
Publications Manager

Michelle Bloom, WB1ENT
Production Supervisor

Sue Fagan
Graphic Design Supervisor

Dianna Roy, Deborah Strzeszkowski
Senior Production Assistants

Advertising Information Contact:

Brad Thomas, KC1EX, *Advertising Manager*
American Radio Relay League
203-667-2494 direct
203-666-1541 ARRL
203-665-7531 fax

Circulation Department

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

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

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

Elsewhere by Airmail: ARRLMember $48 nonmember $60.

QEX subscription orders, changes of address, and reports of missing or damaged copies may be marked: *QEX* Circulation. 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:
Perfboard construction works fine for this 8085 microcomputer by N4UAU.

141

Features

Columns

November 1993 QEX Advertising Index

Brian Beezley, K6STI: Cov III
Communications Specialists Inc: 23
Down East Microwave: 19
Henry Radio: 24
L.L Grace: Cov II
LUCAS Radio/Kangaroo Tabor Software:12
P.C. Electronics: 19
Rutland Arrays: 12
Yaesu: Cov IV
Z Domain Technologies, Inc: 23

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

Tales from the Crypt-o

One of the oldest rules regarding ham radio is that hams are not to transmit encrypted communications. The idea, of course, is that hams aren't supposed to be communicating anything that *needs* to be hidden, and allowing hiding of content opens the door to abuse of the amateur bands. (There is an exception made for satellite control links; we do need to have those secure!)

That doesn't necessarily mean that there is no place in Amateur Radio for cryptographic techniques, however. Using techniques detailed in the *5th Computer Networking Conference Proceedings* by Hal Feinstein, WB3KDU, and in the 6th *Proceedings* by Phil Karn, KA9Q, and elsewhere, cryptographic techniques can legitimately be used by amateurs for purposes of *authentication*. As used here, authentication means ensuring that a transmission, or a message, relayed via amateur digital communications originates from the station it appears to originate from. The techniques suggested in these papers and elsewhere don't require sending any encrypted data, at least, anything that isn't also sent in the clear.

Why go to all this trouble, though? Isn't Amateur Radio a hobby, after all? Is a concern for network security and tracking part of "real ham radio?" In brief, yes, it is. It is because what we amateurs want to do is build a network that serves our communications needs. And any network that serves those needs well will allow communication of violative communications (to use the FCC's phrase) as easily as legitimate communications. If we don't want to be backed into a regulatory corner that restricts us so much that an efficient network becomes impossible, we have *got* to solve the problem of identifying the source of illegal data in the amateur network.

At present, I can easily change the MYCALL on my TNC to another call sign—yours, say—and then log on to a PBBS that will unquestioningly accept that I am you. Imagine, though, if you could send a message that the recipient—any recipient—could verify was from you regardless of how the message entered the network. And any attempt to forge a message "from" you was detectable, as was any tampering with the text of a message you actually sent. A network that provided that kind of security as a routine operation would make a powerful case that we amateurs were in control of our facilities and could prove it.

Such a network is possible. Modern public-key cryptographic techniques can provide digital "signatures" of messages. These signatures are data blocks appended to the end of a message that act as a checksum to test the message integrity. But they are checksums that only the specified originator can create. Thus if I receive a message from you and the signature checks out, I can be certain the message is from you and it was not altered enroute.

The challenge will be to implement this kind of authentication on amateur networks. The logical first step—and the easiest one—is to provide authentication within BBS systems. One can at least ensure that messages traverse the BBS network uncorrupted. And if the BBS SYSOP can ensure that the message originator is who he says he is, again using a cryptographic technique, such as that employed by the latest version of the AA4RE BBS program, a pretty high degree of certainty as to the message source is possible.

If you would like more information about this, I suggest getting a copy of *Answers to Frequently Asked Questions About Today's Cryptography*, from RSA Laboratories. This document is available via the Internet on host rsa.com in the /pub/faq directory. A freeware copy of RSA's RSAREF package of free C source code for cryptographic routines is available on the same machine in the /rsaref directory.

This Month in *QEX*

Sam Ulbing, N4UAU, wanted a building-block microprocessor board to use in developing projects—so he designed one. The result can be found in "An 8085-Based Computer System."

Gary Sutcliffe, W9XT, provides a primer on how to use the PC parallel port as a general-purpose digital interface in, "Simple and Inexpensive PC Interfacing."

Part 5 of our series on automatic link establishment (ALE), by Paul C. Smith, K3ZMO, and Dennis Bodson, W4PWF, introduces a (relatively!) low-cost approach to ALE.

In this month's "RF" column, Zack Lau, KH6CP/1 presents a simple, state-of-the-art 13-cm PHEMT preamp design. And a book review from Ed Hare, KA1CV, rounds out this month's menu of goodies.—*KE3Z, email: jbloom@arrl.org (Internet)*

An 8085-Based Computer System

Here's a design you can use as a basic building block microcomputer—or just use it to program EPROMs.

By Sam Ulbing, N4UAU

Introduction

You have probably read articles in *QST* or *QEX* describing various circuits that use microprocessors: digital filters and memory keyers are just two of the recent projects. With the cost of these chips decreasing every day and their availability increasing, you can be certain that they will find their way more and more into amateur use.

Unfortunately, if you try to develop your own microprocessor system to control a project, you will soon discover hurdles to overcome. You need to buy or build a host micro system for development; you need to write a basic operating system to control the host system; and you need to be able to program EPROMs to use any programs you develop. All this before you can even start on the project you really wanted to do in the first place!

This article will make the job much easier and less expensive for you. It describes a general-purpose microprocessor system which can communicate with a PC, control many different hardware systems and program EPROMs. The photo on this page shows a proto-

The 8085 system with the EPROM program module in place.

type of the system being used to program an EPROM.

The system is modular, which provides great flexibility. (See Fig 1.) A PC interface is provided via an RS232 port. A second interface is provided via a 44-pin edge connector. Various modules can be plugged in here and run by downloading software from the PC to the 8085. With the hardware shown and the menu-driven software provided you will be able to program 27C64 (or similar) EPROMs. Just insert the EPROM module into the 44-pin socket, plug the system into a PC with a terminal program, start up the 8085 and follow the menu, shown in Fig 2.

The menu of Fig 2 lists a number of selections in addition to the EPROM programmer. These might be of inter-

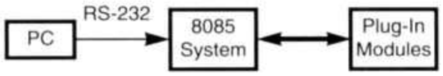


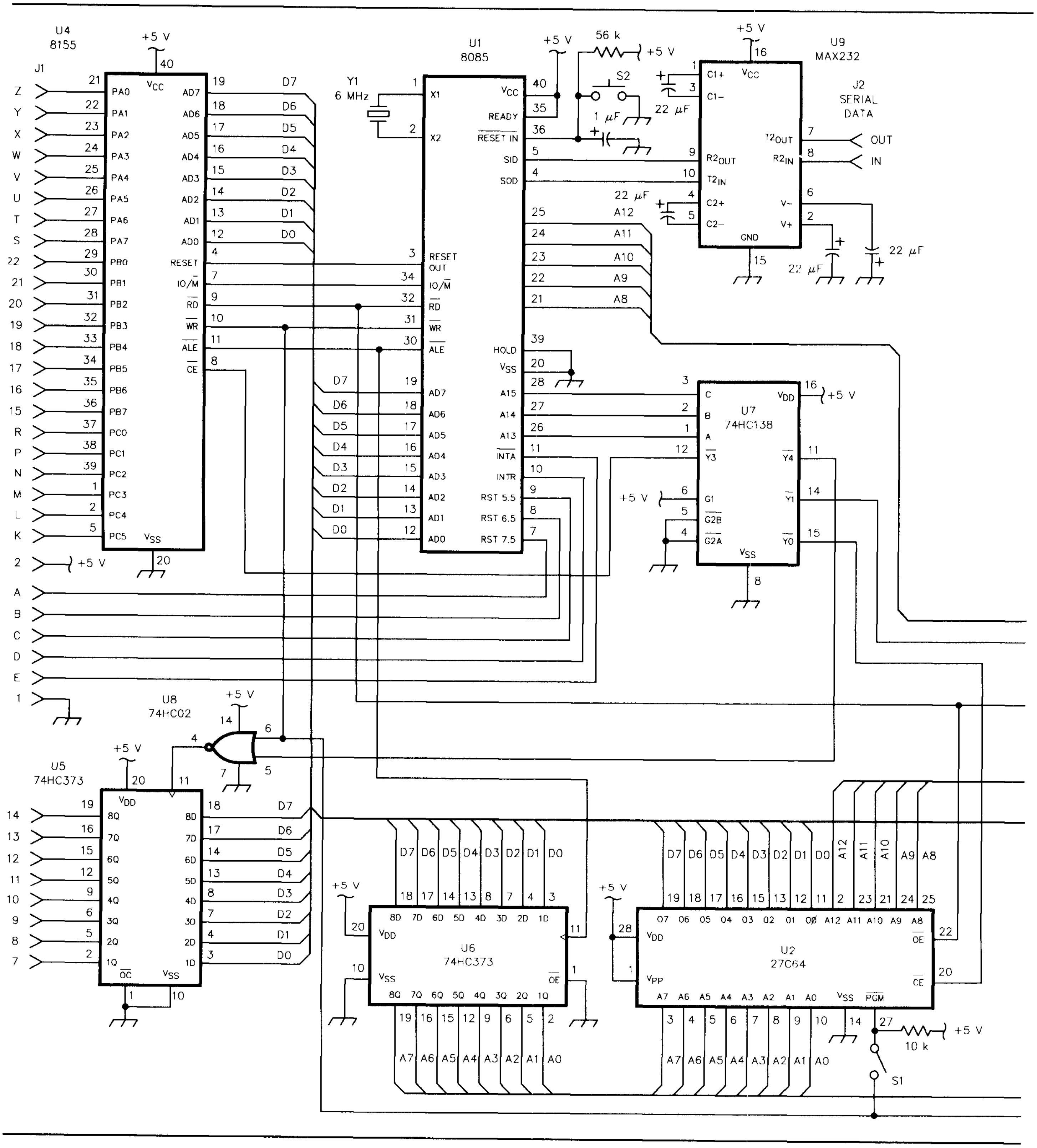
Fig 1—8085 system block diagram.

```
TYPE A LETTER FOR YOUR
CHOICE NOW OR TYPE M FOR
THE MENU

MENU SELECTIONS ARE:

  I = INTRO PROGRAM
  M = MENU
  E = EDIT
  O = OBJECT LOADER
  P = EPROM PROGRAMMER
  B = BLOCK MOVE
  U = UPLOADER
  D = DUMP REGS
  S = ASCII LOADER
  R = RUN PGM
  G = BCDNUM GAME
```

Fig 2—8085 system menu.

est to the person who wants to do development using this system. Since the system has a basic operating system already written, it is necessary to write only the software for the specific project you have in mind and build the

5305 NE 57th Lane
Gainesville, FL 32606



hardware needed for your project on a 44-pin board. Some of the projects I have built for use with this system are: a game, a moving LCD dot-matrix display (like you see at some road construction sites) and a module to program the 8751 40-pin microprocessor, which I'll describe in a future article.

The 8085 System Hardware

Fig 3 shows a schematic of the system. The heart of the system is an 8085, U1. This Intel microprocessor has an 8-bit CPU. That is, data is processed 8 bits (1 byte) at a time. A 6-MHz clock crystal results in a system clock speed of 3 MHz.

U2 is an EPROM that contains the basic software necessary to operate the computer. U3 is a static RAM chip that

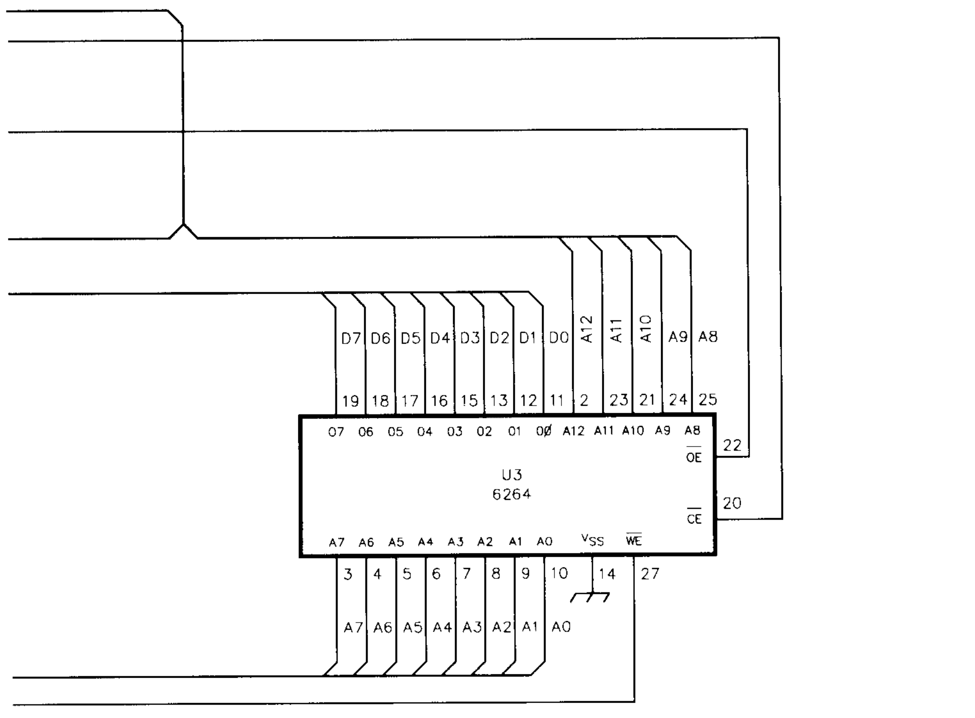


Fig 3—Schematic of the 8085-based computer system.
U1—8085AH2 8-bit HMOS microprocessor
U2—27C64 8-k × 8 CHMOS ROM
U3—6264 8-k × 8 SRAM
U4—8155 programmable I/O device with RAM and timer
U5—74HC373 three-state address latch
U6—74HC373 three-state address latch
U7—74HC138 address decoder
U8—74HC02 quad 2-input NOR gate
U9—MAX232 serial interface

	Address Bit Number			*Hex Address Range*	
	15	*14*	*13*		
ROM-U2	0	0	0	0000	1FFF
RAM-U3	0	0	1	2000	3FFF
NOT USED	0	1	0	4000	5FFF
PIA-U4	0	1	1	6000	7FFF
LATCH-U5	1	0	0	8000	9FFF
NOT USED	1	0	1	A000	BFFF
NOT USED	1	1	0	C000	DFFF
NOT USED	1	1	1	E000	FFFF

Fig 4—8085 system memory map.

provides up to 8 kbytes of temporary data storage. U4, an 8155 programmable interface adapter, provides three buffered I/O ports (two 8-bit ports and one 6-bit port), 256 bytes of RAM and a timer. The ports are parallel ports, so data is sent out simultan-eously on multiple lines. A three-state latch, U5, provides the system with an additional 8-bit output port. U6 is a three-state latch that demultiplexes the address and data. It is used because the 8085 can address up to 64 kbytes of data, which requires 16 address bits. Since there are not enough pins available on the 40-pin package for 16 dedicated address lines, the 8085 multiplexes the low 8 bits of the address and the 8 bits of data on the same set of lines. Pin 11 of U6 is strobed by the 8085, causing it to latch the address bits and hold them while data is put out on these same lines.

