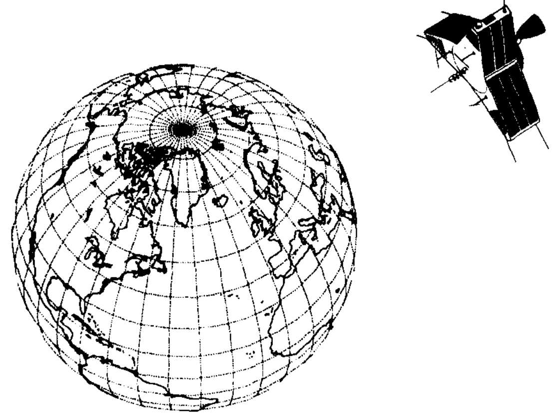
QEX 17
July 1983

The ARRL Experimenters' Exchange



Earth View of Phase IIIB at Apogee 35,800 Km.

AMSAT-OSCAR 10 Launch Successful

A few seconds before noon UTC on Thursday, June 16, 1983, the European Space Agency's Ariane L6 rocket thundered skyward from its launch pad in Kourou, French Guiana. High in the nose of the rocket, wrapped within a carbon-filament "cocoon" called the Sylda, rode the AMSAT Phase IIIB spacecraft. This launch, unlike the Ariane L02/AMSAT Phase IIIA mission of three years earlier, was destined to succeed. Capping a letter-perfect flight to transfer orbit, at 12:16:53 UTC Phase IIIB was separated from the launcher, becoming AMSAT-OSCAR 10.

Shortly after 14:44 UTC the first telemetry reports began to reach the AMSAT engineering crew. The spacecraft was functioning well, and the transfer orbit was right on the money -- but every indication was that OSCAR 10 was in a disadvantageous orientation toward the sun. The solar-array current was low; the spacecraft temperature was lower than expected; the temperature difference between the +Z (antenna) and -Z (kick-motor-bell) surfaces was high, and the sun-angle channel was improper. The kick-motor firing, scheduled for the 4th apogee, was postponed until the problem could be rectified.

AMSAT needed time to analyze the situation. Compounding the problem was the low (200-km) perigee of the transfer orbit: OSCAR 10 could not be left in the orbit indefinitely as atmospheric drag at perigee would cause the orbit to decay over a month or more. What were the options? The satellite is equipped with magnetorquers that can be pulsed under computer control to cause the desired reorientation. The magnetic field of the torquing coils, caused by the pulsing, and the magnetic field of the earth would interact, producing the desired torque and resultant movement. But precious time would be needed to determine precisely the spacecraft's physical behavior.

Continued observation revealed, however, that the satellite was shifting orientation in a favorable direction a little over 3 degrees per day with *no* active AMSAT intervention. Postulating that normal seasonal movements of the earth in relation to the sun, differential drag at perigee on the kick-motor bell housing and possibly eddy currents induced as the spacecraft frame passed through the earth's magnetic field were causing the desired movement at a sufficient rate, AMSAT decided to wait for the natural corrections before firing the kick motor. As this is being written, AMSAT is waiting until sometime in the neighborhood of orbit 50 or about July 7.

When OSCAR 10 is properly oriented the initial kick-motor firing will lift the perigee to a safe 1500 km. Then, once the spacecraft is stabilized and again oriented properly, the final burn will lift the inclination angle from about 11 degrees to the final 57 degrees. Following another period of ranging and testing, the transponders will be turned on, and a normal operating schedule will begin.

Here are the latest transfer orbital elements:

Epoch:	83.180.5 (noon UTC June 29)
Inclination:	8.5210 degrees
R.A.A.N.:	243.7810 degrees
Eccentricity:	0.7286985
Argument of Perigee:	188.7810 degrees
Mean anomaly:	311.0310
Mean motion:	2.30084130 orbits/day
Decay rate:	0.000155 orbits/day squared
Anomalistic period:	About 625.9 minutes (perigee to perigee)

Typical cw telemetry received on the General Beacon a week after launch:

UBAT (battery voltage):	14.5 V
TBAT (battery temperature):	7.1 to 4.3 degrees C (varies during orbit)
IARRAY (solar current):	0.4 A
SA (sun angle):	0.1 (beyond sun sensor limits)
SPIN:	2 r/m (later spun up to over 41 r/m)

Congratulations to the AMSAT crew for a great launch and a continuing great job of nursing OSCAR 10 into its final orbit. - Steve Place, WB1EYI.

Correspondence

An AMTOR Protocol Change?

The advantages of AMTOR have interested me ever since reading "Amtor, An Improved Error-Free RTTY System" by J. P. Martinez, G3PLX (June 1981 QST, pp. 25-27). However, it is possible some problems would be solved by modifying the protocol. Each seven-bit character could be surrounded by start and stop bits, thus changing from synchronous to semi-synchronous operation.

Currently, AMTOR cannot be implemented with off-the-shelf components in the same sense that standard RTTY can be implemented with a UART chip or HDLC (packet radio) can be implemented with an Intel 8273 chip. The only approaches I know of use dedicated microprocessors. This doubtlessly discourages many hams from using AMTOR and enjoying its advantages. If start and stop bits were sent, standard UART chips could be used for character generation and reception. UART chips are cheap, widely available and often already included in personal computers. The small loss in efficiency from the extra bits would be more than offset by the increased ease in getting on the new mode. The degree of compatibility is demonstrated by the fact that my present homebrew computerized RTTY system, designed and built for standard RTTY, would need no hardware modifications for the new semi-synchronous AMTOR.

Lessening of the critical timing requirements inherent in synchronous protocols is another advantage. Some degree of synchronization between two stations must be maintained to prevent data collision (both stations transmitting at the same time). This allows each station to determine when the other station's tranmission, lost in the noise, is overdue so a request for a repeat can be made. With the microprocessor no longer responsible for character generation and reception, synchronization amounts to waiting for the "data ready" flag from the UART and requesting a repeat if this has not occurred by a certain time. If no characters have been lost, this synchronization procedure is unnecessary.

It is difficult for anyone other than the two conversing stations to monitor the communication on the current AMTOR protocol. This is a significant disadvantage. Changing to the new semi-sychronous system would allow monitoring almost as if the protocol was fully asynchronous, although the error correcting possibilities of AMTOR would be unavailable to the monitor. This would make the change attractive to the FCC, who are always concerned about such things.

