TEXAS INSTRUMENTS

Programming The TMS9902 In The T.I. 99/4A Part II

by Randy Holcomb

In the last issue we covered the Input CRU bit assignments of the TMS 9902 Asynchronous Communications Controller Chip. In this part we will cover the Output CRU Bit assignments, programming examples and the TMS 9902 in the TI 99/4A environment.

CRU Output Bit Assignments (Also see Table 2.) Described below are the Output CRU Bit assignments and their function:

Bit 31: Reset. Setting this bit high causes the 9902 to reset the entire chip. All interrupts are disabled, RTS is set high, all register flags (LDCTRL, LDIR, LRDR, LXDR) are set high and the BREAK flag is reset. No operation to the TMS 9902 should be performed for 11 clock cycles after setting this bit.

Bits 30-22: Not used.

Bit 21: Data Set Change Interrupt Enable. Setting this bit causes the 9902 to generate an interrupt when input bit 29 (Data Set Status Change) is set. Resetting this bit disables the DSCH interrupt. Writing any value to this bit will reset input bit 29.

Bit 20: Timer Interrupt Enable. Setting this bit causes the 9902 to generate an interrupt when TIMELP (Input bit 25) is set. Resetting this bit disables the TIMELP interrupts. Writing any value to this bit will reset input bit 25.

Continued on Page 115.

Address S0	S1	S2	S3	S4	Base 10	Name	Description
1	1	1	1	1	31	RESET	Reset
					30-22	-	Not used
1	0	1	0	1	21	DSCENB	Data Set Status Change Interrupt enable
1	0	1	0	0	20	TIMENB	Timer Interrupt Enable
1	0	0	1	1	19	XBIENB	Transmitter Interrupt Enable
1	0	0	1	0	18	RIENB	Receiver Interrupt Enable
1	0	0	0	1	17	BRKON	Break On
1	0	0	0	0	16	RTSON	Request To Send On
0	1	1	1	1	15	TSTMD	Test Mode
0	1	1	1	0	14	LDCTRL	Load Control Register
0	1	1	0	1	13	LDIR	Load Interval Register
0	1	1	0	0	12	LRDR	Load Receiver Data Rate Register
0	1	0	1	1	11	LXDR	Load Transmit Data Rate Register
					10-0		Control, Interval, Receive/Transmit data Rate, and Transmit Buffer registers

Table 2 - THS 9902 ACC Output CRU Bit Assignments

TMS9902 Continued From Page 107

Bit 19: Transmit Buffer Interrupt Enable. Setting this bit causes the 9902 to generate an interrupt when XBRE (input bit 22) is set. Resetting this bit disables XBRE interrupts. This bit does not affect the current value of XBRE.

Bit 18: Receiver Interrupt Enable. Setting this bit causes the 9902 to generate an interrupt when RBRL (input bit 21) is set. Resetting this bit disables RBRL interrupts. Writing any value to this bit causes RBRL to reset.

Bit 17: Break On. Setting this bit causes the XOUT line to go to logic zero whenever the transmitter is active and XBR and XSR (both transmitter buffer and shift registers) are empty and inhibits loading characters into the transmit buffer register. Resetting this bit causes the transmitter to return to normal operation.

Bit 16: RTS On. Setting this bit causes the RTS handshake signal to be active (low.) Resetting this bit causes RTS to go high after XSR and SBR are empty and BRKON is reset. (RTS will not go inactive until a character has completed transmission.)

Bit 15: Test Mode. Setting

Continued On Page 116

COMPUTER SHOPPER DISPLAY ADVERTISING & CLASSIFIED DEADLINE IS THE 2ND MONDAY OF EACH MONTH

TMS9902 Continued From Page 115

this bit causes RTS to be internally connected to CTS, XOUT to RIN, DSR held internally low, and the interval timer to operate at 32 times its normal rate. This is useful for debugging programs that are transmitting and receiving data and for testing to insure that the internal 9902 functions are working properly. Resetting this bit causes the 9902 to return to normal operating mode.