U7 is an address decoder. The system uses inexpensive 8-k memory modules which require only 13 lines for an address. The highest 3 bits of the address from the 8085 are decoded by U7 to activate one of up to 8 different chips. Fig 4 shows the memory mapping of the system.

U9 is an RS232 transmitter/receiver. It is the serial port of the system and provides the communications interface to the RS232 port of a PC.

Switch S1 permits or inhibits a write signal to the EPROM address. When an EPROM is used, S1 has no function, but I have often found it helpful to use a Dallas 1225 nonvolatile RAM in the ROM socket for developing software. If an EPROM is used, the program to be tested must be loaded and tested in RAM space with different addresses or programmed into an EPROM and then moved to the ROM socket for testing. With the nonvolatile RAM, and S1 closed, you can directly load programs to ROM space for testing. Of course you can also wipe out everything in the ROM space, so care must be taken! As soon as I load the program I open S1 to disable writing to ROM space.

This system is about the simplest general purpose computer you can build with an 8085, but it is really quite powerful. With a 3-MHz clock, it will execute more than a half million instructions per second, and the 8 k of ROM will hold about 4000 instructions. 22 I/O lines, 8 output-only lines and 5 interrupt lines allow good parallel communications with the project hardware, while the RS232 port allows communication with a PC.

The physical layout of the system is shown in the photo on the cover. I tried to lay it out to permit future growth. The board I used is a Radio Shack 276-147 universal board. While a custom PC board would be neater, I had no desire to go to the effort of making one. Besides I wasn't sure exactly what my final system would look like. I decided to use "ugly" computer construction (ie, soldered wires). It may look ugly, but it runs beautifully and no one can see the way it looks when it is in the box anyway. I ran bare bus wires on the underside of the board for the address and data lines. This did two things for me: (1) If I ever want to expand ROM, RAM or I/O ports, all I need do is attach another board to the end of the first and continue the bus wires across the next board. (2) It reduces the mass of wires somewhat. Of course, I used sockets for all the ICs. I selected the fast, low-power HC family of CMOS logic since the cost for the "best" was insignificant.

The I/O ports from U4 and U5 connect to a Radio Shack 276-1551 44-pin connector socket mounted on the top of the "8085 main frame." This allows many different circuits to be connected to the 8085 just by building a circuit on a 276-154 circuit board and plugging it in. Besides the I/O ports and the system interrupts, a switched 5-V source and the 8085 ground are available at the socket. With the already large current draw of the mother board, I would recommend running only low-current systems from this source. I did not connect the U4 timer outputs to the socket, but you could.

This system draws about 200 mA at 5 volts. Power is provided from a 7805 voltage regulator in a TO-220 case that is driven from a 12-V source. The power lost in the voltage regulator is fairly large, 1 to 2 W, so have it well heat sinked. Of course, any stable, well-regulated 5-V source could be used. I also used a heat sink on U1 as it runs a little warm. To keep smoke from coming out of the project box, I put diodes in the 12-V line.

I got my aluminum box from my "junk" collection. It was open at the top, so I used some hardboard to build a top surface. The 44-pin socket is mounted on this. The RS232 port is a 9-pin socket in the side of the box. Only the send, receive and ground terminals are used. An on-off switch, power-on LED indicator and a reset button (S2 in Fig 3) are mounted on the box.

The EPROM Programmer

The 27C64 EPROM programmer is built on a 276-154 44-pin board so it can be inserted into the socket of the 8085 system. The EPROM to be programmed is installed in a zero-insertion-force (ZIF) socket to permit easy insertion and withdrawal. The programmer board communicates with the 8085 via the four I/O ports. The 8155 and the latch provide the data and address lines that connect directly to the EPROM. The control lines for the rest of the hardware come from the 8155.

To understand the hardware, it is useful to know the programming algorithm. After the EPROM has been verified to be blank (all bits are 1), the sequence of programming is:

1. Address and data are applied to the EPROM.
2. V_{cc} is raised from 5 volts to 6.25 volts.
3. V_{pp} is raised from 5 volts to 12.75 volts.
4. $\overline{CE}$ goes from 5 volts to 0.
5. PGM goes from 5 volts to 0 for 100 microseconds to program the data byte. This is the "quick pulse programming" method.
6. OE goes from 5 volts to 0 to permit read-back verification that the byte was successfully programmed.
7. $\overline{OE}$ is returned to 5 volts.
8. If the programming was not successful, steps 6 and 7 are repeated up to 25 times.
9. Upon successful programming, the data and address are incremented and steps 5 through 8 are repeated for the rest of the bytes to be programmed.
10. V_{cc} and V_{pp} are lowered to 5 volts.
11. $\overline{OE}$ is set to 0 and the entire EPROM is verified.
12. The EPROM is passed or failed.

The schematic shown in Fig 5 will accomplish this process in conjunction with the software in the 8085. Power is supplied by a 13.5-volt RMS ac wall transformer, and the bridge rectifier and filter provides around 17-volts dc with little ripple. The data sheet for the 27C64 shows that as much as 60 mA can be drawn during programming, so size parts accordingly. Q1 controls the power to the circuit. The 27C64 data sheet indicates that V_{cc} and V_{pp} need to be within ±0.25 volts of nominal. While I have seen designs that control these voltages through NPN transistors, I felt that better control of the voltage values would be obtained by using a circuit shown in the National Semiconductor literature. The output voltage of an LM317 three-terminal regulator depends on the ratio of resistors used between the output, the adjust pin, and ground. A simple switching circuit with NPN transistors will quickly and accurately change the resistance in the circuit and hence the output voltage. U2 and U3, and Q2 and Q3, perform this way. The transistors are driven from an LM339 comparator that senses the state of the input control lines. This allows the switching level to be set between logic high and low levels (at around 1.5 volts). Additionally, the LM339 minimizes current draw from the 8085 output lines.

It is important to note that any voltage over 13.0 V at V_{pp}—even a momentary glitch—can destroy the EPROM. Four 0.1-µF capacitors are used to eliminate transient overshoots during switching. To avoid transients at start up, start the 8085 EPROM program first so that the voltages on the control lines are stable. The EPROM must not be inserted or extracted while power is applied to the ZIF socket. The green LED, D2, shows when the power is on, and the red LED, D1, indicates when power is present at the ZIF socket. Programming takes around 3 seconds once the EPROM has been socketed and verified.

Using the Machine

To use the 8085, hook it to your PC via the RS232 port, put your PC in the terminal mode using a standard terminal program, turn on the power to the 8085, press RESET, then tap the space bar. The menu will be displayed. Tapping the space bar sends an ASCII space character (20_{16}) to the PC, which it uses to determine the baud rate of the host computer. I have found 4800 baud to be a good rate. I have used both my NEC Multispeed laptop and my 486-25 with *Windows* as hosts. The laptop has a simple terminal program in it, and it worked immediately. The 486 worked, too, except that when I sent a file via the object loader the 8085 hung up. With some trial and error I discovered that the *Windows* terminal program will "strip LF after a CR" unless you override it. I did so, and now it works great. Now that the 486 works, I don't use the laptop much at all! I use the *Windows* Notepad to write my source code, toggle to DOS to assemble the code, toggle to the terminal program to load and run it and then, too often, toggle back to the note pad to correct programming errors I made! I must say I feel a sense of awe and pride that my simple little hand-wired 8085 can work so well with a state-of-the-art 486.

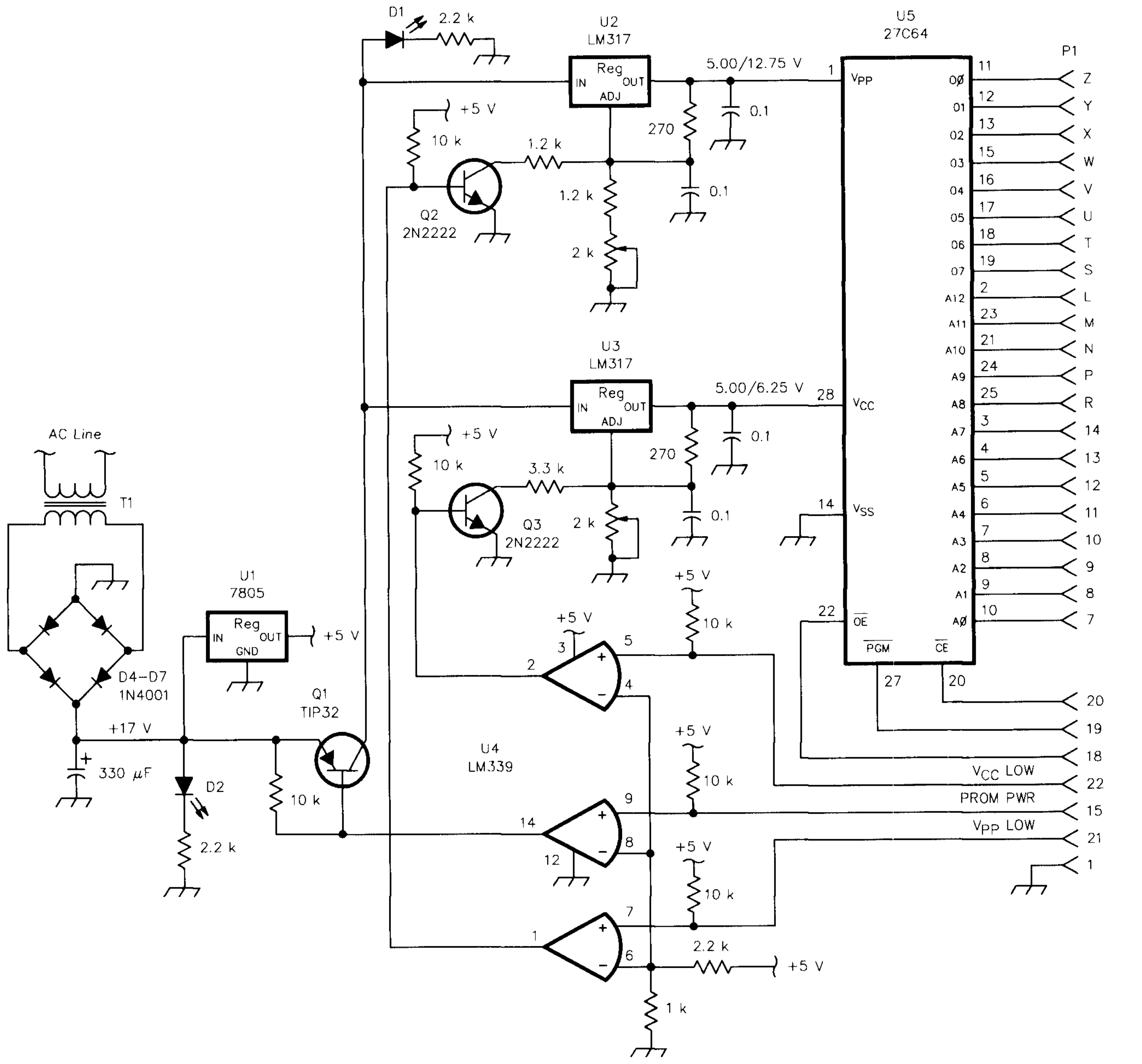


Fig 5—Programmer board for 27C64 EPROMS. Use 10-turn potentiometers.

If you want to use this system only to program EPROMs, you do not need to learn any 8085 code. Just follow the menu and you will be able to program them. Fig 6 shows the PC screen for a typical EPROM programming session. If you want to use the 8085 for other purposes, the other routines on the menu may be of interest. The main programs in the system are listed below:

1. Menu program—lists the menu.
2. Edit—permits the user to view and modify the hexadecimal code at a specified location in memory. The program requests a starting address from the user, then lists the address and the data byte at that location. If the user enters a carriage return, the next address and its data are listed. Enter a 2-digit hex number to modify the byte in memory. Enter minus and a 1-digit number to backspace the memory location being displayed. ESC terminates the edit program.
3. Object loader—loads Intel hex format code. The loading address is specified by the source code through the assembler's ORG command at compile time. It is sometimes useful to load at an address different from the compiled location (eg, when loading code to RAM for later use in programming an EPROM). The object program will request an offset in kbytes from the compile location and then load the code at the location specified by the sum of this value plus the ORG address.
4. EPROM programmer—provides the necessary control signals to program a 27C64 EPROM. The data for programming must be already loaded to RAM memory 2000H to 3FFFH (use the object loader program). The program checks to see that the EPROM is blank and

AN EPROM PROGRAMMING SESSION

User response is on left side, system response on right. My comments are in small letters.

The object code is loaded to RAM (object loader). In this case it was assembeld for location 0 so a 2-k offset is requested for loading to RAM.

```
        TYPE A LETTER FOR YOUR CHOICE NOW
         OR TYPE M FOR THE MENU
O
        WHAT COMPILE/LOAD OFFSET IN K BYTES
2
        SEND FILE NOW. DO NOT TOUCH KEYBOARD UNTIL FILE LOADED

        FILE IS NOW LOADED
```

The EPROM programmer is run. Repeated commands are an attempt to reduce operator error.

```
        TYPE A LETTER FOR YOUR CHOICE NOW
OR TYPE M FOR THE MENU
P
         IS PROGRAM IN RAM?
Y
        VERIFY POWER ON & EPROM NOT SOCKETED
Y
        VERIFY POWER ON & EPROM NOT SOCKETED
Y
        INSERT EPROM IF LITE NOT RED
Y
         INSERT EPROM IF LITE NOT RED
Y
        EPROM IS BLANK

        ARE U SURE U WANT TO PROGRAM EPROM?
Y
        ARE U SURE U WANT TO PROGRAM EPROM?
Y
         EPROM IS PROGRAMMED

        REMOVE CHIP
```

Fig 6—Example EPROM programming session.

aborts if it is not. Data is copied from RAM to the EPROM a byte at a time starting with the data at 2000H and continuing to location 3FFFH. The program verification then compares the data in the EPROM to the data in RAM.