This is not meant to say that synchronous protocols are burdened with so many disadvantages that they should be avoided at every opportunity. It is that AMTOR does not make use of the advantages of synchronous systems while carrying all of the inherent disadvantages. If we continue to use the current synchronous protocol, we ought to get the most out of it. For example, some of the technology for coherent cw from a few years ago might be applied to AMTOR. Error-free RTTY using mark-only or space-only detection could be possible. If it could be made to work, modulators and demodulators would become a thing of the past. On the coming elliptical-orbit satellites (such as Phase IIIB), battery drain could be substantially reduced by lowering the RTTY duty cycle. All of this is possible only with a synchronous protocol, and it would be worth the trouble. Still, we are not doing it and until we do, adding start and stop bits around each charcter might be the best approach. - Michael S. Bilow, N1BEE, Forty Plantations, Cranston, RI 02920

Modifications to Real Time Satellite Tracking Programs for HP-41C/CV Programmable Calculator

The November, 1982 issue of QEX contained an article on two HP-41C/CV programs for real time tracking of satellites. A program bug that did not reveal itself until recently exists in both of these programs. The bug has almost no effect on circular orbit calculations but shows up quickly on highly eliptical orbit calculations. It exists in "SATELLITE ORBITS I" dated 9/10/82 and "SATELLITE ORBITS II" dated 10/1/82.

The "fix" for both of these programs is easily done using the procedures listed below. One caution however; make the changes in the sequence shown or you might change the wrong program step.

CHANGES TO PROGRAM "SATELLITE ORBITS I"

1. Delete program step 486 (RCL 24).
2. Delete program step 480 (STO 22) and replace it with the following two new steps:
   - New program step 480 X 22.
   - New program step 481 SIN.
3. Delete program step 454 (STO 24).

CHANGES TO PROGRAM "SATELLITE ORBITS II"

1. Delete program step 425 (RCL 24).
2. Delete program step 419 (STO 22) and replace it with the following two new steps:
   - New program step 419 X 22.
   - New program step 420 SIN.
3. Delete program step 393 (STO 24).

My thanks to NØAN for calling this bug to my attention. - Roy D. Welch, WØSL, 908 Dutch Mill

Update on the Sinclair Computers for RTTY Terminals

I have a short update on the use of the Sinclair Computers for RTTY. This should help get people plugged into low-cost RTTY.

My first impression is that these computers could serve as terminals for AMTOR, or even packet radio at low data rates, i.e., typical of hf operation. All of the RTTY programs operate in a polling mode at the video frame rate. This means you can handle up to 60 characters/s. Not bad for the price.

The development of software and inferface circuits to allow the Sinclair ZX81 and TS-1000 computers to serve as RTTY terminals has moved rapidly. Today you can buy the computer and the parts for the interface for $50 - $60. Since so much work has been done I will only summarize where the programs and data are available.

Baudot RTTY Programs for the ZX81 and TS-1000:

See QZX Vol. 1, No. 5.

1. Available from: Alex F. Burr, K5XY, 2025 O'Donnell Drive, Las Cruces, NM 88001.

2. Contact: L. Willson, AF8J, 149 Hoehn Court, P.O. Box 465, Dimondate, MI 48821.

3. Contact: Scarab Systems, 141 Nelson Road, Gillingham, Kent, ENGLAND ME7 4LT

4. See: Sinclair Projects, February/March, 1983, p. 12 (receive-only system).

5. "ASCII RS-232-C Interface & Program," Radio & Electronics World, February 1983, p. 74.

Most of these programs take advantage of the more powerful system that is resident in the ZX81 and TS-1000 computers. In most cases, the program is a mixture of BASIC and machine language that can be loaded from a cassette. However, if the program were entirely done in machine language, then it could be held in ROM or nonvolatile RAM.

I have a transmit program for the ZX80. If interested, please sent an s.a.s.e. - Ken Heitner, WB4AKK, 2410 Garnett Court, Vienna, VA 22180.

Continuous RTTY Reception on the ZX80

By Kenneth Heitner,* WB4AKK

This article describes a program and interface originally developed for the ZX80 which will allow it to receive and display RTTY. Since then, a more powerful ZX80 (and Z1000) have become available. While this system is not directly compatible with these newer computers, it can be adapted for their use.

The program was tested at 60 wpm only, but the electronics are sufficiently fast to operate at higher RTTY speeds. The Baudot code was used at one signaling rate. The approach was to **keep it simple**. All the components used (and their specifications) are available and amateurs should be able to adopt these concepts to operate with a wider variety of codes, speeds and interface devices.

Use of the UART Interface

This system was designed around the use of a Universal Asynchronous Receiver Transmitter (UART) located in the hardware interface. The UART performs the conversion of the serial Baudot code to parallel form for processing in the computer. Use of the UART is critical to the system design because it enables the microprocessor system to spend most of its time providing the video signal to the display. With a simple computer such as the ZX80, this appears to be the only approach that will allow continuous display of received characters. It also simplifies system timing since the video display timing and received character rate are entirely indpendent.

The Interface

The use of the UART as a serial interface for the ZX80 microprocessor is described by Ciarcia [1], and specifications for the AY-5-1013 UART can be found in "The Microprocessor/LED Book". [2] Basically, the UART automatically monitors the serial input line for incoming data pulses. In the circuit shown in Fig. 1, the UART is hard wired to look for 5-bit (Baudot) characters. When a character is received, it is stored in an internal buffer for use by the computer.

The key to the interface is the 74LS138 multiplex decoder. Use of this decoder was inspired by Brian Davis, W9HLQ. It allows three basic computer commands to control the UART.

1. Determine if the UART has a valid character for the computer.

2. Transmit (read) the character into the computer.

3. Reset the UART for the next character.

The decoder assures these commands are sent to the right gates on the UART at the right time. Because of timing problems with the first command, the external gates (74LS367), shown in Fig. 1, were added. The UART circuit is fairly simple to build, but it is a good idea to adequately bypass the +5-V supply at each chip.

*2410 Garnett Court, Vienna, VA 22180.

Interface Clock

The UART has to know what speed the input is expected. Therefore, it must have a clock of its own at sixteen times the bit rate. For 60 wpm, it is simply done with the timer circuit in Fig. 2. More sophisticated crystal-controlled multi-speed clock circuits are available. [3]

Interface to RTTY Loop

Test characters can be generated on your hardware by interfacing its existing RTTY loop. This is useful for testing purposes. A simple optocoupler circuit given in Fig. 3 will assure complete isolation of the computer from high-voltage loops.