Bits 14-11: Register Load Control Flags. These bits are described below:

Bit 14: Load Control Register. This bit when set (either explicitly or after the RESET bit is set) causes the control register in bits 0-7 to be loaded into the 9902. Resetting this bit (or upon reception of the last control register bit) causes the control register to be inhibited from loading.

Bit 13: Load Interval Register. Setting this bit causes the next 8 bits (bits 0-7) to be loaded into the 9902's interval register. The Bit is reset when the last data bit is loaded or a logic zero is written to LDIR. To load this register, LDCTRL must be reset (See Table 3.)

LDCTRL	LDIR	LRDR	LXDR	Register Enabled
1	x	x	x	Control Register
0	1	x	x	Interval Register
0	0	$	x	Receive Data Rate Register*
0	0	x	1	Transmit Data rate Register*
0	0	0	0	Transmit Buffer register

Table 3 - TMS 9902 ACC Register Load Selection

Bit 12: Load Receive Data Rate Register. Setting this bit causes the next 11 bits (0-10) to be loaded into the 9902's Receive data rate register. The bit is reset when the last data bit is loaded or a logic zero is written to LRDR. To load this register LDCTRL and LDIR must be zero.

Bit 11: Load Transmit Data Rate Register. Setting this bit causes the next 11 bits (0-10) to be loaded into the 9902's Transmit data rate register. The bit is reset when the last data bit is loaded or a logic zero is written to LXDR. To load this register LDCTRL and LDIR must be zero.

Note: If both the transmit and receive data rates are the same both registers can be loaded simultaneously by setting both LRDR and LXDR.

Bits 10 thru 0: Data. Depending on the control bits set data rate registers, the interval register, the control register or data to be transmitted are written to these bits.

The Control Register. The control register is used to define the character size, the parity of the word, the number of stop bits and the operating frequency of the 9902. Table 4 outlines the bit positions and their settings. Of importance is bit 3: for the /4A environment this bit should always be zero. The PENB bit determines if parity checking is done, if set parity checking is done; if reset no parity checking is done. The PODD bit determines if the

Continued On Page 118

TMS9902 Continued From Page 116

9902 will check for/generate odd parity; if set odd parity is processed; if reset even parity is processed.

The Data Rate Registers. Both receive and transmit registers share the same format. The data rate registers are basically 2 programmable counters that divide the system clock depending on the bit settings in the counters to derive the data rate. The first counter is a divide by eight counter that is active if bit 10 of the data rate register is loaded. Since the internal 9902 clock frequency is 1 MHz, setting this bit causes its output to be 125KHz, while resetting this bit passes the system clock right thru. The output of this first counter is fed into a 10-bit counter and is further divided down by a value from 0 to 1023. The output from this counter is actually twice the data rate (interval for half the bit period.) So this output is divided by 2 to get the actual data rate. Since this formula is not exactly the easiest way to get the baud rate, here's a list of the more popular speeds and their register values for your use:

Speed	Hex Value
110 bps	>400 + >238
300 bps	>400 + >0D0
600 bps	>341
1200	bps>>1A1
2400	bps>0D0
4800	bps>068
9600	bps>034

(Note the >400 is the divide by 8 counter enable. If you need an oddball baud rate, the formula is below:

F = Eint (1 MHz)
bps 2 * (D9-D0) * (8 D10)

Address	Name	Description
7	SBS1	Stop Bit Selection. (SBS1=0,SBS2=0: 1 1/2 bits;
6	SBS2	SBS1=0,SBS2=1: 2 bits; SBS1=1,SBS2=x:1 bit)
5	PENB	Parity Enable
4	PODD	Odd Parity Select
3	CLK4M	ø Input Divide Select
2	-	Not used
1	RCL1	Character Length Select (RCL1=0,RCL0=0:5 bits;
0	RCL0	RCL1=0,RCL0=1:6 bits,RCL1=1,RCL0=0:7 bits, RCL1=1,RCL1=1:8 bits

Table 4 - TMS 9902 ACC Control Regster Format

where D9-D0 is from 0 to 1023 and D10 is the divide by eight counter.)