5. Block move—copies a specified block of data from one memory location to another.
6. Uploader—copies a specified memory area to the RS232 port. I have used this to document and verify the code in the 8085. Your PC must be able to receive the data at the rate at which the 8085 sends it since there is no handshaking to stop the 8085 until it is done.
7. Dump regs—dumps the registers. Register a first, then the flag register in bit format, then registers B, C, D, E, H and L. It works, but I find it of limited value as a debugging tool. If a program hangs up, it is necessary to reset the CPU and most register data is lost. Periodic calls to dump in a program under debug might aid the process.
9. ASCII loader—loads a file exactly as it is in ASCII. I used it to view the ASCII contents of the Intel hex format and discovered the presence of a CR at the end of each data line. This was not visible in the PC print out.
10. Run—requests an address in the 8085 and then transfers control to that location.
11. BCDNUM game—a numbers game similar to NIM. It requires a plug-in game board module I built. I wrote this because it provides a noncomputer literate person with an easy to understand demonstration of the capabilities of the machine, and I wanted to impress my friends!

The programs above have worked for me so far. They are not optimum or elegant, but they have enabled me to do many things with my 8085. If I ever wish to fancy up the code, I can use this code to load and program a new EPROM with all the bells and whistles.

I will be glad to send anyone interested my source code on a 3.5" or 5.25" disk for $5. I can also provide a preprogrammed EPROM for $12. The charges are only to cover my costs. You can also get the program from the ARRL BBS (203 666-0578) or via the Internet from ftp.cs.buffalo.edu in the /pub/ham-radio directory. The file name is 8085SYS.ZIP.

Where To Go From Here

If you have read this far you may be interested in developing some of your own modules. That will require learning 8085 assembler code (or a suitable higher level language). Intel documentation on the 8085 is available from JDR Micro Devices.[1] The *MCS 80/85 Family Users Manual* covers the hardware and software aspects reasonably well. An 8085 assembler is necessary to convert the assembler code to machine code. There are a number of sources. I used an assembler from Pseudocorp.[2] It is available as freeware, or a commercial version is available for $50. Another assembler is TASM.[3] It is also available as freeware and a commercial version with tables for 10 different microprocessors is available for $40. (Note: TASM is available on the ARRL BBS, 203 666-0578.) Note that different assemblers may have slightly different syntax rules requiring minor changes in source code formatting.

Acknowledgments

I was able to do this project only with the support of others. Dr. James Humphries of Santa Fe Community College agreed to allow me to audit the College's Microprocessor course. It was here that I learned the basics. Mr. Ron Tinckham, the instructor, was willing to spend some of his free time after the course to monitor and advise me as I developed my own system. I thank the State of Florida for having a Community College system where a person can go to learn more about subjects of interest. In a way, it was my Elmer for this project.

Notes

[1]JDR Micro Devices, 2233 Samaritan Dr, San Jose, CA 95124, tel. 800 538-5000.
[2]Pseudocorp, 716 Thimble Shoals Blvd, Newport News, VA 23606, tel. 804 873-1947.
[3]Speech Technology, Inc, 837 Front St S, Issaquah, WA 98027, tel 206 392-8150.

The Growing Family of Federal Standards for HF Radio Automatic Link Establishment (ALE)

Part V: An Amateur's Practical Approach to HF ALE Radio Systems

ALE is becoming doable for amateurs. Here's one low-cost approach used now.

Paul C. Smith, K3ZMO and Dennis Bodson, W4PWF

Introduction

If you have been following this series of articles on Automatic Link Establishment (ALE) technology and those by Adair, Wickwire, and others, you have probably been convinced that this new technology is worth looking at for use by amateurs, but if you are like most hams, you will reserve judgment until you have had the opportunity to get some "hands on" experience with real equipment. Magazine articles can go just so far. Alas, here is the problem. These systems were designed with the requirements of the Federal Government in mind, emergency preparedness agencies and the military services in particular. We all know too well what that means: sticker prices in the $10,000 to $20,000 range which places those technologies out of reach for years, until they begin to appear on the surplus market. There is at least one manufacturer that produces an ALE system for less than $10,000, but that price is still well beyond what most hams could comfortably afford.

The engineers at the National Telecommunications and Information Administration/Institute for Telecommunication Sciences (NTIA/ITS) in Boulder, Colorado, have long been involved in the development of a family of telecommunication standards for federal agencies describing adaptive HF radio systems. Several years ago, ITS developed a need for inexpensive ALE radio systems to support the early testing and evaluation of new concepts. ITS is a very small government agency that receives the majority of its annual operating funds indirectly from other federal agencies, rather than in the form of direct appropriations from Congress. As funds for the purchase of equipment, such as ALE radios, are in a constant state of short supply, some degree of ingenuity was needed to acquire the needed ALE equipment. Hams can now benefit from that ingenuity.

The remainder of this article describes the approach that ITS took to home-brew an ALE radio system using off-the-shelf components. The total cost of this project was under $4,000, and it can be done for even less now. This approach should be of some interest to those hams who may want to experiment with this new technology without spending the family's fortune to do so. This might make a good club project, too.

System Specifications

An HF ALE radio system is nothing more than an HF SSB transceiver and an associated modem that supports the unique FED-STD-1045 (and MIL-STD-188-141A) protocols. The ALE modem is both a modem in the conventional sense and a controller for the transceiver, causing the radio to scan, transmit, and receive on command. Except for the robust protocols employed by the ALE modem (eg, deep interleaving, triply redundant word transmission and Golay encoding for error detection and correction), it can be considered to be very similar to the AX.25 packet modems (TNCs) familiar to most hams. It seemed logical to ITS that any modern low-cost transceiver developed for the amateur market and capable of

Paul C. Smith, K3ZMO
12328 Jasmine St
Brighton, CO 80601

Dennis Bodson, W4PWF
233 N Columbus St
Arlington, VA 22203

being controlled by a computer would work as part of a home-brewed ALE system. ITS selected the ICOM IC-725 because we already possessed an ICOM IC-781 transceiver and were familiar with the programming and commands of this series of radios.[1] Since the government operates on frequencies which are, in some cases, far from the ham bands, it was necessary to convert the IC-725 to general coverage on both transmit and receive. This was easily accomplished following instructions obtained from ICOM America, Inc, in Bellevue, Washington. To control the transceiver by computer, it was also necessary to obtain one additional piece of equipment: the ICOM communications interface, CI-V (model CT-17). The CT-17 is a signal-level converter that connects between the transceiver and a computer (or terminal) with an EIA RS-232C communications port.

The final item needed in our build-it-yourself ALE project was the ALE modem. At the time there was only one source for a stand-alone ALE modem: the Model 1045 ALE controller from Frederick Electronics of Frederick, Maryland (with a 1991 price of $3,000). As far as ITS is aware, this is still the case, although we understand that Space Research Technologies, Inc, intends to produce an ALE controller on a card for insertion into a PC, to be sold to the amateur market.[2] The card price is projected to be somewhere in the $700-$800 range and to be available in late 1993.

The Frederick ALE modem is designed to interface with several different radios, as well as external high-speed data modems and terminal devices. Table 1 lists the transceivers with which the Model 1045 has been interfaced. ITS provided Frederick with the ICOM radio command set. These commands were incorporated into the Model 1045 firmware with the radio hex address that was selected (to match those of our other ICOM products).

Interfacing a stand-alone ALE modem/controller to an SSB HF transceiver is no more difficult than interfacing a packet terminal node controller (TNC). One simply provides paths between the two devices for control signals and the FSK audio tones to and from the radio. However, the impedances of the inputs and outputs of the various devices have to be considered.

Upon receiving the new Frederick modem it was discovered that the input and output impedances for the FSK tones are 600 Ω balanced, while the ICOM transceiver requires impedances of 10 kΩ unbalanced for input to the radio and 4.7 kΩ unbalanced from the radio. A quick test revealed that the system could correctly receive the ALE tones from another station but could not transmit them successfully without providing a closer impedance match; it was necessary to provide impedance matching between the radio and the ALE modem/controller. A quick look at a recent Radio Shack catalog provided the answer. Radio Shack offered a microphone transformer designed to match an unbalanced high impedance (nominally 10 kΩ) to a 600-Ω balanced line impedance. While not the perfect solution, these transformers provided adequate signal levels for the ALE tones. Fig 1 shows the prototype ITS low-cost ALE system components. Fig 2 is a block diagram of this system showing the major I/O ports on the ALE modem. Fig 3 is a schematic of the "field-expedient" audio impedance transformation unit.

Equipment Setup

The final step was to interface a video display terminal or personal computer running a communications program that permits the emulation of a Digital Equipment Corp (DEC) VT 100 or VT 220 terminal. ITS chose to use an available laptop computer with *ProComm Plus* communications software. Once the system is assembled and interconnected as shown in Fig 2, power is applied to the radio (via an external power supply) and the system with *Procomm Plus* booted up on the computer. At this point, the computer-to-ALE modem parameters are set up. These are presented in Table 2.

TABLE 1

Transceivers Currently Interfaced to Frederick Model 1045 ALE Modems

Transceiver	*Models*
TransWorld Comm:	TW/RT-100
MacKay Comm:	MSR-8050A
ICOM:	IC-781,725,etc
Signal One:	MilSpec 1030C
Ten-Tec:	585 Paragon
Rohde & Schwarz:	XK-852C1/C2
Skanti:	TRP 82XX
SGC:	SG-2000
Hagenuk:	RX 1001 M/L11
	EX 1010 /L11

[1]Notes appear on page 12.

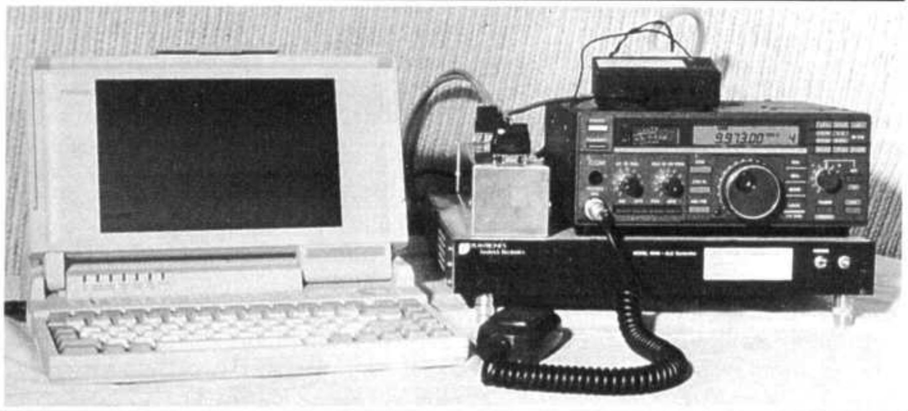


Fig 1—Low-cost HF ALE radio system.

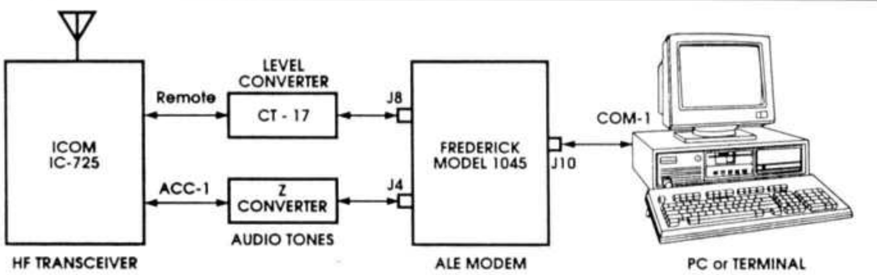


Fig 2—Low-cost ALE system block diagram.

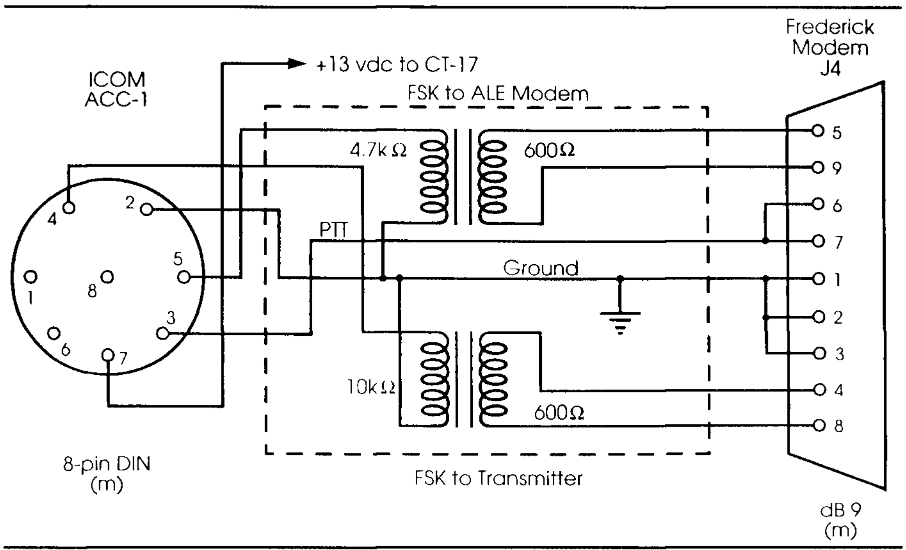


Fig 3—Audio impedance transformation interface box.

Table 2

Terminal Communications Parameters

Display Mode = VT 220 (VT 100)	
Data Rate = 9600 Baud	
8 data bits, 1 stop bit	
No parity, Full duplex	

Having set the terminal communications parameters, we are at the point where power may be applied to the Frederick modem. As the power is applied, you will see the Frederick header screen with its credits and the software version number (currently 2.20). After a few seconds, the screen will change to the working screen with its menus. This screen should look like the one shown in Fig 4. The bottom line, in reversed video, is the *Procomm Plus* status line. Everything else above the status line belongs to the ALE system. The ALE modem has its own status line at the very top of the screen. In the example shown in Fig 4, we see that our ALE system is scanning the frequencies of channel group #1, at a rate of 2 channels per second and listening for a call. We are currently on channel #4 with a transceiver frequency of 10.224 MHz. The indicated time is 08:01:45 (may be either UTC or local, as defined by the operator). The volume/mute indicator has no meaning for our system as this function is not used.