Testing the UART Circuit

A short test program in BASIC and machine language was written to check the hardware and determine if the UART circuit is functioning properly. The machine language portion of the program is stored in a REM statement. 4 Enter the BASIC program listed in Table 1. Use the command GOTO 100 to load the machine language in Table 2. [5]

In testing the program, use your RTTY equipment to send characters to the computer via the interface. Each character should show up as a number between 0 and 31. Approximately 100 characters must be sent before the display appears. The number corresponding to each character will be discussed in the lookup table.

If the UART circuit checks, proceed with the main program. If not, try to check the UART interface with the simple test program.

A Sketch of the Main Program

The program used to operate the ZX80 as a continuous RTTY receive display is outlined in Fig. 4. The program structure is determined by the need for the microprocessor primarily to generate the video signal for the display. All the other functions must be accomplished during the one ms video retrace. This can best be appreciated by understanding the video generation process.

Video Generation

Video is generated in the ZX80 by a combination of on-board hardware and software. The basic technique is to place the characters in a display file and scan them to generate a video signal. The process is explained in detail in "Microace/ZX80 Video Secrets Revealed". [6] These machine-language techniques are used to write a video routine capable of displaying up to four lines of text. Each cycle of the program generates one frame of video with a total time of 16 ms. To provide a continuous display after each frame, the program loops back for the next frame after an appropriate retrace delay. Only if a key closure is detected will the program exit to the BASIC control program.

UART Read

As shown in Fig. 4, the end of the video generation cycle is used for the UART read. That

(continued on next page)

Continuous RTTY Reception on the ZX80
(continued from previous page)

is, the UART is actually pulled every 16 ms to see if a new character has been received. This is faster than the characters are received (167 ms for 60 wpm). Thus on most loops, the retrace continues after the UART read cycle through the normal time delay. The generation of the next frame is then begun. If there is a new character in the UART buffer, the program branches to the right in Fig. 4. The receive character is transferred to a register for processing and the UART is reset.

Character Shift

Blank or shift character is the first determination made on the received character. Blank character has a value of zero. This value simply initiates a return to the video loop through the normal time delay.

If the received character is figures (27) or letters (31), then shifting is initiated through separate routines. These routines change the address of the first byte of the lookup table. If we shifted to figures for example, this value would be changed to the address of the first byte of the figures table. All incoming characters are read from the figures table until a downshift is received. This program **does not** unshift on space.

Lookup Table

All other characters represent data but must be converted to the Sinclair character code to be displayed. [7] The characters are listed in order of their Baudot code value which simplifies the lookup table. This makes the first character blank (0) and the last letters (31). The table is 64 characters long. This includes all letters and figures characters. The ordered arrangement of the table allows indexed addressing instructions to be used in the lookup program.

The table is a series of memory locations whose contents are the Sinclair character codes of the received Baudot characters. For example, imagine the tenth entry in the table to be the character R (Baudot value 10). The entry in that memory location is 55 which is the Sinclair code for R.

Display File

The display file for the ZX80 8,9 in this program is restricted to four lines for memory conservation. It is filled with Blanks (Sinclair code 0), so the initial display is a blank screen. Received characters are simply placed in the display file during the retrace period after lookup. The display file program indexes to the next vacant display file address after each character is received. At the end of the line, the file pointer is indexed past the end-of-line character to start a new line. The character delay assures proper video timing after each new character is received looked up and displayed.

Scroll Display

A scroll is initiated when the screen is filled. The last three lines are moved up one line and the last line is refilled with blanks. The display pointer is reset and a display of new characters is now on the last display line.

BASIC and Machine-Language Program Listings

The BASIC and machine-language programs used for continuous RTTY receive follow the same format as the short UART test program stated earlier. The machine-language program is held in a long (280 byte) 1 REM statement shown in Table 3. This program is loaded after the BASIC program is typed into the computer by use of the GOTO 100 command. Load 25 to 30 bytes of the machine language each time, being careful to verify program correctness.

BASIC Program Highlights

The BASIC program has two simple functions. The double loop starting on line 10 fills the display file with blanks. This file must be created before the machine language video-generation routine is called to assure stable operation. The POKE commands start on line 31. It sees that the printing starts at the beginning of the display area. Once the video-generation routine is called, the remaining operations are machine language unless a keypress (shift key) is used to return to BASIC.

Machine-Language Program Highlights

The machine-language program functions have been explained in an earlier section. To aid in understanding the program, review Table 4. It relates the main parts of the program from Fig. 4 to the actual listing in Table 5.

Suggestions for Improvements

This program and interface is basic and does not contain many features. Some simple changes the user might consider adding through a program change include shift on space, carriage return and word wraparound. In word wraparound, a new line is started if a space occurs in the last 5 to 7 characters. Normally few words in the display are split, but it is unavoidable for longer ones.

Other Speeds and Codes

More sophisticated hardware and software can handle other speeds and codes. Higher baud rates can be obtained by changing the UART clock rate. A different program can be devised to handle a limited version of ASCII, such as ASCII 6. 10 This version of ASCII has all upper case characters which are compatible with the limited display capabilities of the ZX80. Changes require rewiring the UART to handle a 6-bit code, as well as a new lookup table. No shift program is required.

Transmission from the Keyboard

Transmission from the keyboard in real time is a possibility using this approach. Keyboard scanning, character lookup and output of new data to the UART's transmit buffer would have to be done during the video retrace delay. Suggestions on how to scan the ZX80 keyboard are offered in Williams book, "Microace/ZX80 Video Secrets Revealed." [11]

Addendum

The following suggestions on program operation were made by W9HLQ.

1. If you make an error during loading machine code, enter any letter such as Q. This returns to the command mode softly, and you can reload the machine language correctly.

2. Intiation of the display file in the BASIC program (Table 3) could be done as:
```
10 FOR I = 1 TO 160
20 PRINT "-";
30 NEXT I
```

3. The stability of the display can be enhanced by careful adjustment of the time delay constants. They are:
   Normal delay at 16469
   Shift delay at 16695
   Character delay at 16673

References

[1] Ciarcia, "Build Your Own Z80 Computer," BYTE Books.

[2] "Microprocessor/LED Data Book," Jim-Pack Electronic Components.

[3] Hoff, "All About the UART," RTTY Journal.

[4] QEX, January 1982.

[5] Zuks, "How to Program the Z80."

[6] Williams, "Microace/ZX80 Video Secrets Revealed," May 1981.

[7] Davenport, "A Course in Basic Programming," Sinclair Research Ltd.

[8] See Reference 6.

[9] See Reference 7.

[10] Lancaster, "TV Typewriter Cookbook," Howard W. Sams & Co., Inc.

[11] See Reference 6.