The Interval Register. The Interval register is a 8-bit counter which is decremented every 64 clock cycles. When the interval counter reaches zero, the appropriate interval register flags are set and if interval interrupts have been enables the 9902 will generate an interrupt. Taking into account the register size the interval counter has resolution from 64 to 16,320 microseconds in 64 microsecond increments.

Programming The TMS9902. This seems like an awful lot of stuff to swallow in order to program the 9902 but in reality a lot of the nasty things are done for you by the chip, so now lets look at some code examples, but first we need to know something about the way the 9902 is addressed in the /4A environment. Most of you who have gotten the disassembly listing of the RS232 DSR know that the DSR code is at CRU address >1300. The 9902 themselves are at +>40 for the first 9902 and the second is at +>80, so the values used for 9902 accesses are >1340 and >1380. If you have the additional RS232 card its CRU offset is >1500 and the 9902s are also +>40 and >+80. Its suggested that when you code your routines you have a subroutine which sets up the base address to turn the DSR on and then adjusts R12 to access the propet 9902.

Interrupts in the RS232 DSR. Tracing thru the disassembly listing reveals that during an interrupt request from the RS232 DSR it determines the device which fired the interrupt and determines if a character was received. If a character was not received it RESETS the 9902 - so be careful how you program the 9902 in the /4A environment. As a side benefit tho, if you use interrupts the DSR provides you with a mechanism of getting incoming characters and buffering them. Carefully trace the interrupt code and you'll find that >8300 contains a VDP address and >8302 and >8304 are pointers that are manipulated to place characters into a queue in VDR RAM.

Example. To give you an idea of what goes on with the 9902, Assembly source code is being provided that shows how the 9902 is initialized and how characters are transmitted and received.

The program is the infamous T99 dumb terminal emulator I wrote over a year ago that uses first RS232 port for a modem running at 300 baud, 1 stop bit and even parity with a printer attached off of port 2 running at 9600 bps at odd parity and 1 stop bit. The printer is turned on by receipt of a control R and is shut off by receipt of a control T. This code also shows you some of the things that are involved in writing a terminal emulator and can serve as a basis for writing more powerful emulators.

Summary. There is still a lot of things that we didn't cover in regards to the 9902 but with the information we have provided here you should be in a good position to do some exploring on your own; such as using one or two of the control lines to hook up with modems that provide auto-answer/auto-dial capabilities and write an assembly program to get you those features. The rest is up to you and your imagination!

If you would like the T99 dumb terminal emulator that goes along with this article, please send a self addressed envelope with two twenty cent stamps to:

COMPUTER SHOPPER
c/o Theda Brinckerhoff
P.O. Box F
Titusville, FL 32781 ●

See page 75 for detailed ad

Linking TI99/4A To IBM PC

by Michael A. Covington

This article tells how to store TI-99 programs on an IBM PC's disk drives by linking the computers through their RS-232 ports. The program given here should run with little or no modification on other MS-DOS systems (such as the TI Professional Computer) and is easily adapted to other kinds of computers.

If you have both a TI-99 and a more elaborate, expensive computer such as an IBM PC or TI PC, you may have wondered whether there was any way for the TI-99 to use the larger machine's disk drives to store and retrieve its programs instantly rather than grinding them out slowly onto cassettes. That way, you would get the benefits of disk storage without having to pay hundreds of dollars to add a disk drive to the TI-99 itself.

It turns out that such a linkup is perfectly possible. All you have to do is connect the two machines through RS-232 serial ports and program the larger machine so that it can record data received from the TI-99 and then, later on, feed the recorded data back to the TI-99 in the proper format. What follows is an account of how I did this with an IBM Personal Computer. The information given here should enable you to link the TI-99 to other kinds of computers in a similar way.

Making The Connection

The name RS-232 designates the standard format for serial data transmission between one computer and another. RS-232 ports have distinctive 25-pin connectors, and the function of each of the 25 conductors is rigidly specified.