The line of ALE commands just above the *Procomm Plus* status line is the point of operator-machine interface with the ALE system. The example screen in Fig 4 shows the cursor on the "call" function key. After first accessing the system, you must program your station variables into the ALE software. These values include station call sign(s), called self addresses in the ALE vernacular, a set or sets of frequencies to be scanned, the sounding interval, and numerous other technical system parameters. For basic simplex operation, it is only necessary to load a single self address (call sign), the individual addresses (calls) of stations that you might wish to contact, and a set of frequencies that have been mutually agreed upon in the appropriate licensed bands.

Summary

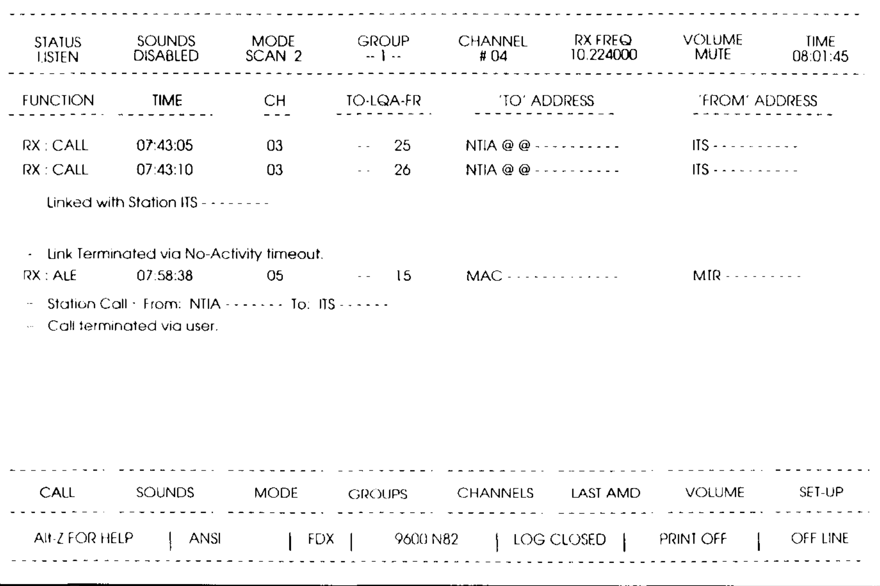
This low-cost ALE radio system has been a part of ITS's HF radio Interoperability Test Facility since December 1990 and has been used in numerous ALE performance and interoperability tests since that time. Laboratory personnel find the system easy to learn and very user friendly. The screen of the computer permits much more information to be displayed than does the LCD displays of most ALE equipment.

We recommend this approach to individuals or groups desiring to get into ALE for the lowest possible cost. Amateurs involved in traffic-handling for the NTS or MARS should find the new ALE technology particularly useful.

As expounded upon in previous articles, ALE radios operate quite successfully for linking and slow-speed data traffic at S/N ratios of 0 to 6 dB. These levels are unusable for voice and other modes having no error correction. This means that previously unusable noisy portions of the ham bands are now available for passing data traffic using ALE radios. And they *do not* interfere with existing modes of HF communications. Once again—new technology is exciting and opens a whole new realm of possibilities.

Acknowledgments

This work was supported by the National Communications System (NCS) and the National Telecommunications and Information Administration/



```
----------------------------------------------------------------------------------------
 STATUS     SOUNDS      MODE       GROUP      CHANNEL    RX FREQ     VOLUME      TIME
 LISTEN     DISABLED    SCAN 2     -- 1 --    # 04       10.224000   MUTE      08:01:45
----------------------------------------------------------------------------------------
 FUNCTION   TIME         CH      TO-LQA-FR      'TO' ADDRESS        'FROM' ADDRESS
 --------   --------     ---     ---------      ------------        --------------

 RX : CALL  07:43:05     03       --   25       NTIA @ @----------  ITS----------
 RX : CALL  07:43:10     03       --   26       NTIA @ @----------  ITS----------
    Linked with Station ITS--------

 - Link Terminated via No-Activity timeout.
 RX : ALE   07:58:38     05       --   15       MAC-------------    MTR---------
 -- Station Call - From: NTIA------- To: ITS------
 -- Call terminated via user.



---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
  CALL      SOUNDS      MODE      GROUPS     CHANNELS   LAST AMD    VOLUME     SET-UP
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
 Alt-Z FOR HELP | ANSI     | FDX |   9600 N82  | LOG CLOSED |  PRINT OFF  |  OFF LINE
----------------------------------------------------------------------------------------
```

Fig 4—Sample computer terminal display of ALE data.

Institute for Telecommunication Sciences (NTIA/ITS).

Bibliography

Federal Standard 1045, *Telecommunications: HF Radio Automatic Link Establishment*, General Services Administration, Washington, DC, January 24, 1990.

Horzepa, S., "ALE: A Cure for What Ails HF Communications?" Packet Perspective, *QST*, November 1992, p 107.

Wickwire, K., "The Status and Future of High Frequency Digital Communication, Part I: Overview," *QEX*, June 1992, pp 3-14.

Wickwire, K., "The Status and Future of High Frequency Digital Communication, Part II: HF Modems and Their Performance," *QEX*, July 1992, pp 3-15.

Wickwire, K., The Status and Future of High Frequency Digital Communication, Part III: Simulating the Performance of HF Digital Networks," *QEX*, August 1992, pp 12-17.

Wickwire, K., "The Status and Future of High Frequency Digital Communication, Part IV: Where is HF Digital Networking and Where is it Going?," *QEX*, October 1992, pp 10-18.

Adair, R.T. and Peach, D.F., "A Federal Standard for HF Automatic Link Establishment," *QEX*, January 1990, pp 3-7.

Adair, R.T. and Bodson, D., "A Family of Federal Standards for HF ALE Radios, *QST*, May 1992, pp 73-76.

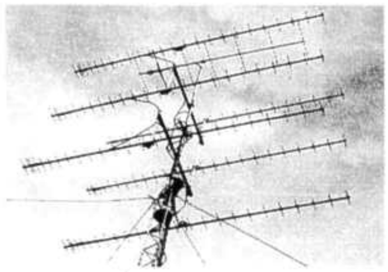
Smith, P.C., Wortendyke, D.R., Redding, C., and Ingram, W. J., "Interoperability Testing of FED-STD-1045 HF Radios," RF Expo West, Santa Clara, CA, February 5-7, 1991, pp 119-126.

Notes

[1]Certain commercial equipment and software products are identified in this paper to adequately describe the design of the experiment. In no case does such identification imply recommendation or endorsement by NTIA, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

[2]Space Research Technologies, Inc, Attn: Mr Laurence Rennie, 31255 Cedar Valley Dr, Westlake Village, CA 91362, phone (818) 991-0693. □□

RUTLAND ARRAYS
FOR THE ULTIMATE PERFORMANCE
VHF/UHF ANTENNAS & ACCESSORIES



EME:ATV:OSCAR:TROPO:FM:PACKET

MODEL	Freq.	# Ele	Length	Meas. Gain	Cost
RA4-50	50-51MHz	4el	12.3 ft	WE MEASURE GAIN USING ONLY NBS AND EIA STANDARDS. In side by side measurements our antennas may outperform other manufacturers products having advertized gain several dB more than ours.	139.95
RA7-50	50-51MHz	7el	26.5ft		279.95
RA8-2UWB	144-148MHz	8el	11.8ft		91.75
FO12-144	144-146.5MHz	12el	17.3ft		142.50
FO12-147	145-148MHz	12el	17.3ft		142.50
FO15-144	144-145MHz	15el	25.1ft		192.50
FO16-222	222-225MHz	16el	17.3ft		129.95
FO22-432	432-438MHz	22el	14ft		114.95
FO22-ATV	420-450MHz	22el	14ft		114.95
FO25-432	432-438MHz	25el	17.1ft		134.95
FO33-432	432-438MHz	33el	24.3ft		223.95
FO11-440	440-450MHz	11el	6ft		69.95

ALSO AVAILABLE
POWER DIVIDERS-STACKING FRAMES
CALL OR WRITE FOR OUR NEW CATALOG

WE USE ONLY 6061-T6 ALUMINUM OF U.S. MANUFACTURE

RUTLAND ARRAYS
1703 WARREN ST * NEW CUMBERLAND PA 17070
Orders 1-800-536-3268 Info. 1-717-774-3570 7pm-10pm EST
DEALER INQUIRIES ARE INVITED
WE DESIGN AND BUILD ANTENNAS FOR PERFORMANCE NOT PRICE!

"CAPMAN"
SKYWAVE ANALYSIS
"Use IONCAP+ like a pro!"
YOU GET

CAPMAN - Menu driven input & control with context-sensitive help at a mouse click!
Multicolor-output Graphics!
Extra or Novice, its EASY to use!

IONCAP - a 32 bit tool with the same features the pros have been using to get S/N, MUF, FOT, LUF, Reliability, & More!
Long-path model for DXers!

MININEC & ELNEC interface for your custom antenna input!

INCLUDE your contacts & friends in the contact library of 494 prefixes!

Entire CAPMAN package -- $89
USA postage paid. Overseas add $3.50

LUCAS Radio/Kangaroo Tabor Software
2900 Valmont Rd, Suite H
Boulder, CO 80301
(303) 494-4647 (494-0937 fax)
VISA MC CHECK MONEY ORDER

Simple and Inexpensive PC Interfacing

The PC printer port provides a useful and easy interface with the circuits described here.

Gary C. Sutcliffe, W9XT

As PC users upgrade to faster 386 and 486 computers their old machines are often gathering dust in closets. Who wants to use an old 4.77-MHz 8088-based PC with minimal memory and a single floppy drive? You can barely give them away at hamfests! Yet, such computers make great platforms for dedicated controllers for equipment in the ham shack or other electronic projects.

The bad news is that you are likely to have to pay a lot more for an interface board to connect to the outside world than the computer itself is worth. Fortunately, the computer probably already has a built-in interface that is often overlooked. It is the line printer interface port, commonly known as the LPT port.

The LPT port has 12 bits of output and 5 bits of input used for communicating with a printer. Ham radio contesting software, such as K8CC's *NA* and K1EA's *CT*, uses LPT ports for controlling voice keyers and sending CW. Some PC-based PROM programmers also make use of LPT ports. With a little imagination, you can use the port to control your project. This article explains the LPT port, shows ways of using the LPT I/O (input/output) lines, and discusses programming the LPT port in BASIC and C. Finally, I've provided a design and sample software for a generic interface suitable for bench-top experimentation.

3310 Bonnie Lane
Slinger, WI 53086
Email: ppvvpp@mixcom.com (Internet)

The Hardware Side of the LPT

Table 1 lists the signals found at the DB-25 connector of a standard LPT port. The signals are all TTL voltage levels, but the drive capability of the outputs will vary from computer to computer. Portable and laptop computers frequently have less drive capability than desk-top units due to their power conserving designs. Drive capability comes into play when long cables, with their associated large stray capacitance, are used, so this may be a consideration if the device is some distance from the computer.

When the port is connected to the printer, each of the signals has a defined purpose. But when you are controlling something else, you can define the signals to represent whatever you want as long as you keep the inputs and outputs straight. There are a couple of things to keep in mind when using a few of the signals; they will be discussed later when the software side of the LPT port is covered.

Table 1 - LPT Port I/O Pins

Outputs	
Pin 1:	$\overline{\text{STROBE}}$
Pins 2-9:	Data bits 0-7
Pin 14:	$\overline{\text{AUTO LF}}$
Pin 16:	$\overline{\text{INIT}}$
Pin 17:	$\overline{\text{SLCT IN}}$
Inputs	
Pin 10:	$\overline{\text{ACK}}$
Pin 11:	BUSY
Pin 12:	PE
Pin 13:	SLCT
Pin 15:	$\overline{\text{ERROR}}$
Grounds	
Pins 18-25	

Output Interfacing

If the device to be controlled has TTL-level inputs and outputs, it can usually be connected directly to the LPT port. But many applications will require more drive power than the port can supply, or switching voltages different from 5-V TTL levels. In some applications you will want to provide isolation to protect the computer. Fig 1 shows some simple interface techniques to accomplish this.

Fig 1A is an example using a transistor to act as a switch for turning on a load. The voltage and load currents can be any reasonable values the transistor can safely handle. The diode in the in-

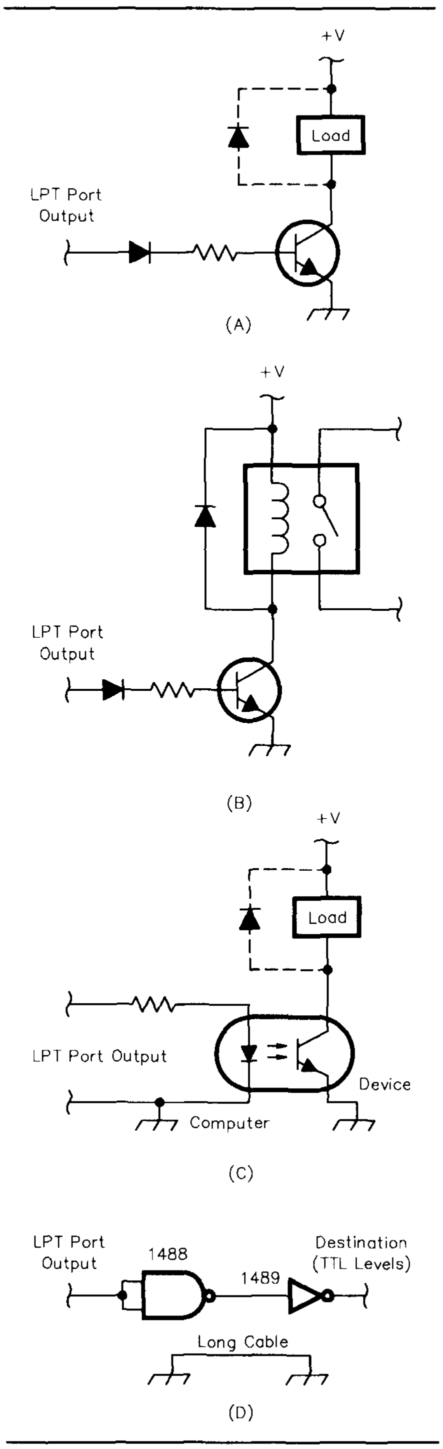


Fig 1-Output interface circuits.

put connector may not be needed in some cases, but it ensures that the transistor will turn off when the output line goes low—to a few tenths of a volt. If the load is inductive (such as a solenoid or a relay) the diode across the load is needed to protect the transistor from currents generated when the transistor is switched off and the magnetic field of the coil collapses. Fig 1B is just a special case of Fig 1A, using a relay. The relay provides better isolation than just a transistor.