(continued from previous page)

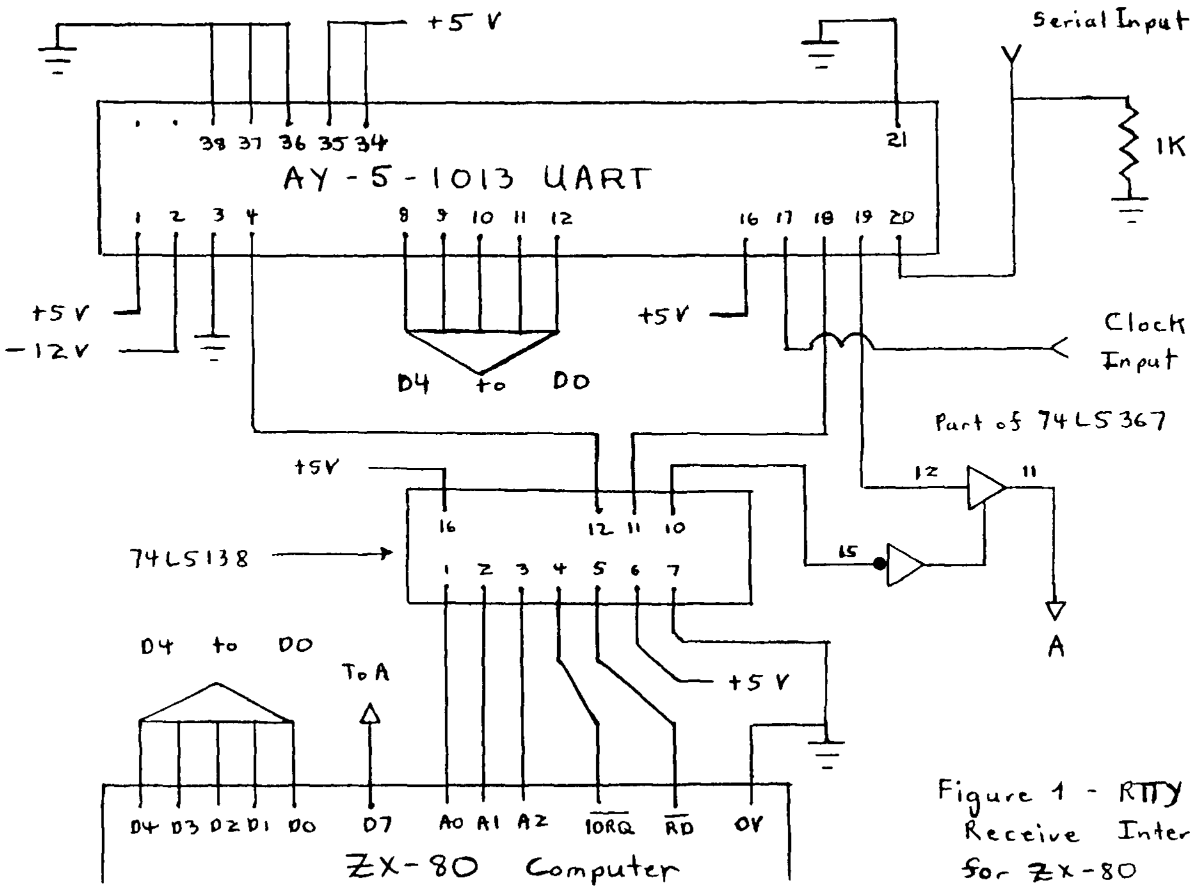
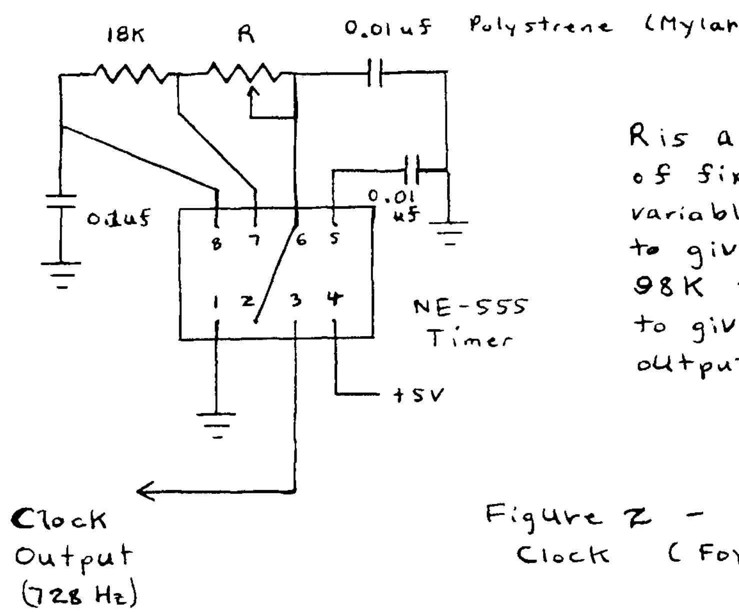


Figure 1 - RTTY Receive Interface for ZX-80



R is a combination of fixed and variable resistors to give about 98K - Adjust to give 728 Hz output

Figure 2 - RTTY Clock (For 60 WPM)

(continued from previous page)