Naturally, if two pieces of equipment are to be connected together, one unit's input line must be the other unit's output, and vice versa. For this reason, the RS-232 standard allows for two types of equipment: data communication equipment (DCE), which uses pin 2 for input and pin 3 for output, and data terminal equipment (DTE), in which the roles of pins 2 and 3 (and also of pins 6 and 20, the equipment-ready signals) are reversed. Ordinary RS-232 cables are designed to connect DTE to DCE. (Most computer terminals are wired as DTE; most modems, as DCE.)

The RS-232 port on the IBM PC is wired as DTE, and the RS-232 card for the TI-99 Peripheral Expansion System is DCE. They can be connected with an ordinary cable, one that connects each pin on one end to the pin with the same number on the other end. If you're using some other type of RS-232 port with the TI-99, check its specifications; if it's DTE, use a "modem eliminator" cable--one that swaps the appropriate conductors--rather than a straight RS-232 cable.

Data transmission

Before dealing with disk storage of programs, I decided to implement a considerably simpler function, that of linking the TI-99 to the IBM's printer. This is dealt with in lines 320-530 of the program. All that the IBM has to do is listen for characters coming in through the communications port and transmit them to the printer as they come in.

When no special options are given, the TI-99 transmits in 7-bit format, with odd parity, at 300 baud. Line 340 of the program opens the IBM's communications port to match it. I decided to leave the transmission speed at 300 baud in order to keep the commands on the TI-99 as simple as possible, although my printer (an Epson MX-80) will print somewhat faster. If the full graphics capabilities of the printer were

Continued On Page 120

Linking TI99 From

Page 119

to be used, it would be necessary to use 8-bit transmission on both ends.

Since the TI-99 thinks it is connected directly to a printer, it transmits a carriage return and a line feed at the end of each line. The IBM would ordinarily add a line feed of its own after every carriage return; to suppress this, specify a line length of 255 characters (line 330).

The main loop for this part of the program consists of lines 480 to 530. On each cycle, the IBM first looks to see if the operator has pressed a key, which would be the signal to end this routine. If not, it tests whether there is anything in the communications buffer and, if so, sends it to the printer. While this is going on, the user of the TI can treat device name "RS232" as if it did in fact refer to a real printer; a program can be listed with the TI-99 command

LIST "RS232"

and a statement such as

OPEN #1:"RS232"

enables a TI-99 program to write on the IBM printer.

Transferring programs

Transferring a TI-99 program to the IBM in such a way that it can be transmitted back and reloaded into the memory of the TI-99 is more complex. The problem is that TI-99 programs are always stored in tokenized form--that is, each BASIC reserved word is replaced by a one-byte code, thus saving a considerable amount of space. The LIST "RS232" command transmits a program to the IBM in de-tokenized form, with reserved words spelled out in the usual way, suitable for printing. But there is no way to transmit a de-tokenized program back to the TI; the TI accepts un-tokenized input only from its keyboard, and keyboard functions cannot be reassigned to any other device.

Fortunately, there is another way. The engineers at TI provided a way to transmit programs from one TI-99 to another in tokenized form by connecting the machines' RS-232 ports together (swapping input and output wires, of course) and typing

SAVE RS232

on one TI-99 and

OLD RS232

on the other. (Unlike LIST, SAVE and OLD do not require quotation marks around the device name.)

What is needed, then, is a way to make the IBM simulate one of the TI-99's in such an exchange. Unfortunately, the documentation for the TI-99 nowhere states exactly how this type of program transfer is carried out. However, after considerable experimentation I've been able to put together the following rough picture:

(1) SAVE RS232 is typed on the machine that is to transmit.

(2) OLD RS232 is typed on the machine that is to receive. This causes it to transmit a CHR$(22) to the sender.

(3) The sender transmits 4 bytes (probably a checksum).

(4) The receiver transmits CHR$(6).

(5) The sender transmits 258 bytes (presumably a 256-character block plus a checksum).