Isolation can be provided by use of an optoisolator, as shown in Fig 1C. In this case, light from the LED turns on the transistor built into the optoisolator. Inexpensive optoisolators can provide isolation of a thousand volts or more. Optoisolators are also available with TRIAC outputs that let you switch ac voltages.

LPT ports were designed to drive a printer located near the PC. The maximum recommended cable length is often about 10 feet. The device you want to control may be located much further away. In this case, some sort of driver IC may be in order. Fig 1D shows an LPT port output signal using an RS-232 driver to drive a long cable. The RS-232 receiver at the far end has a TTL output. You could use this to drive one of the circuits in Figs 1A or 1B to provide additional drive capability.

If the output changes state infrequently (say less than a few hundred Hertz), this circuit should work fine for cable lengths of several hundred feet or more. Keep in mind that the circuit in Fig 1D only uses RS-232 voltages (±12V) and does not directly support RS-232 serial data transmissions (although it could do that with proper software).

Using the Inputs

Monitoring external signals is not difficult. If the external device has TTL-compatible outputs, they normally can be directly connected to the LPT input pins. Switches will need a pull-up resistor. Fig 2A shows a simple method of detecting switch closures. The LPT input pin will sense a logic 1 when the switch is open, and a logic 0 when it is closed.

As with outputs, there will be times when you want to provide additional isolation between the external device and the computer. The optoisolator again proves a good way to do this. Fig 2B shows one way. The resistor, R, is chosen to limit the LED current to acceptable limits. Keep in mind that Fig 2B incurs a logical inversion. That means if the signal driving the diode in the optoisolator is high (a logical 1) the input pin will detect a logical zero. You could put an inverter between the optoisolator and the input pin, but it is cheaper and easier to account for the inversion in your software.

If you need to monitor a signal from a distance, you can use a circuit like Fig 1D to use RS-232 signaling levels. In this case, have your remote switch (or whatever) drive the 1488 and connect the 1489 receiver output to the LPT port input pin.

The Software View of the LPT

The PC talks to the LPT port (and other I/O devices, for that matter) through registers. Registers are 8- or 16-bit locations in the peripheral device similar to bytes in main memory. The 80x86 microprocessor family has a range of I/O locations separate from main memory and uses special instructions to act upon these I/O registers. LPT ports have 8-bit registers, so unless noted otherwise, from now on registers will be assumed to be 8 bits.

Every I/O device has what is known as a base address. The base address is the address of the first register of the device. Each additional register has an address one higher (or two for 16-bit registers) than the previous one.

For older PC and XT class machines (especially those with monochrome graphics adapters), the LPT port often has a base address of 3BCh (the h means the address is in hexadecimal, or hex for short). AT-class machines usually have LPT1 at 378h, with LPT2 at 278h.

You can find out how many LPT ports your PC has, and what their base addresses are, by running the program "FINDLPT.BAS" found in Listing 1. This program is written in BASIC, and works with Microsoft GWBASIC and QBASIC. It may need some changes for use with other BASIC dialects. When you run the program, FINDLPT will check the BIOS data area and print out

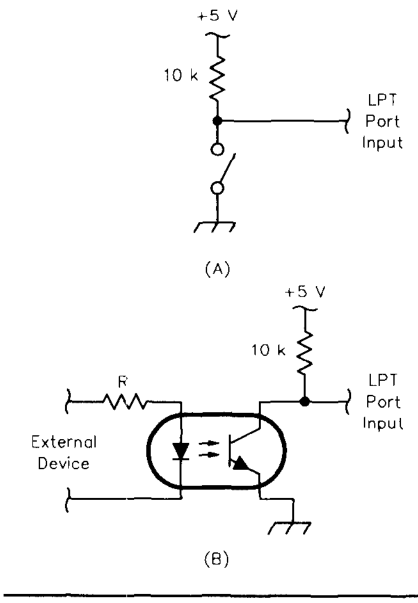


Fig 2-Input interface circuits.

Table 2 - LPT Register Definitions

Data Register - Base Address

Bit 0:	Data bit 0	- Output, pin 2
Bit 1:	Data bit 1	- Output, pin 3
Bit 2:	Data bit 2	- Output, pin 4
Bit 3:	Data bit 3	- Output, pin 5
Bit 4:	Data bit 4	- Output, pin 6
Bit 5:	Data bit 5	- Output, pin 7
Bit 6:	Data bit 6	- Output, pin 8
Bit 7:	Data bit 7	- Output, pin 9

Status Register - Base Address + 1

Bit 0:	Undefined	
Bit 1:	Undefined	
Bit 2:	Undefined	
Bit 3:	$\overline{\text{ERROR}}$	- Input, Pin 15
Bit 4:	SLCT	- Input, Pin 13
Bit 5:	PE	- Input, Pin 12
Bit 6:	$\overline{\text{ACK}}$	- Input, Pin 10
Bit 7:	BUSY	- Input, Pin 11

Control Register - Base Address + 2

Bit 0:	$\overline{\text{STROBE}}$	- Output, Pin 1
Bit 1:	$\overline{\text{AUTO LF}}$	- Output, Pin 14
Bit 2:	INIT	- Output, Pin 16
Bit 3:	$\overline{\text{SCLT IN}}$	- Output, Pin 17
Bit 4:	Interrupt Enable, Internal	
Bit 5:	Undefined	
Bit 6:	Undefined	
Bit 7:	Undefined	

the base addresses of the LPT ports it has found.

The LPT port has 3 registers, as shown in Table 2. The table lists the standard names for each of the defined bits, but a detailed description of their purpose will not be given here since you can use them for pretty much anything you want to for your project.

The first register (found at the base address) is the data register. This is simply the 8 data bits that normally output the ASCII character to be printed. The bit pattern you write to this register will appear at output pins 2-9 and remain there until you write something else to the register. If you read the data register, you will get back the last byte you wrote to it.

The next register is the status register. Its address is the base address + 1. The bits in this register reflect the levels on the input pins. The status register is read-only, and is normally used for keeping track of the printer's condition. Normally, the software will check these bits and report situations like the printer being off-line or out of paper. Writing to this register has no effect.

Bits 0-2 of the status register are not defined. They generally read as all 0's or 1's depending on the make of the computer. Any software you write should not depend on them being in a particular state.

Two bits in the status register deserve special mention. First, bit 7 (BUSY, from pin 11) is inverted. This means that if the signal connected to pin 11 is high (a logical 1), it will be a zero when you read the status register. The other special bit is bit 6, the ACK signal from pin 10. This signal will cause an interrupt if the interrupt enable bit (to be described shortly) is set.

The final register is the control register, which has an address equal to the base address + 2. This register is normally used to tell the printer to reset, or to accept a new character to print, etc. Bits 0-3 appear at the LPT port output connector. Note that bits 0, 1, and 3 are inverted. If you write a 1 to one of the inverted bits, the output pin will be low, and vice versa. Bit 4 is the *interrupt enable* bit. If you write a 1 to this bit, and if pin 10 is low, it will generate an interrupt to the PC. Reading the control register will return what you have previously written.

Many PCs implement the control register bi-directionally, using TTL drivers with open collector outputs. You could use these bits for both inputs and outputs, but I have come across at least one PC clone that only implemented the outputs. For compatibility reasons I only use the control register for outputs.

Writing Software

Once you have connected the LPT port to your external device, either directly or through additional circuitry similar to that found in Figs 1 and 2, you will have to write software to run it all.

There are two ways to write software to control hardware. The first—and normally the best—way is through the operating system. With PCs this is usually accomplished through BIOS (Basic Input/Output System) calls or loadable device drivers. The BIOS contains special low-level software routines found in ROM that control and monitor the hardware. BIOS routines are accessed through software interrupts. Device drivers are loaded into RAM for use by DOS and application programs. Video and mouse drivers are common examples of these.

BIOS calls and device drivers are fine if you are going to be using the PC's hardware (disks, printer and serial ports) in the manner it was normally intended to be used. If you are doing something different, say controlling a repeater or antenna switching, these functions are not likely to do precisely what you want them to do. For example, when you want to print a character, the proper BIOS call will put the ASCII character in the data register. It then writes a logical 1 to bit 0 ($\overline{\text{STROBE}}$) of the control register to tell the printer to accept the new character, and finally, clears the $\overline{\text{STROBE}}$ bit after a few microseconds. But you may want to use the $\overline{\text{STROBE}}$ bit in a different way, making the use of BIOS calls impractical.

Device drivers could be written to control your antenna switching system, but writing them is an involved procedure and is of no particular benefit to a stand-alone control program.

The other way to use the LPT port is to directly access the registers in question. The advantage of this method is that it allows you to control the register bits exactly as you wish. The danger is that you can end up with problems if you accidentally cause an interrupt or switch between using the LPT port for controlling your device and printing without rebooting first. But you should not have any problems if you use care.

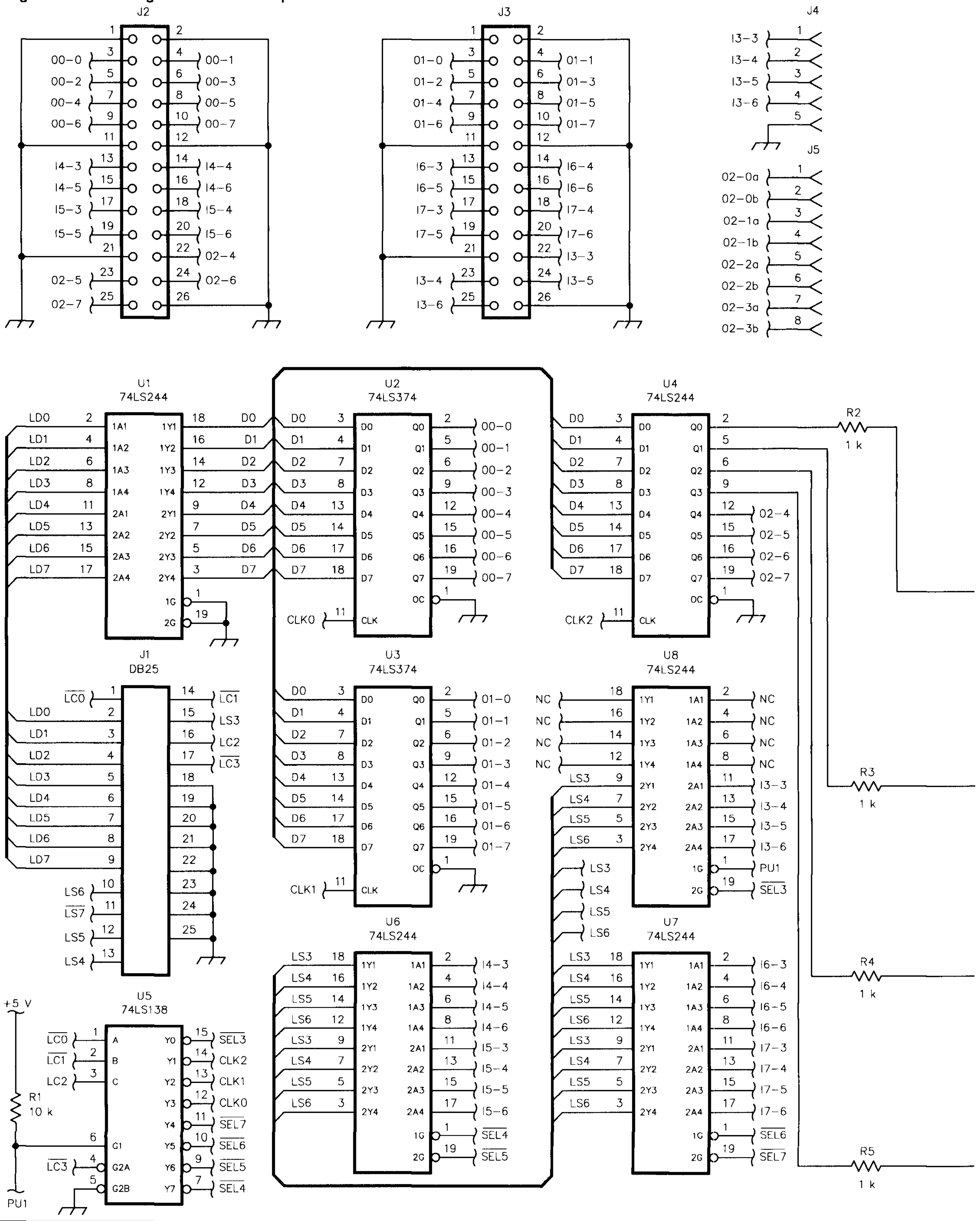
This article covers programming the hardware registers. Using interrupts will not be discussed. Interrupt-driven software is more complex. The books mentioned in the reference section are a good place to start if you are interested in digging into device drivers and interrupts.

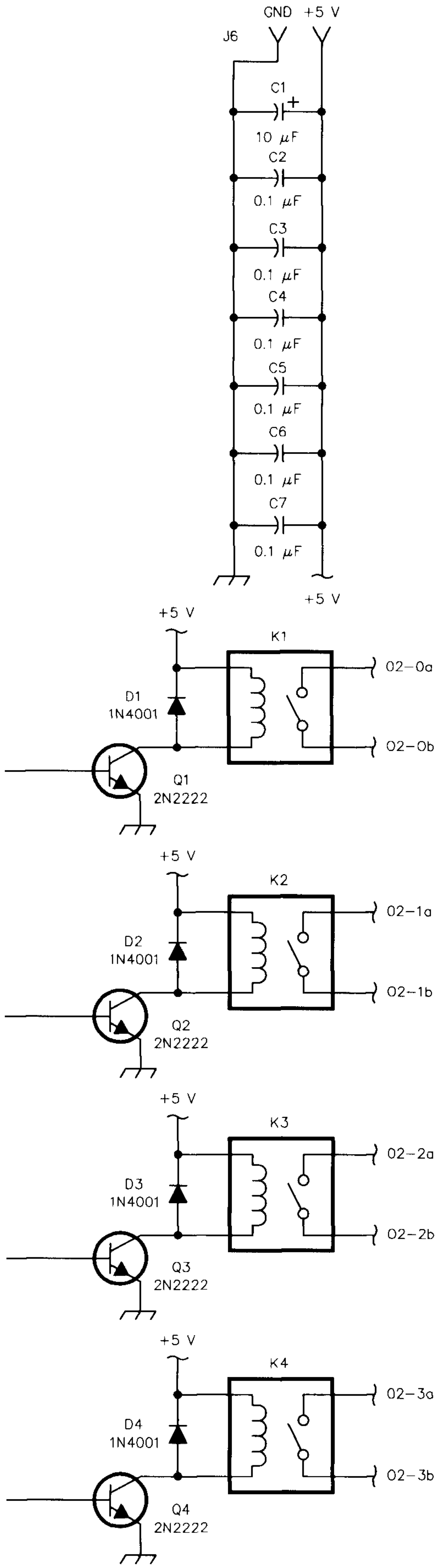
BASIC and C have functions for reading and writing directly to the registers of boards plugged into the expansion slots. These may be unfamiliar to programmers who normally write application programs that do not directly deal with the hardware.