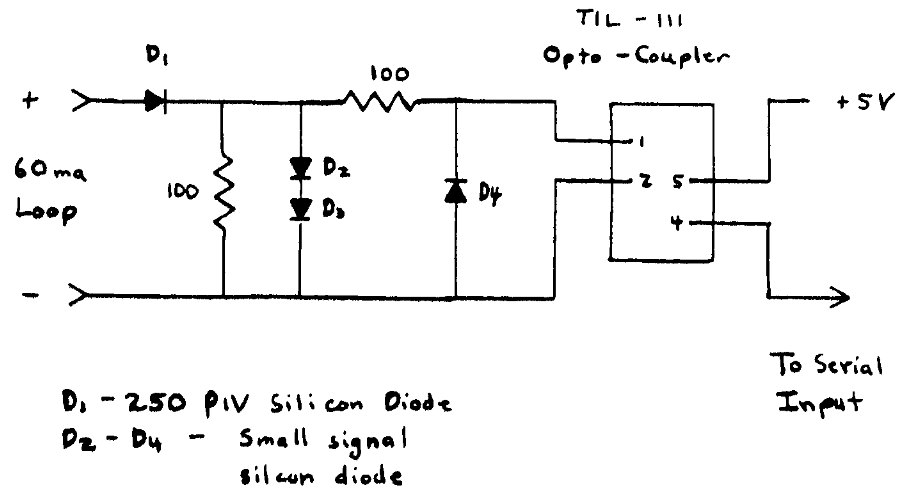


Figure 3 - Interface to 60 ma RTTY Loop Circuit

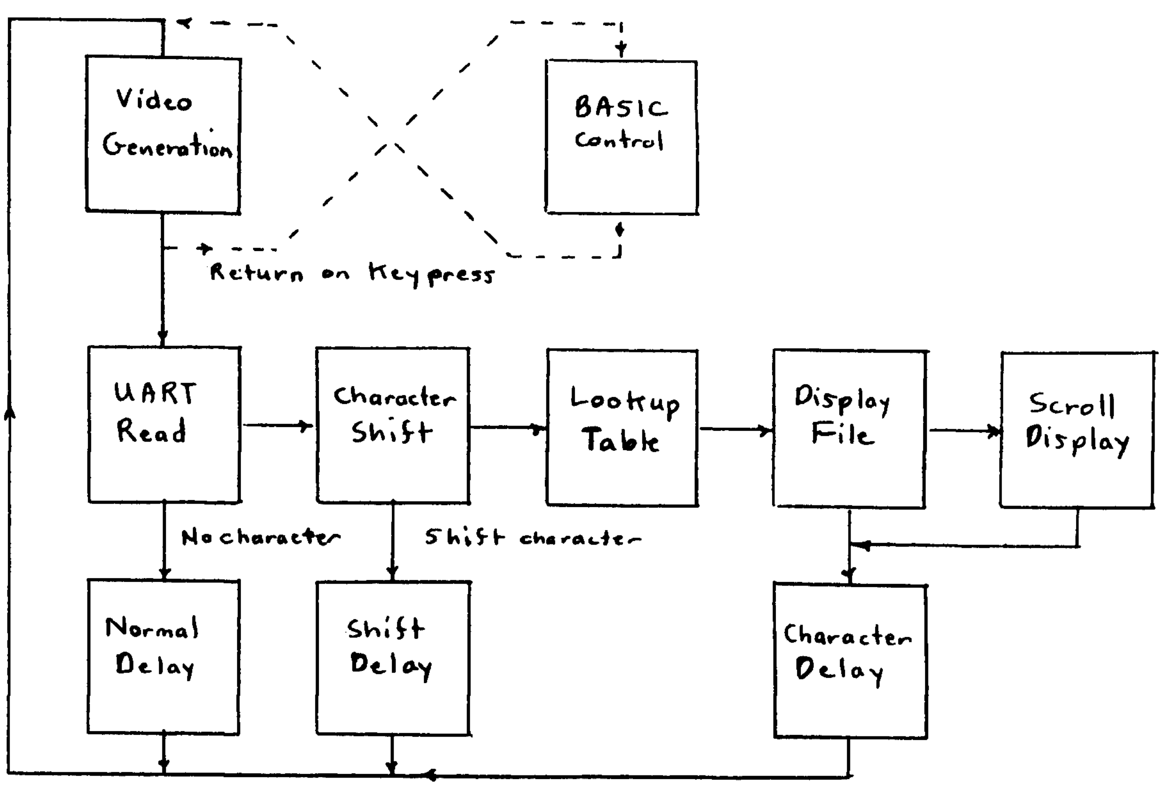


Figure 4 - Flow Diagram of RTTY Receive Program for ZX-80

(continued from previous page)

Table 1
UART Test Program (Basic)

```
1 REM (Follow with 30 Dots)
2 REM (These statements assure 1 REM is held  off
      screen)
9 REM
10 For I = 1 to 100
20 LET Q = USR (16429)
30 PRINT PEEK (16428)
40 NEXT I                          (SUB PROGRAM FOR
50 STOP                             LOADING ML)
100 INPUT J
110 PRINT J
120 INPUT K
130 PRINT K
140 FOR M = J TO K
150  GO  TO  180                   (USED  ONLY  TO
160   INPUT  C                      PRINT ML PROGRAM)
170  POKE  M,   C
180 PRINT (M-16000), PEEK (M)
190 NEXT M
200 STOP
```

Table 3
BASIC Program for Continuous RTTY Receive

```
1 REM (300 Dots)
2 REM                             (REM STATEMENTS USED
                              TO HOLD ML OFF DISPLAY)
9 REM
10 FOR I = 1 TO 32
15 FOR J = 1 TO 5
20 PRINT " ";
25 NEXT J
30 NEXT I
31 POKE 16428, 0
32 POKE 16429, 0
35 POKE 16430, (PEEK (16396) + 1)
40 POKE 16431, (PEEK (16397)
45 LET Q = USR (16466)
50 STOP
100 INPUT M
110 PRINT M,
120 INPUT N
125 PRINT N,
130 FOR K = M TO N
131 GO TO 145
135 INPUT C
140 POKE K, C
145 PRINT (K - 16000);"-";PEEK (K),
150 NEXT K
160 STOP
```

Table 2 UART Test Program
(Machine Language Program)

Decimal Address	Lower Order Address Byte	Decimal Contents	Hex Op Code	Source Statement	Comments
16428	44				Holds Data
29	45	62	3E	LD A,0	Clears A Register
30	46	0			
31	47	219	DB	IN A,(5)	Check if character is available.
32	48	5			
33	49	203	CB	BIT 7,A	Test
34	50	127	7F		
35	51	202	CA	JPZ	Return to start of routine if no character
36	52	45			
37	53	64			
38	54	219	DB	IN A,(3)	Read character into A.
39	55	3			
40	56	230	E6	AND 31	Limits value to 0-31
41	57	31			
42	58	50	32	LD 64-44,A	Loads value into buffer
43	59	44			
44	60	64			
45	61	219	DB	IN A,(4)	Resets UART
46	62	4			
47	63	201	C9	RET	Returns to BASIC

(continued from previous page)

Table 4
Machine-Language Program for Continuous RTTY Receive

Routine Name	Addresses (Decimal)
Video Generation	16433 - 16465
UART Read	16476 - 16490
Normal Delay	16466 - 16475
Character Shift	16491 - 16505
	and 16562 - 16577
Shift Delay	16694 - 16700
Lookup Table	
Routine	16506 - 16515
Table	16578 - 16642
Display File	16516 - 16561
Character Delay	16669 - 16677
Scroll	16643 - 16667
	and 16679 - 16692

Variable Locations	Address (Decimal)
Display File Column Number	16428
Display File Row Number	16429
Display File Address (2 Bytes)	16430 - 31
Lookup Table Start Address	16511