(6) Steps 4 and 5 are repeated until no more data is received. (There is surely an end-of-file code, but I have not been able to decipher it.)

Lines 570-1110 make the IBM simulate the TI-99 that is receiving the incoming program on disk. Unlike most files, a tokenized TI-99 program is not organized into lines that end with carriage return and line feed; hence, the incoming data cannot be stored in an ordinary ASCII sequential file. Instead, I chose to use a file of fixed-length 45-byte records, each consisting of a 43-byte data field (one sixth of a 258-byte block) and a 2-byte length field, indicating how many of the 43 bytes are used and thus making it possible to accommodate the 4-byte checksum at the beginning of the file and the incomplete block at the end. Reception continues until the communication buffer becomes empty and remains so on 150 successive examinations (line 890). Then a record with a zero length field is written to signal the end of the file.

In lines 1180-1560, the IBM simulates a transmitting TI-99. It reads a file in the format just described and parcels it out to the receiver, waiting for a go-ahead character before transmitting each block.

Since high speed is desirable here, and the dependence on go-ahead characters ensures that neither machine will overload the other by transmitting data too fast, I chose to do all the program transmission at 9600 baud. This means that the TI-99 operator must refer to the RS-232 port as "RS232.BA=9600" rather than just "RS232".

Performance

TI-LINK is much more convenient to use than the cassette recorder that it replaces, if for no other reason than that many files are stored on a single disk and there is no need to shuffle through a box of tapes or fumble with the rewind button and tape counter. Since TI-99 programs tend to be small, a 320K IBM diskette will hold dozens of them.

Storing a TI program on the IBM disk takes about the same amount of time as recording it on a cassette; apparently, the TI-99 outputs tokenized programs at a fixed rate. Going in the other way, however, TI-LINK offers a dramatic speed advantage. In one test, a program that took 32 seconds to record on cassette, and 31.4 seconds to record on disk, took only 4.2 seconds to load back into the TI-99 from the same disk. Since the main usefulness of TI-LINK is in maintaining a program library, the high transfer speed contributes greatly to its utility.