BASIC uses the INP and OUT functions to read and write bytes to and from the I/O registers. These are similar to PEEK and POKE, but deal with I/O registers as opposed to main memory. BASIC is not the best language for controlling hardware since it is cumbersome in its handling of individual bits. It is discussed here because nearly every PC has BASIC and nearly every programmer has experience with it.

Standard C does not include functions to communicate with PC I/O addresses since the language was designed to be machine independent. Software companies who write compilers include their own extensions to handle machine-specific operations. Unfortunately, there is no standard for I/O functions in the various dialects of C for the PC, so consult your compiler manual to find what your compiler uses. The C examples included in this

Fig 3-Schematic diagram of the LPT Expander circuit.





C1	10 µF, 16 V min
C2-C7	0.1 µF, 16 V min, 20%
D1-D4	1N4001
J1	DB-25 right angle connector
J2,J3	26-pin right angle PC-mount ribbon cable connector
J4	5-position screw terminal block
J5	8-position screw terminal block
K1-K4	5-V SPST DIP relay, EAC D1A05H or similar. Note some DIP relays may have different pin outs.
R1	10 k, 5%, 1/4 watt
R2-R5	1 k, 5%, 1/4 watt
Q1-Q4	2N2222 or similar
U1,U6-U8	74LS244
U2-U4	74LS374
U5	74LS138

Fig 4—LPT Expander parts list.

article use Borland Turbo C which uses the inportb() and outportb() functions for reading and writing bytes to and from I/O ports.

Using the I/O functions to communicate with an LPT port is pretty straight forward once you know the base address of your port. You just use the correct functions to read bits from the status register and to write bits to the data and control registers. Here are a few tips to keep you out of trouble:

1) Never allow a 1 to be written to bit 4 of the control register unless you have included software to handle interrupts.
2) Keep in mind that the input bits you read from the status register appear in locations 3-7, not starting at bit 0.
3) Never depend on an undefined bit to be in a certain state. Likewise, it is best to choose comparison strategies that can't be affected by any bits other than the particular bits of interest.
4) Remember that switch contacts bounce and can create a series of 1's and 0's when the switch is toggled. It is good practice to put a 50-millisecond delay in your software after you detect a switch change to handle de-bouncing.
5) Your computer operates in the world of microseconds or faster. Relays and other electo-mechanical devices operate in the world of tens or hundreds of milliseconds. Don't set a bit to turn on a relay and expect it to be closed by the next computer instruction!
6) Be sure you keep track of which inputs and outputs are inverted between the registers and pins.

The LPT Expander

I have used LPT ports to interface PCs to the outside world on several occasions in both my electronic consulting business and for playing around with stuff at home and in the shack. I thought it would be nice to have a little generic interface that would make it easy to play with controlling things on the bench without having to go through the trouble of building new interface circuits every time. I also wanted to increase the number of inputs and outputs from the number available on the LPT. These considerations gave rise to the LPT Expander, shown in Fig 3.

The LPT port connects to J1 through a 25-pin cable. The signals from the port have been renamed to make it easy to know where they come from. All the signals that are from the LPT data register start with LD. Control register signals start with LC, and status register signals start with LS. The number following the signal name represents the bit in the corresponding register. If a signal takes the form of $\overline{LXn}$, it means there is a logical inversion between the register in the LPT port and the connector pin.

The signals from the data register go through U1, which acts as a buffer, and drive U2-U4. These are 8-bit registers that latch the data, expanding the outputs from 8 to 24 bits. The outputs of these ICs are labeled O*n*-*m*, where *n* is the output port number (0-2) and *m* is the corresponding bit number in the data register. Bits 0-3 in U4 (port 2) drive relays to provide isolated switch closures. The other outputs go to connectors as TTL-level signals.

The input signals that ultimately find their way to the status register go through ICs U6-U8. These are three-state drivers that allow the selection of 20 input signals, 4 bits at a time. The input signals from the connectors are numbered I*n*-*m*, where again *n* is the port number (3-7), and *m* is the corresponding bit in the status register.

The control register bits all go to U5, which acts as a decoder to select which set of outputs or inputs to use. Bits 0-2 form a binary number to select the desired output port (0-2) or input port (3-7).

Note that bits 0 and 1 from the control register are inverted but bit 2 is not. This means that if you write a 000 bit pattern in the control register, it will appear at the input pins of U5 as 011. Rather than trying to remember to keep this in mind every time you want to write the control register bits, the outputs of U5 are wired so that when you write the number 0 to bits 0-2 you get CLK0 (port 0), when you write the number 5, you get SEL5 (port 5), etc.

A problem can occur when using a simple decoder for generating clocks for latches as is done for U2-U4: switching glitches can cause the wrong output latch to clock. This is prevented by using control register bit 3 to enable the outputs of U5. This makes the software to drive the LPT Expander a bit more complex, but an extra instruction or two is less expensive than more complex hardware.

The output connectors take two forms on the unit I built. J4 and J5 are screw-type terminal blocks. They have the four relay outputs and four TTL-level inputs. This type of connector is used to make it easy to hook up simple circuits. J2 and J3 have more TTL inputs and outputs. J2 has 12 outputs and 8 inputs, while J3 has 8 outputs and 12 inputs. These are used where more complex interfacing is needed, and the extra trouble of making a ribbon cable is worthwhile.

A couple of months after I build something like the LPT Expander I usually forget exactly how to use it. This project was designed to be easy to figure out what is going on and use. Along with naming the I/O signals to give an immediate indication of the LPT port bit and register they are associated with, I wrote special software routines to make it easy to read and write to the outside world—without rein-

```
10 REM FINDLPT.BAS - THIS PROGRAM CHECKS THE BIOS DATA AREAS TO FIND WHAT
20 REM LPT PORTS DOS HAS FOUND.
30 REM GARY C. SUTCLIFFE, W9XT, JUNE 1993
40 DEF SEG = &H40    'DEFINE SEGMENT
50 PRINT
60 PRINT " DOS REPORTS LPT PORTS FOUND AT THE FOLLOWING I/O ADDRESSES: "
70 PRINT
80 FOR I = 1 TO 3
90 X = PEEK(I + I + 6)      'LOW BYTE OF I/O ADDRESS
100 Y = PEEK(I + I + 7)      'HIGH BYTE OF I/O ADDRESS
110 IF Y = 0 THEN 160    'IF Y =0, NO MORE LPTS
120 X$ = HEX$(X)         'CONVERT BYTES TO HEX STRING
130 Y$ = HEX$(Y)
140 PRINT "      LPT"; I, Y$; X$
150 NEXT I
160 PRINT
170 END
```

Listing 1 - Program to find LPT port addresses.

```
/**********************************************************************/
/*  CLPTEX.C  - This program contains C functions for controlling the   */
/*  LPT Expander.  The main() function only show examples of the        */
/*  functions that talk to the LPT Expander throught the LPT port.      */
/*                                                                      */
/*  This program was written with Borland's Turbo C.  Some changes      */
/*  may be needed if another compiler is used.                          */
/*  July 27, 1993 - Gary C. Sutcliffe W9XT                              */
/**********************************************************************/

#include <dos.h>

#define LPT1 0x378      /*base address for LPT1 - may need to be changed */
                        /* to 0x3BC on some XT class machines */
#define LPT2 0x278      /* May need to be changed on some PCs */

void OutBIF();
unsigned char InBIF();
unsigned char InBIFD();

void main()
{
unsigned char indat;   /* input data read from external device */
unsigned char outdat;  /* output data to write to externai device */

/*  The following are example calls to the functions shown later */

outdat = 0x05;           /*sample output data*/

OutBIF(LPT1,2,outdat);  /*write 0Fh pattern to LPT Expander port 2*/
                        /* This example turns on relays K1 & K3 on the Expander*/

indat = InBIF(LPT1,3);  /*read 4 bits of input from Expander port 3*/
                        /*Note bits are in location 3-6*/
printf("\n\nPort 3 shows data: %Xh",indat);

indat = indat >> 3;     /*This shifts them so the low bit is at bit 0*/
                        /* in case it is more useful to operate on them */
                        /* this way. */
printf("\nThis shows Port 3 data shifted down to bit 0: %Xh",indat);

indat = InBIFD(LPT1,4);  /*read the bits in Expander ports 4 & 5 and get */
                         /*them as a single 8 bit byte. */
printf("\n\nThis is combined data from ports 4 & 5: %2Xh\n\n",indat);


} /* end of main */

/**********************************************************************/
/* OutBIF() - This function is used to write a byte of data to one of the */
/* three output ports of the LPT Expander.                            */
/*                                                                    */
/* ARGUMENTS:  base - base address of the desired LP port             */
/*             port - port number (0-2)                               */
/*             data - data to write                                   */
/*                                                                    */
/* RETURNS: void                                                      */
/* CALLS: uses outportb(), found in <dos.h>                           */
/*--------------------------------------------------------------------*/

void OutBIF(int base, char port, unsigned char data)
{
int creg;                           /* control reg, used to select ports*/
creg = base + 2;                    /* set to proper address */

outportb(base,data);                /*put the data in the LPT DATA Reg*/
outportb(creg,port);                /*select the proper output port (0-2)*/
outportb(creg,port | 0x08);         /* now clock it  */
outportb(creg, port);               /*remove the clock */
}  /* end of OutBIF */
```

```
/**********************************************************************/
/* InBIF() - This function is used to read 4 bits of data from one of    */
/* the five input ports of the  LPT Expander.                            */
/*                                                                       */
/* ARGUMENTS:  base - base address of the desired LP port                */
/*             port - input port number (3-7)                            */
/*                                                                       */
/*                                                                       */
/* RETURNS:  data byte from selected input port                          */
/* CALLS: uses outportb() & inportb, found in <dos.h>                    */
/* NOTES:  Only bits 3-6 have data.  The other bits are zeros            */
/*-----------------------------------------------------------------------*/
unsigned char InBIF(int base, char port)
{
unsigned char indat;            /*input data byte*/
int creg;                       /* control reg, used to select ports*/
creg = base + 2;                /*set to proper address*/

outportb(creg,port);            /*select port*/
outportb(creg,port | 0x8);      /*enable the port*/
indat = inportb(base+1);        /* read the data */
outportb(creg,port);   /*deselect the port*/
indat = indat & 0x78;           /* blank unused bits (only bits 3-6 used)*/
return(indat);
}   /* end of InBIF */

/**********************************************************************/
/* InBIFD() - This function is used to read two 4 bit input ports and to */
/* combine them to form a single 8 bit byte of input data.              */
/*                                                                      */
/* ARGUMENTS:  base - base address of the desired LP port               */
/*             port - lowest port number (4 or 6, other values will     */
/*                    return invalid data)                              */
/*                                                                      */
/* RETURNS:  data byte from selected input ports                        */
/* CALLS: uses outportb() & inportb, found in <dos.h>                   */
/* NOTES: Only ports 4 or 6 should be used.  Zeros will be returned if  */
/*        another value is used.                                        */
/*        Ports 4 & 6 are the low nibble, 5 & 7 are high                */
/*----------------------------------------------------------------------*/
unsigned char InBIFD(int base, char port)
{
unsigned char indat;            /*input data byte*/
unsigned char tmp;              /* temporary data*/
int creg;                       /* control reg, used to select ports*/
creg = base + 2;                /*set to proper address*/


if(port != 4 && port != 6)return(0);  /*bug out if wrong port selected*/
outportb(creg,port);            /*select first port*/
outportb(creg,port | 0x8);      /*enable the port*/
tmp = inportb(base+1);          /* read the data */
outportb(creg,port);            /*deselect the port*/
tmp = tmp & 0x78;               /* blank unused bits (only bits 3-6 used)*/
indat = tmp >> 3;               /* save low nibble*/
port++;                         /* now use the next input port*/
outportb(creg,port);            /*select second port*/
outportb(creg,port | 0x8);      /*enable the port*/
tmp = inportb(base+1);          /* read the data */
outportb(creg,port);            /*deselect the port*/
tmp = tmp & 0x78;               /* blank unused bits (only bits 3-6 used)*/
tmp = tmp << 1;                 /* move high nibble to correct position*/
indat = indat | tmp;            /*combine the nibbles to form the byte*/
return(indat);
}  /* end of InBIFD */
```