Table 5 Machine Language Program
Page 1

Decimal Address	Lower Order Add. Byte	Decimal Contents	Hex Op Codes	Source Statement	Comments
16428	44				Display Col. No.
29	45				Display Row No.
30	46				Display Location Address
31	47				
32	48	0	0	NOP	Buffer
33	49	211	D3	OUT (FF), A	Start Video Generation
34	50	255	FF		
35	51	62	3E	LD A, EC	
36	52	236	EC		
37	53	1	1	LD BC, 5-32	
38	54	32			
39	55	5			Nominally 5 Video lines
40	56	42	2A	LD HL, (400C)	
41	57	12	0C		
42	58	64	40		
43	59	203	CB	SET 7, H	
44	60	252	FC		
45	61	205	CD	CALL 01B0	Call ROM Routine
46	62	176	B0		
47	63	1	01		
48	64	62	3E	LD A, E8	
49	65	232	E8		
50	66	4	4	INC B	

(continued from previous page)

Table 5 Page 2

Decimal Address	Lower Order Add. Byte	Decimal Contents	Hex Op Codes	Source Statement	Comments
16451	67	43	2B	DEC HL	
52	68	14	E	LD C, 184	
53	69	184			Nominally 23 Blank Lines
54	70	205	CD	CALL 01B0	Call ROM Routine
55	71	176	B0		
56	72	1	01		
57	73	62	3E	LD A, FE	
58	74	254	FE		
59	75	219	DB	IN A, (FE)	
60	76	254	FE		
61	77	31	1F	RRA	Return to BASIC on Key press
62	78	208	D0	RET NC	
63	79	195	C3	JP	Jump to UART Read Routine
64	80	92			
65	81	64			
66	82	219	DB	IN A, (4)	UART Reset
67	83	4			
68	84	6	6	LD B, 200	Normal Delay
69	85	192			
70	86	16	10	DJNZ 0	
71	87	254			
72	88	195	C3	JP	Jump to Start of Video Generation
73	89	49			
74	90	64			
75	91	0		NOP	

Table 5 Page 3

Decimal Address	Lower Order Add. Byte	Decimal Contents	Hex Op Codes	Source Statement	Comments
16476	92	62	3E	LD A, 0	Start UART Read
77	93	0			
78	94	219	DB	IN A, (5)	Check UART DAU Line
79	95	5			
80	96	230	E6	AND 128	
81	97	128			
82	98	203	CB	BIT 7, A	
83	99	127	7F		
84	100	202	CA	JP Z	If no new character Jump to Normal Delay
85	101	82			
86	102	64			
87	103	219	PB	IN A, (3)	Read Data from UART
88	104	3			
89	105	230	E6	AND 31	Start Character Shift
90	106	31			
91	107	254	FE	CP 0	Test for
92	108	0			Blank
93	109	202	CA	JP Z	If Blank Jump to Normal Delay
94	110	82			
95	111	64			
96	112	254	FE	CP 27	Test for Figures
97	113	27			
98	114	202	CA	JP Z	If Figures Jump to up shift Routine
99	115	178			
16500	116	64			

(continued from previous page)

Table 5 Page 4

Decimal Address	Lower Order Add. Byte	Decimal Contents	Hex Op Code	Source Statement	Comments
16501	117	254	FE	CP 31	Test for Letters
02	118	31			
03	119	202	CA	JP Z	If Letters Jump to Downshift Routine
04	120	186			
05	121	64			
06	122	50	32	LD (nn), A	Load character value as displacement in lookup command
07	123	131			
08	124	64			
09	125	221	DD	LD IX, nn	Load index register with address of start of lookup table
10	126	33	21		
11	127	194			
12	128	64			
13	129	221	DD	LD A, (IX+d)	Load A register with character code from lookup table
14	130	126	7E		
15	131	0			
16	132	42	2A	LD HL, (64-46)	Point HL at display file address
17	133	46			
18	134	64			
19	135	119	77	LD (HL), A	Load character in display
20	136	219	DB	IN A, (4)	Reset UART
21	137	4			
22	138	35	23	INC HL	
23	139	34	22	LD (64-46), HL	Store next display file address
24	140	46			
25	141	64			

Table 5 Page 5

Decimal Address	Lower Order Add. Byte	Decimal Contents	Hex Op Code	Source Statement	Comments
16526	142	58	3A	LD A, (64-44)	Load register A with column number
27	143	44			
28	144	64			
29	145	60	3C	INC A	Increment A
30	146	254	FE	CP 32	Test for end of line
31	147	32			
32	148	194	C2	JP NZ	If not end of line Jump to character delay
33	149	29			
34	150	65			
35	151	42	2A	LD HL (64-46)	Increment display file pointer past the end of line character
36	152	46			
37	153	64			
38	154	35	23	INC HL	
39	155	34	22	LD (64-46), HL	
40	156	46			
41	157	64			
42	158	62	3E	LD A, 0	Reset column no. to zero
43	159	0			
44	160	50	32	LD (64-44), A	
45	161	44			
46	162	64			
47	163	58	3A	LD A, (64-45)	Increment row number
48	164	45			
49	165	64			
50	166	60	3C	INC A	

(continued from previous page)

Table 5 Page 6

Decimal Address	Lower Order Add. Byte	Decimal Contents	Hex Op Code	Source Statement	Comments
16551	167	50	32	LD (64-45), A	
52	168	45			
53	169	64			
54	170	254	FE	CP 4	Test for Scroll
55	171	4			
56	172	204	CC	CALL Z	Call for scroll subroutine at (65-4)
57	173	4			
58	174	65			
59	175	195			Jump to Video Generation
60	176	48			
61	177	64			
62	178	33	21	LD HL	Start of Shiftup Routine
63	179	127			
64	180	64			
65	181	54	36	LD (HL), n	Load shift address with start of Figures Lookup table
66	182	226			
67	183	195	C3	JP	
68	184	54			Jump to Shift Delay
69	185	65			
70	186	33	21	LD HL	Start of Downshift Routine
71	187	127			
72	188	64			
73	189	54	36	LD (HL), n	Load shift address with start of Letters lookup table
74	190	194			
75	191	195	C3	JP	
76	192	54			Jump to shift Delay
77	193	65			