Listing 2 - C program to control the LPT Expander.

```
10 REM  BLPTEX.BAS  - THIS PROGRAM GIVES SOME EXAMPLES ON CONTROLLING THE
20 REM  LPT EXPANDER IN BASIC.  THIS PROGRAM WAS WRITTEN WITH MICROSOFT
30 REM  GWBASIC.  SOME CHANGES MIGHT BE NEEDED IF ANOTHER BASIC
40 REM  INTERPRETER/COMPILER IS USED.
50 REM  JULY 1993 - GARY C. SUTCLIFFE W9XT
60 REM  ***********************************************************
70 B = 888  'BASE ADR, 888 = 378H (LPT1); USE 632 FOR 278H, 956 FOR 3BCH
80 P = 2   'BLPTIF PORT, 0-2 ARE OUTPUTS, 3-7 ARE INPUTS
90 REM - SAMPLE OPERATIONS
100  D = 15      'DATA TO WRITE
110  GOSUB 500    'WRITE DATA D TO PORT P ON LPT WITH BASE ADR B
120  P = 3        'SELECT INPUT PORT 3
130  GOSUB 600    'READ INPUT PORT 3 AND SAVE IN VARIABLE D
140  D$ = HEX$(D) 'CONVERT IT TO ITS HEX VALUE
150  PRINT "BLPTIF PORT ";P;"READS THE HEX VALUE ";D$    'PRINT IT OUT
160  P = 6        'CHANGE TO INPUT PORT 6
170  GOSUB 700    ' READ INPUT PORTS 6 & 7 AS A COMBINED 8 BIT VALUE
180  D$ = HEX$(D)
190  PRINT "BLPTIF PORTS ";P;" & ";P+1;" READ A COMBINED HEX VALUE OF ";D$
200  END      ' OF EXAMPLES
500 REM *********************************************************
510 REM  OUTPUT DATA D TO PORT P ON LPT WITH BASE ADDRESS B
515 OUT B,D           'SEND OUT DATA TO THE LPT DATA REG
520 OUT B+2,P         'SELECT THE PROPER LPT EXPANDER OUTPUT PORT
530 OUT B+2,P OR 8    'CLOCK THE DATA
540 OUT B+2,P         'REMOVE THE CLOCK
550 RETURN    'END OF SUBROUTINE TO WRITE TO LPT EXPANDER OUTPUT PORT
600  REM *·*******************************************************
610  REM INPUT DATA FROM PORT P OF LPT EXPANDER CONNECTED TO LPT AT BASE ADDR B
620  REM NOTE THAT ONLY BITS 3-6 ARE USED.  OTHER BITS ARE SET = 0
630 OUT B+2,P              'SELECT THE RIGHT INPUT PORT ON LPT EXPANDER
640 OUT B+2,P OR 8         'ENABLE SELECTED PORT
650 D = INP (B+1)          'GET THE DATA
660 OUT B+2,P              'DISABLE SELECTED INPUT PORT
665 D = D AND 120          'BLANK UNUSED BITS
670 RETURN    'END OF SUBROUTINE TO READ FROM LPT EXPANDER INPUT PORT
700 REM **********************************************************
710 REM INPUT 8 BITS FROM TWO CONSECUTIVE LPT EXPANDER PORTS (ONLY 4-5 OR
720 REM 6-7).
730 OUT B+2,P              'SELECT PORT OF LOW 4 BITs
740 OUT B+2,(P OR 8)       'ENABLE PORT
750 D = INP(B+1)           'GET LOW 4 BITS
760 OUT B+2,P              'DISABLE SELECTED INPUT PORT
770 D = D AND 120          'BLANK UNUSED BITS
780 D = D / 8              'SAME AS SHIFTING 3 BIT POSITIONS TO RIGHT
790 P = P + 1              'NOW USE NEXT HIGHER PORT
800 OUT B+2,P              'SELECT THE PORT WITH THE HIGH NIBBLE
810 OUT B+2, P OR 8        'ENABLE THAT PORT
820 T = INP(B+1)           'GET THE DATA AND SAVE IN T
830 OUT B+2,P              'DE-SELECT THE PORT
840 T = T AND 120          'BLANK UNUSED BITS
850 T = T * 2              'SHIFT 1 BIT POSITION TO LEFT
860 D = T OR D             'COMBINE UPPPER AND LOWER BITS
870 P = P - 1              'RETURN PORT NUMBER TO ORIGINAL STATE JUST IN CASE...
880 RETURN
```

Listing 3 - BASIC program to control the LPT Expander.

venting the wheel for each application.

The LPT Expander design shown in Fig 3 is a good starting point for experimenting with parallel computer interfacing. Use the design as a starting point for your own interface, but don't be afraid to make changes to the design for your needs. For example, you may want to change the numbers of inputs or outputs, or include some of the interfacing techniques shown in Fig 1 or 2. You may also want to use different connectors. Some of the parts used were designed in largely because they were already in my junk box, although all of them are readily available through mail-order sources.

Listings 2 and 3 include some basic routines that can be used with the LPT Expander. They show the basic operations needed to control it, but the biggest advantage is they can be reused for the next project without having to go back and look at the circuit to figure out what has to be done to control the thing. Listing 2 is in the C language, and Listing 3 is in BASIC.

Summary

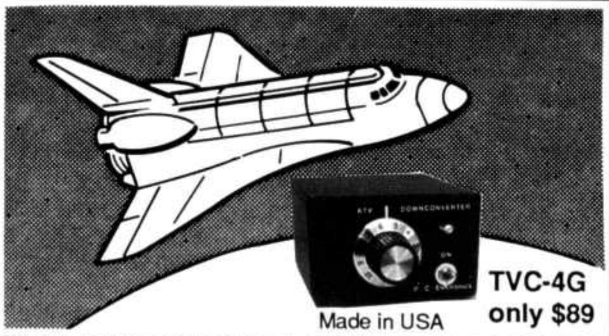
Using an old PC to control external devices is not all that difficult or expensive if you understand a bit about programming and interfacing techniques. Give that old clunker PC of yours a second life!

References

Jourdain, R., *Programmer's Problem Solver*, New York, Brady Publishing, 1992.

Lai, R.S., *Writing MS-DOS Device Drivers*, New York, Addison-Wesley, 1987. □□

AMATEUR TELEVISION



TVC-4G only $89

Made in USA

SEE THE SPACE SHUTTLE VIDEO

Many ATV repeaters and individuals are retransmitting Space Shuttle Video & Audio from their TVRO's tuned to Satcom F2-R transponder 13 or weather radar during significant storms, as well as home camcorder video. If it is being done in your area on 70 CM - check page 461 in the 93-94 ARRL Repeater Directory or call us, ATV repeaters are springing up all over - all you need is one of the TVC-4G ATV 420-450 MHz downconveters, add any TV set to ch 2, 3 or 4 and a 70 CM antenna. We also have downconverters, antennas, transmitters and amplifiers for the 400, 900 and 1200 MHz bands. In fact we are your one stop for all your ATV needs and info.

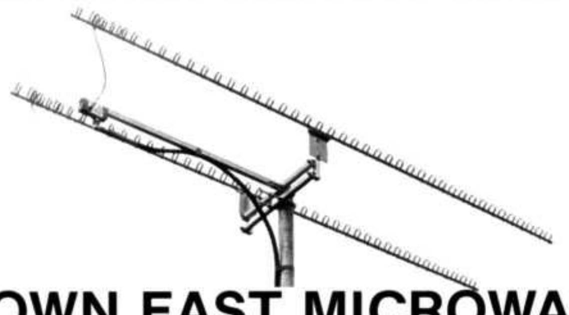
Hams, call now for our complete ATV catalogue!

We ship most items within 24 hours after you call.

(818) 447-4565 m-f 8am-5:30pm pst. **Visa, MC, COD**

P.C. ELECTRONICS
2522 Paxson Ln Arcadia CA 91007

Tom (W6ORG)
Maryann (WB6YSS)



DOWN EAST MICROWAVE

Amateur Microwave Antennas and Equipment

902, 1269, 1296, 2304,
2320, 2400, 3456 MHz

TROPO, EME, WEAK SIGNAL,
OSCAR MODE L, MODES,
ATV, REPEATERS

LOOP YAGIS, POWER DIVIDERS, COMPLETE ARRAYS
KIT FORM OR ASSEMBLED AND TESTED
SOLID STATE LINEAR AMPLIFIERS FOR 902 & 1296 MHz

Write for Free Catalog to:

DOWN EAST MICROWAVE
Bill Olson W3HQT, Box 2310 RR1
Troy, ME 04987 (207) 948-3741

RF

Zack Lau, KH6CP/1

Designing a High-Performance 13-cm Preamp

Al Ward, WB5LUA, has designed some fine preamps for 13 cm using PHEMT devices made by Hewlett Packard.[1] He has wrung those devices just about dry, so I decided to see whether I could do any better with NEC's high-performance PHEMTs. To do that, I had to forget any idea of a no-tune design. (If you want a no-tune preamp, build Al's ATF-10135 preamp in the *1993 ARRL Handbook*, page 32-22.) If you are looking for the ultimate preamp and have the ability and resources to tweak a circuit, you might find this design useful.

Design Strategy

The first thing I did was look at the losses of preamps such as Al's PHEMT design. This uses a series capacitor and series inductor at the input. One of the things Al did to reduce the input loss was to elevate the inductor above the board to reduce dielectric loss. But there is a weak area: the input capacitor. Look at Table 1, which shows the Q of several values of ATC 100A chip capacitors at various frequencies. Notice how bad the Q gets at microwaves. I've since heard that better chip capacitors are available—except that they can't be soldered in like ordinary capacitors, since they are really meant to be attached with lead-bonding equipment. Few amateurs have access to such exotic equipment; I know *I* don't!

Anyway, I decided to see what I could do to eliminate the lossy input capacitor. The elegant solution is to ground the gate and source-bias the device. Source biasing is simple and reliable. Its disadvantages, supposedly, are decreased broadband stability and worse noise figure. But is this really true? As many designers have discovered, for many devices there are specific amounts of source inductance that actually *enhance* stability. Perhaps the source-biasing capacitor can add just enough stray inductance to stabilize the device. As for the reportedly worse noise figure, I think it stems from poor source bypassing. In the design presented here, the increase in noise figure due to source losses is more than offset by the improvement in input loss. If you just add source bypassing and don't improve the input circuit, you can expect results to get worse.

I found that 100-pF ATC 100A chip capacitors do a pretty good job of source bypassing NEC 32684A PHEMTs. You might experiment to see if there is actually a better source bypass; I merely investigated the short list of candidates I had on hand. (For some reason, I always seem to be designing preamps a few days before a noise figure contest—when there isn't enough time to order more or better parts!) There might also be a better electrical model for the transistor leads. The device sits on two chip caps whose center-to-center distance is about 140 mils. Is the lead length to model the part to the edge, or all the way to the center of the chip capacitor? Or is it something else? I'm not kidding! The difference is quite significant with lead lengths on the order of 15 mils and chip capacitors 55 mils wide.

As both Jim Davey, WA8NLC, and Randy Rhea, N4HI, pointed out at the 1993 Microwave Update just held in Atlanta, you probably want all of the source-bypass capacitors to be of the same type. Otherwise, you may get parallel resonances that severely degrade the bypassing at certain frequencies. Of course, this effect probably isn't too bad given the lousy Q these capacitors have at microwaves! Still, it may be an effect worth considering, especially if you are trying to find out why your preamp doesn't work right. In any case, I *don't* recommend installing 6 or 7 source-bypass capacitors in hopes of blindly improving this design unless you have a good way of avoiding such problems. A good computer program, coupled with accurate device models, is probably the only way to handle such complicated situations. I don't have accurate device models because I don't have a microwave network analyzer for such measurements. Instead, I often estimate parasitics based on the physical dimensions of the part, with corrections based on circuit performance or educated guessing.

Finally, I decided to make the preamp unconditionally stable under all loads, although I suppose you might want to compromise on this if you just wanted to win a noise figure contest and knew what mixer your preamp would be tested with. I planned to eventually use one of these with a mode-S converter, so I decided to build something useful. Resistor RD1 does a fine job of obtaining unconditional sta-

Table 1—ATC 100A Chip Capacitor Q

Capacitance (pf)	*Frequency (MHz)*	*Q*
470	150	150
100	1000	18
10	2304	14
10	3456	7
10	5760	3
1	5760	11
1	10368	4

[1] A. Ward, "PHEMT Low Noise Amplifiers for 2304 MHz and 3456 MHz," *Proceedings of Microwave Update '92*, published by the ARRL, pp 148-152.

225 Main Street
Newington, CT 06111
email: zlau@arrl.org (Internet)

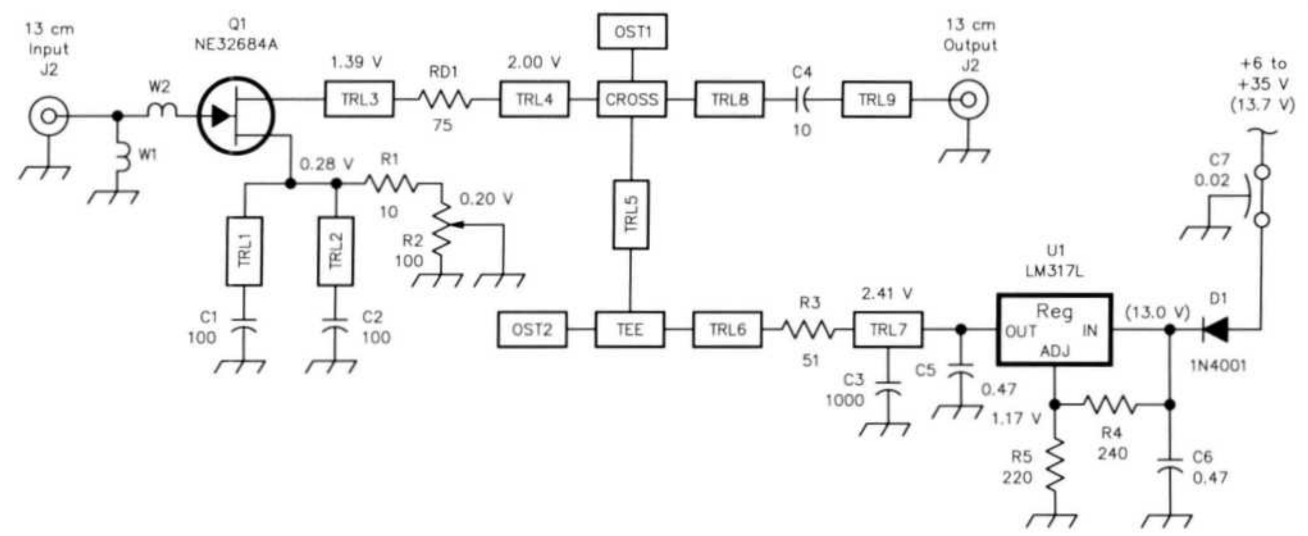


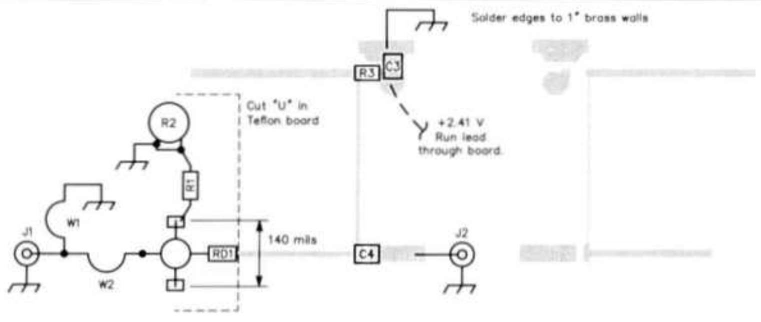
Fig 1—Schematic of the 2.3- to 2.45-GHz preamplifier with measured dc voltages shown at key points.

C1, C2—100-pF ATC 100A chip capacitors. Substitution *not* recommended.
C3—1000-pF chip capacitor.
C4—10-pF chip capacitor.
C7—0.02-µF feedthrough capacitor or EMI filter. Value not critical: anything from 100 pF to 0.1 µF should work fine.
D1—1N4001 reverse polarity protection diode. This is useful in noise figure contests when you hook up the power supply backwards!
J1—SMA connector with captivated center contact.
J2—SMA connector.
RD1—75-Ω chip resistor.
U1—LM317L adjustable three-terminal regulator IC.
W1, W2—wire loops made from no. 32 silver-plated copper wire. See Fig 4 for a precise description.
Q1—NEC 32864A PHEMT.

bility over a wide frequency range. I thought it might also reduce the preamp's sensitivity to variations in the output load, but the improvement is small. Single-stage preamps will often be quite load sensitive, particularly as you get close to the maximum useful frequency of the device. This one is no exception.

Construction

The key is the input circuit: it has to be as low-loss as possible. I built two of these preamps, one with 1-inch-high walls and another with ½-inch walls. The one with 1-inch walls is pretty much insensitive to attachment of a cover, with the gain being affected by only a few hundredths of a dB. On the other hand, the one with ½-inch walls will occasionally measure a tenth or two higher than normal when a cover is installed but not tightly attached to the walls with screws, despite some microwave absorber material attached to the cover. I suspect that the inductor loops just don't have enough room in the smaller preamp, and the cover interacts with the loops. The two input loops are made from 32-gauge silver-plated solid copper wire. This is actually a single strand taken from some 20-gauge Teflon hook-up wire. Ordinary copper should do as well, but I figured that every little bit helps! The SMA



Solder edges to 1" brass walls
R3
C3
+2.41 V Run lead through board.
Cut "U" in Teflon board
R2
R1
W1
140 mils
J1
W2
RD1
C4
J2

Fig 2—Parts-placement diagram for the 13-cm preamplifier. A mirror image of the board I etched should also work just fine.

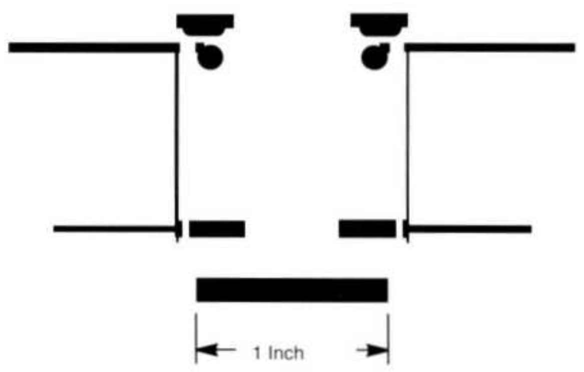


Fig 3—Etching pattern for the 13-cm preamplifier output network. (A Postscript image file is available from the ARRL BBS, 203 666-0578, or via Internet on ftp.cs.buffalo.edu in the /pub/ham-radio directory.)

```
*2304 HEMT preamp using the NEC 32684A
*bias 2 volts at 10 mA

LL:15MIL
* source lead length

W50:90MIL
* 50 ohm line thickness

WBIAS:19MIL
* bias line width

PBIAS:875MIL
* bias line length--roughly quarter wavelength

WOUT:35.2662MIL
* output matching line width

POUT:528.111MIL
* output matching line length

WOS:16.6553MIL
POS:27.8347MIL
* matching stub

BLK

  WIRE 1 0      D=8MIL L=250MIL  S= 200MIL  SUB2
  WIRE 1 4      D=8MIL L=?322.574MIL? R=1.59  S=?97.5077MIL? SUB2

* input matching network--wire loops
* L is the length of wire, S is the separation between wire
* ends.  Height of ends is 55 mils. Silver wire is modeled.

  two 4 39 300  NE326
  trl 39 40     w=20mil p=20mil sub1

  srl 40 44      r=75 l=2nh

*added resistor for stability at a minor loss in gain

  trl 300 310  w=20mil p=11 sub1
  trl 300 320  w=20mil p=11 sub1
  slc 310 0    l=0.28nh c=100pf q=1.26 f=2.3ghz
  slc 320 0    l=0.28nh c=100pf q=1.26 f=2.3ghz
  srl 320 0    r=10 l=30nh

*source biased FET.

  trl 44 50    w=wout p=pout sub1
  cross 50 60 72 150 w1=wout w2=wos w3=w50 w4=wbias sub1
  ost   60  w=wos p=pos sub1

  trl   150 152 w=wbias p=pbias sub1
  tee   160 170 152 w1=50mil w2=50mil w3=wbias sub1
  trl   160 162 w=50mil p=20mil sub1
  srl   162 164 r=51 l=2nh
  trl   164 166 w=50mil p=50mil sub1
  slc   166 0   l=0.4nh c=1000pf
  ost   170      w=50mil p=pbias sub1

*output biasing

  trl 72 74 w=w50 p=20mil sub1
```

```
  slc 74 76 l=0.3nh c=10pf q=4 f=6ghz
  trl 76 78 w=w50 p=300mil sub1


  amp: 2por 1 78


end

freq
 2304mhz 2400mhz 2447mhz
  step 0.1ghz 1ghz .2ghz
  step 1ghz 20ghz 1ghz

end

opt
*amp ms11 .1 lt
amp ms22 .1 lt
amp nf .3db lt
end
data
NE326:S
*NE326  NEC32684A 2V, 10mA
  .1ghz .999   -2.3 6.402  178.  .002  88.9 .48    -2
  .2ghz .999   -4   6.366  176.1 .003  88.6 .48    -2.9
  .5ghz .998   -9.6 6.384  170.6 .007  87.6 .48    -5.9
 1ghz   .986  -18.9 6.298  161   .014  77.6 .476  -11.5
 2ghz   .948  -36.2 6.108  143.6 .026  72.2 .467  -22
 3ghz   .894  -52.3 5.792  127.1 .038  62   .449  -31.2
 4ghz   .833  -67.8 5.404  112.1 .049  55.8 .436  -40.4
 5ghz   .766  -82.1 5.063   97.6 .058  49.4 .413  -49.2
 6ghz   .703  -95.8 4.713   84.3 .067  44.1 .394  -58.3
 7ghz   .653 -108.6 4.385   71.3 .074  38.3 .382  -67.4
 8ghz   .607 -119.6 4.112   59.8 .082  33.8 .374  -74.9
 9ghz   .569 -130.6 3.861   48.6 .09   28.3 .375  -83.3
10ghz   .529 -141.4 3.686   37.5 .097  23.4 .378  -90.8
11ghz   .487 -153.2 3.517   26   .107  18.4 .374  -99.4
12ghz   .554 -166   3.376   14.8 .114  11.8 .365 -108
13ghz   .428 -179.3 3.262    4.1 .121   6.3 .354 -116.8
14ghz   .407  167.8 3.165   -5.9 .13     .6 .341 -125.5
15ghz   .387  154.5 3.081  -17.3 .138  -5.4 .345 -136.6
16ghz   .368  139.9 3.007  -27.9 .146 -13.5 .352 -148.4
17ghz   .359  124.4 2.934  -40.2 .154 -21.6 .353 -160
18ghz   .358  107.6 2.91   -52.3 .16  -30.1 .349 -170.6
19ghz   .361   89.4 2.865  -65.4 .169 -39.1 .337  177.9
20ghz   .379   73.7 2.812  -77.8 .174 -49   .319  165.2

noi rn
1ghz  .28 .9    17 .45
2ghz  .3  .85   32 .37
4ghz  .33 .72   64 .27
6ghz  .37 .62   91 .21
8ghz  .40 .54  116 .15
10ghz .45 .48  138 .1
12ghz .5  .42  164 .07
14ghz .62 .38 -169 .07
16ghz .75 .34 -139 .08
18ghz .91 .34 -101 .09
20ghz 1.1 .38 -77  .1

sub1:ms h=30mil er=2.2 met1=cu .7mil rgh=75uin tand=0.0011

* output substrate is 30 mil 5880 Duroid or equivalent

sub2:ms h=55mil er=1 met1=au .7mil

* input substrate is low loss "air microstrip"  stripline?

end
```

Fig 4—*Microwave Harmonica* input file.

connector has a captivated center contact. This is important since the contact isn't attached to anything rigid. I increased the diameter of the center pin with a 0.27-inch length of 3/32-inch diameter brass tubing to decrease the impedance of the input line, making it a little closer to 50 Ω. I could have just moved the pin closer to ground, but this would have made accidentally shorting the input to ground that much easier. The gap between the center pin and the ground plane looks to be about 55 mils.

The entire input network, transistor, and most of the drain resistor are built over a copper ground plane (a piece of double sided, unetched circuit board). The drain resistor connects from the transistor to the printed-circuit board output network, which is on 1/32-inch Teflon board (with a dielectric constant of 2.55) that is soldered to the unetched board. The Teflon board has a "U" cut into it to accommodate the PHEMT and input network. A lead is brought through the Teflon board for the 2.5-volt supply, which is built on the underside of the preamp.

The preamp was adjusted for lowest noise figure by varying the two input coils with a pair of stainless tweezers and a soldering iron. Yeah, I know it's not the easiest technique. A trimmer capacitor might make things

```
  Freq     NF      MS21    MS11    MS22     MS11     MS12      K
  GHz      dB       dB      dB      dB       dB       dB
           AMP     AMP     AMP     AMP      AMP      AMP      AMP

  0.100   23.01 -17.43   -0.01  -32.41   -0.01   -86.93  102.34
  0.300   13.37  -7.42   -0.01  -23.26   -0.01   -70.31    8.31
  0.500    8.91  -2.09   -0.01  -19.14   -0.01   -60.58    2.00
  0.700    6.14   1.85   -0.02  -16.81   -0.02   -53.64    1.12
  0.900    4.27   5.05   -0.06  -15.46   -0.06   -48.22    1.15
  1.000    3.57   6.50   -0.11  -15.03   -0.11   -45.87    1.23
  2.000    0.52  19.04   -9.53  -11.65   -9.53   -27.05    1.28
  2.304    0.38  17.77   -8.77  -17.38   -8.77   -26.81    1.38
  2.400    0.39  17.06   -7.06  -17.34   -7.06   -27.07    1.42
  2.447    0.39  16.69   -6.38  -16.91   -6.38   -27.23    1.44
  3.000    0.84  12.40   -2.60  -11.51   -2.60   -29.18    1.68
  4.000    3.02   3.55   -1.02   -5.87   -1.02   -34.26    3.09
  5.000    4.91   3.16   -1.02  -11.66   -1.02   -31.68    2.58
  6.000    7.13  -0.68   -0.77  -14.43   -0.77   -32.84    3.80
  7.000    8.31  -1.99   -0.44   -4.85   -0.44   -32.02    2.11
  8.000    9.39  -3.02   -0.39   -5.79   -0.39   -31.16    2.12
  9.000   15.09 -16.46   -0.29   -3.45   -0.29   -42.98   16.47
 10.000   10.44  -5.34   -0.41   -7.06   -0.41   -30.53    2.55
 11.000   14.44 -16.57   -0.50   -2.52   -0.50   -40.41   17.48
 12.000    9.54  -4.55   -0.46   -8.73   -0.46   -26.72    1.79
 13.000    8.54  -4.36   -0.76   -5.37   -0.76   -26.02    1.92
 14.000    8.31  -9.13   -0.80   -2.08   -0.80   -29.89    2.81
 15.000    5.80  -3.32   -1.67   -5.11   -1.67   -23.08    2.07
 16.000    7.72 -11.51   -3.47   -1.60   -3.47   -30.43   10.08
 17.000    3.08  -1.31   -7.26   -3.26   -7.26   -19.34    2.06
 18.000    3.22  -2.94  -15.38   -1.71  -15.38   -20.34    2.31
 19.000    4.41  -2.31   -5.97   -3.85   -5.97   -19.16    2.69
 20.000    4.99  -8.04   -2.64   -0.74   -2.64   -24.34    1.23
```

Fig 5—*Microwave Harmonica* analysis output.

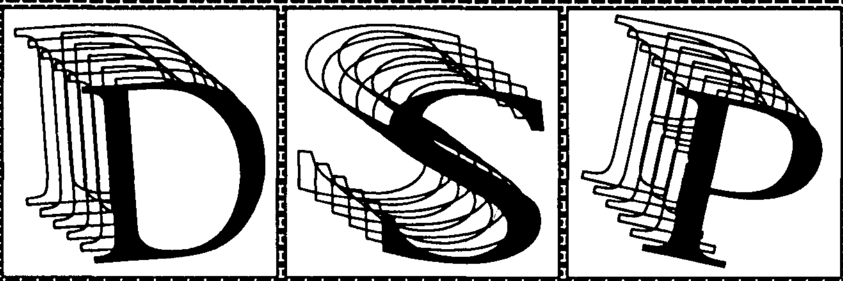
easier, but the losses would almost certainly be worse, degrading the noise figure.

You may want to tie the gate and source leads together while rearranging things—a little static electricity can easily damage the transistor while the gate is floating. Soldering irons can also damage sensitive semiconductors. It isn't unusual for insulators to degrade when heated to high temperatures.

How Does it Work?

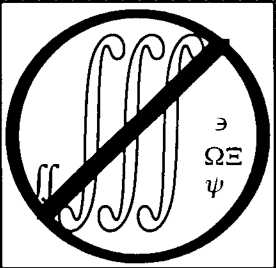
The measured noise figure is approximately 0.4 dB, with a gain around 14 to 17 dB. The gain is a few dB lower—and the noise figure a few hundredths lower, too—as you go up to the high end of the 13-cm band, making the preamp quite useful for satellite work as well as for terrestrial use. More precise numbers are meaningless, since I've seen the same preamp measure as low as 0.33 and as high as 0.46, using different HP 346A noise sources and the same transverter (mixer/filter).

As this design shows, a source-biased preamp can do surprisingly well compared to other designs—even at 13 cm. As always with low-noise microwave circuits, attending to the input losses—wherever they come from—will pay off in improved noise figure. □□



Without Tears™

Learn DSP and Put your Knowledge to work immediately!



Coming to a Place Near You

Ft. Lauderdale
Albuquerque
Washington D.C.
Chicago, Houston
San Jose, Long Beach

Call Z Domain Technologies, Inc.
(800)-967-5034
(404)-664-6738

Call 9-5 EST. Ask for brochure.

Our 2-Day Advanced Course is ready. Call for info

By Taking This 3-Day Course You will really learn DSP. Guaranteed!

Surface Mount Chip Component Prototyping Kits—Only $49.95

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

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

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