Table 5 Page 7

Decimal Address	Lower Order Add. Byte	Decimal Cont. Sinclair Code	Baudot Ch. Value	Character	Comments
16578	194	0	0	Blank	Start of Letters Lookup Table
79		42	1	E	
80		22	2	= (Line Feed)	
81		38	3	A	
82		0	4	Space	
83		56	5	S	
84		46	6	I	
85		58	7	U	
86		24	8	< (Carriage Return)	
87		41	9	D	
88		55	10	R	
89		47	11	J	
90		51	12	N	
91		43	13	F	
92		40	14	C	
93		48	15	K	
94		57	16	T	
95		63	17	Z	
96		49	18	L	
97		60	19	W	
98		45	20	H	
99		62	21	Y	
16600		53	22	P	
01		54	23	Q	
02	218	52	24	O	

(continued from previous page)

Table 5 Page 8

Decimal Address	Lower Order Add. Byte	Decimal Cont. Sinclair Code	Baudot Char. Value	Character	Comments
16603	219	39	25	B	
04		44	26	G	
05		19	27	+	Figures -Upshift (Not displayed)
06		50	28	M	
07		61	29	X	
08		59	30	V	
09		12	31	£	Letters - -Downshift
10		0	0	Blank	Start of Figures Lookup Table
11		31	1	3	
12		22	2	= (Line Feed)	
13		18	3	–	
14		0	4	Space	
15		20	5	'	
16		36	6	8	
17		35	7	7	
18		24	8	< (Carriage Return)	
19		13	9	$	
20		32	10	4	
21		20	11	'	
22		26	12	, (Comma)	
23		14	13	:	
24		14	14	:	
25		16	15	(	
26		33	16	5	
27	243	1	17	"	

Table 5 Page 9

Decimal Address	Lower Order Add. Byte	Decimal Cont. Sinclair Code	Baudot Char. Value	Character	Comment
16628	244	17	18	)	
29	245	30	19	Z	
30	246	0	20	Blank	
31	247	34	21	6	
32	248	28	22	Ø	
33	249	29	23	1	
34	250	37	24	9	
35	251	15	25	?	
36	252	0	26	Blank	
37	253	19	27	+	Figures -Upshift
38	254	27	28	. (Period)	
39	255	21	29	/	
40	0	25	30	;	
41	1	12	31	£	Letters -Downshift
42	2	0	0	0	

Notes on Lookup Tables

1. Not all characters print, some are only diagnostic.
2. For inverse video, add 128 to these values.

(continued from previous page)

Table 5 Page 10

Decimal Address	Lower Order Add. Byte	Decimal Content	Hex Op Code	Source Statement	Comments
16643	3	0	0	NOP	Buffer
44	4	237	ED	LD DE (mm)	Start of Scroll Subroutine
45	5	91	5B		
46	6	12			Display file address
47	7	64			
48	8	19	13	INC DE	
49	9	33	21	LD HL, 33	
50	10	33			
51	11	0			
52	12	25	19	ADD HL, DE	
53	13	1	1	LD BC, 100	For 3 line scroll
54	14	100			
55	15	0			
56	16	237	ED	LDIR	Scroll command (Block transfer)
57	17	176	B0		
58	18	195	C3	JP (65-39)	Jump to 16679
59	19	39			
60	20	65			
61	21	62	3E	LD A, 0	Return from 16690
62	22	0			
63	23	18	12	LD (DE), A	Creates blank line
64	24	19	13	INC DE	
65	25	16	10	DJNZ (-6)	
66	26	250			
67	27	201	C9	RET	End Scroll Subroutine

Table 5 Page 11

Decimal Address	Lower Order Add. Byte	Decimal Content	Hex Op Code	Source Statement	Comments
16668	28	0	0	NOP	Buffer
69	29	50	32	LD (64-44), A	Start of Character Delay
70	30	44			
71	31	64			
72	32	6	6	LD B, 150	
73	33	150			
74	34	16	10	DJNZ 0	
75	35	254			
76	36	195	C3	JP	Jump to Video Generation
77	37	49			
78	38	64			
79	39	27	1B	DEC DE	Patch from Scroll Subroutine
80	40	237	ED	LD (64-46), DE	
81	41	83	53		
82	42	46			
83	43	64			
84	44	6	6	LD B, 32	
85	45	32			
86	46	33	21	LD HL, (64-45)	
87	47	45			
88	48	64			
89	49	53	35	DEC HL	
90	50	195	C3	JP	Patch to 16662
91	51	21			
92	52	65			

Data Communications

Conducted by
David W. Borden,* K8MMO

STD Packet Computing

I have written in the past of packet radio work with the STD computers that our club obtained from Ma Bell surplus (again, no further ones available). These Z80 computers use the Zilog SIO on a board made by MOSTEK. Jon Bloom, KE3Z, took one of these computers and has constructed a card which allows them to enter the DPLL world of the Vancouver and TAPR Terminal Node Controllers. Some have written for schematics for this work and they will be sent soon. Jon has completed the transmit side and has been repeating with the box successfully. I intend to supply receive schematic, transmit schematic and hex dump of the state machines and CWID PROMs. This is an experimental thing. If you want to try it, Zilog SIOs are inexpensive, but add up the cost before proceeding. We obtained these computers very inexpensively. If you did all this from scratch, it might be better to go with the Vancouver TNC or wait for TAPR or AMRAD PAD.

Level-3 Considerations

This month, we are all thinking about true networking. What we have now is not a network, but rather is about 250 separate nodes that can packet to each other if a medium exists. Possibly AMSAT OSCAR 10 will give us a real good medium to use. A network of terrestial nodes however is another goal. That requires that we think about new issues.

First, using the simple approach, what are the problems? Assume that K8MMO, Dave is in Washington DC and KA6M, Hank is in San Francisco. How do I talk to Hank? I am in an area of about a dozen packeteers in rf range of each other. Hank has about 40 packeteers in range of him. How do I connect up with Hank. Specifically:

a. How do I know where Hank is? Actually, today he may not be in San Francisco, he may be in Los Angeles. We can simplify things by assuming that if he is away, a mailbox system on the San Francisco local area network will accept traffic for later pickup by Hank.

b. Do I establish virtual circuits all across the country to Hank? Example, Washington-Kansas City and Kansas City-San Francisco?

c. Assume that some extra software is required for this networking, where is it? In the Terminal Node Controller or in some smarter (larger memory) computer?

d. Do I allow implicit route selection (the computer network controllers choose the route) or explicit route selection (the users choose the route)?

e. What happens when a node goes down in the middle of a connection? How do we recover?

To begin work on the standard we must first review the layers:

Level 1, or the physical control layer of the ISO (International Organization for Standardization) model is: "the mechanical, electrical, functional and procedural characteristics to activate, maintain and deactivate the physical link between the DTE and the DCE." Paul Rinaldo, W4RI has drafted an AX.25 Level 1 definition which has been circulated for comment. A revision is to be prepared in the near future.

Level 2, or the link control layer of the ISO model is: "the physical link control procedure" which specifies the headers and trailers of blocks of data sent. Higher-level data link control (HDLC) is the line-control procedure used by AX.25 as various hardware chips implement it (Intel 8273, Western Digital 1933, Zilog 8530, SIO). The latest description of AX.25 level 2 in a paper by Terry Fox, WB4JFI, was presented at the

(continued on next page)

*Rte 2, Box 233B, Sterling, VA 22170

(continued from previous page)

Table 5 Page 12

Decimal Address	Low-Order Add. Byte	Decimal Contents	Hex Op Code	Source Statement	Comments
16693	53	0	0	NOP	Buffer
94	54	6	6	LD B, 175	Start Shift Delay
95	55	175			
96	56	16	13	DJNZ 0	
97	57	254			
98	58	219	DB	IN A, (4)	Reset UART
99	59	4			
16700	60	195	C3	JP	Jump to Video Generation
01	61	49			
02	62	64			
03	63	0	0	NOP	Buffer

Data Communications
(continued from previous page)

"Second ARRL Amateur Radio Computer Networking Conference," proceedings of which are available from ARRL Hq for $9.00 post paid.

Level 3, or the network control of the ISO model is: "the virtual circuits, logical circuits or logical links" that connect network nodes. Currently VADCG and TAPR TNC software implements a connection and disconnection between two nodes. Amateur Radio call signs are used to identify the nodes. The AX.25 protocol is the defacto standard that defines how to create and accept the required data and control packets. We have chosen to divide Level 3 into two sub-layers, the local network sub-layer, which is directly above the link level. This sub-layer is what is already coded and working AX.25 in both TAPR and VADCG TNC boards. Directly above the local network sub-layer is the internetwork sub-layer and that is what needs definition and code.

Layer 4 of the ISO model, the transport layer, may or may not be required and will not be discussed here.

In the study of Level 3, the internetwork sub-layer, we have chosen to implement a connection type network (vice datagrams) and will follow the CCITT (International Telegraph and Telephone Consultative Committee) X.25 Level 3 protocol having to do with virtual circuits. Thus, we have narrowed the field a little, but much work needs to be done.

Eric Scace, K3NA, has been involved deeply in X.25 and is advising us on what X.25 is all about. One cannot deduce everything about X.25 by reading the CCITT documents. A guide is required, and Eric is our translator. He tells us what they meant when they wrote whatever part of the protocol that we are looking at. Eric has identified the following points of discussion that require decision concerning Level 3, X.25:

a. Logical channel numbers are used in each active virtual circuit. How should the range of LCNs be handled?

b. Should we employ diagnostic packets?

c. Are abbreviated or other forms of the address field allowed? Currently the address fields employ amateur call signs and device identifiers. AX.25 address procedures require a separate appendix.

d. The terms DTE and DCE as they pertain to AX.25 use must be clearly defined.

e. The set of packet sequence numbers allowed to cross the DTE-DCE interface is called the window. Should we force the window size to two and keep it simple?

f. Study of the 63 octet (byte) facility field is required for route selection information.

g. Should internetworking with public data nets be allowed? If so, how about third party agreement enforcement? What about restrictions against business use?

h. Should hunt groups be allowed for reaching an authorized control point without caring which one? How does all this relate to group addressing?

We shall be studying all these questions and more and producing a draft standard for packeteers to examine. - K8MMO.

Intercontinental Packet-Radio Tests Successful

At approximately 23:00 UTC on 27 May, a successful digital packet-radio QSO was held between Tom Clark, W3IWI in Maryland and Ian Ashley, ZL1AOX near Auckland, New Zealand on 28 MHz. This 13850-km path represents the longest distance yet bridged using packet radio. Both stations were using the Terminal Node Controller (TNC) developed by the Tucson Amateur Packet Radio group in conjunction with home built S-100 computers. The QSO used frequency-shift keying (fsk) techniques at a data rate of 1200 bauds.

Although propogation conditions were marginal, data exchanges consisted of "beacon" transmissions and two-way connection acknowledgements using the amateur AX.25 HDLC protocol; with this protocol messages are sent repeatedly until they are successfully acknowledged by the other station.

Also participating in the test was Bob Diersing, N5AHD in Corpus Christi, Texas; W3IWI and N5AHD have held several successful packet radio QSO's. N5AHD monitored the beacon and text transmissions from both W3IWI and ZL1AOX.

W3IWI and Vernon Riportella, WA2LQQ, took advantage of the Memorial Day holiday to try again with ZL1AOX. This time we tried running at 600 bauds. W3IWI connected but had local QRM. WA2LQQ's bigger signal won out, and Rip and Ian went at it for an hour. In Washington, DC, K1HTV and KA1GD were SWLing and copied packets from Ian.

All three stations involved are affiliated with AMSAT and have been experimenting with high-speed digital transmissions in anticipation of conducting similar activities with AMSAT-OSCAR 10 and with the AMSAT PACSAT packet-radio satellite now being designed. - W3IWI.

QEX: The ARRL Experimenters' Exchange is published by the

American Radio Relay League
225 Main Street
Newington, CT 06111 USA
telephone 203-666-1541

The purposes of QEX are to:

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, and

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

Subscriptions are available to ARRL members and non-members at the rates shown on the QEX Subscription Order Card inside this issue.

All correspondence concerning QEX should be addressed to the American Radio Relay League, 225 Main Street, Newington, CT USA 06111. Envelopes containing manuscripts and correspondence for publication in QEX should be marked: Editor, QEX. QEX subscription orders, changes of address, and reports of missing or damaged copies may be marked: QEX Circulation. Members are asked to include their membership control number or a label from their QST wrapper when applying.

Both theoretical and practical technical articles are welcomed. Manuscripts should be typed and double spaced. Please use the standard ARRL abbreviations found in the latest December issue of QST. Authors should supply their own artwork using black ink on white paper. When essential to the article, photographs may be included. 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 it will be when printed 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 QEX are included for your information, not advertising, nor is any endorsement implied. The information is believed to be correct, but readers are cautioned to verify availability of the product before sending money to the vendor. Material may be excerpted from QEX without prior permission provided that the original contributor is credited, and QEX is identified as the source.

QEX: The ARRL
Experimenters' Exchange
American Radio Relay League
225 Main Street
Newington, CT USA 061111

Nonprofit Organization
U.S. Postage
PAID
Hartford, Conn.
Permit No. 2929