The main limitation of this version of TI-LINK is that it does not check for error conditions. Obviously, I have not been able to decipher and use the TI-99's checksums; hence, a transmission error in a program can only be detected by the TI-99. More importantly, TI-LINK does not trap error conditions on the IBM PC; any attempt to read a nonexistent file, write on a disk with no space available, or even access a printer that has been switched off line will bring the program to a sudden stop. A more practical program of this type should obviously deal with errors in a more user-friendly way. ●

```
10 ' TI-LINK     Michael A. Covington  1984
20 ' Data interchange program for TI-99/4A and IBM PC
30 ' (Should run with virtually no alteration on most MS-DOS systems;
40 ' easily adaptable to other Microsoft BASIC environments.)
50 '
60  CLS : KEY OFF : PRINT : PRINT : PRINT : PRINT : PRINT
70  CLOSE #1 : CLOSE #2 : CLOSE #3 ' if they are open
80  PRINT, : COLOR 0,7
90  PRINT "                   TI-LINK 1.0               " : COLOR 7,0
100 PRINT
110 PRINT,"Data interchange program for IBM PC and TI-99" : PRINT
120 PRINT,"            Michael A. Covington  1984"
130 PRINT
140 PRINT,"The following functions are supported:"
150 PRINT
160 PRINT,"1.   Link TI-99 to host printer"
170 PRINT,"2.   Save a TI-99 program on host diskette"
180 PRINT,"3.   Load a TI-99 program from host diskette"
190 PRINT,"4.   Display list of files on a disk"
200 PRINT,"5.   End this program"
210 PRINT
220 PRINT,"Your choice";
230 INPUT CHOICE
240 IF CHOICE >= 1 AND CHOICE <= 5 THEN 270
250 PRINT,"A number between 1 and 5, please."
260 PRINT,  : GOTO 230
270 ON CHOICE GOSUB 290,540,1120,1570,1670
280 GOTO 60
290 '
300 '      Link TI-99 to host printer
310 '
320 CLS : PRINT : PRINT : PRINT : PRINT
330 OPEN "LPT1:" AS #1 : WIDTH #1,255 ' (done to suppress line feeds)
340 OPEN "COM1:300,O" AS #2   ' O for ODD (not 0 for Zero)!
350 PRINT,  "The TI-99 is now connected to the printer of"
360 PRINT,  "the host system." : PRINT
370 PRINT,  "To get a listing of your TI-99 program, type:"
380 PRINT : PRINT, : COLOR 0,7
390 PRINT   " > LIST ";CHR$(34);"RS232";CHR$(34);"  "
400 COLOR 7,0 : PRINT
410 PRINT,  "on the TI-99 console." : PRINT
420 PRINT,  "To send TI-99 program output to the printer,"
430 PRINT,  "use the device name  ";CHR$(34);"RS232";CHR$(34);" ."
440 PRINT
450 PRINT,  "To end this routine, press any key on the host"
460 PRINT,  "computer's console."
470 ' Main loop
480 IF INKEY$<>"" THEN PRINT #1,"" : RETURN
490 '  (Empty print buffer and exit when a key is pressed)
500 IF EOF(2) THEN 480          ' is there anything coming in?
510 INDATA$=INPUT$(LOC(2),#2)  ' get contents of communications buffer
520 PRINT #1,INDATA$;
530 GOTO 480
540 '
550 '      Save TI-99 program on host disk
560 '
570 CLS : PRINT : PRINT : PRINT : PRINT
580 PRINT,  "            SAVE TI-99 PROGRAM TO HOST DISK":PRINT:PRINT
590 OPEN "COM1:9600,N,8,1" AS #1   ' no parity, 8 bits, 1 stop bit
600 PRINT,  "Enter name of disk file to be created: ";
610 LINE INPUT FILENAME$
620 OPEN FILENAME$ AS #2 LEN=45
630 FIELD #2, 2 AS FIELDLENGTH$, 43 AS FILERECORD$
640 '
650 PRINT
660 PRINT,  "To start transmission, type  ";
670 COLOR 0,7 : PRINT " >SAVE RS232.BA=9600 "
680 COLOR 7,0
690 PRINT,  "on the TI-99, then press any key here." : PRINT
700 IF INKEY$="" THEN 700
710 PRINT,  "The number of blocks remaining to be trans-"
720 PRINT,  "mitted is shown at the top of the TI-99 screen."
730 PRINT
740 '
750 PRINT #1,CHR$(22);           ' send "Ready-to-receive"
760 INDATA$=""
770 FOR I=1 TO 4                 ' receive the 4-byte header record
780   IF EOF(1) THEN 780
790   A$=INPUT$(1,#1)            ' get one character
800   INDATA$=INDATA$+A$
810 NEXT
820 GOSUB 1070                   ' write it
830 '
840 PRINT #1,CHR$(6);            ' send "ready-to-receive-a-block"
850 FOR I=1 TO 6                 ' receive 258 bytes in all (6 * 43)
860  INDATA$=""
870   FOR J=1 TO 43              ' handle 43 bytes at a time
880     K=0                        ' count eofs to check if permanent condition
890     IF K>150  THEN 980         ' if so, transmission must be over
900     K=K+1
910     IF EOF(1) THEN 890
920     INDATA$=INDATA$+INPUT$(1,#1)    ' get one character
930   NEXT J
940  GOSUB 1070  ' write it
950 NEXT I
960 GOTO 840
970 '
980 PRINT #1,CHR$(6)             ' send one last prompt to TI
990 GOSUB 1070                   ' write current record
1000 INDATA$="" : GOSUB 1070  ' write a null record to indicate end
1010 PRINT : PRINT, "                 Transmission complete."
1020 PRINT : PRINT, "               Press any key to continue..."
1030 IF INKEY$="" THEN 1030 ELSE RETURN
1040 '
1050 '
1060 ''' Subroutine --
1070 ''' format data and write to fixed-length file record
1080 LSET FIELDLENGTH$ = MKI$(LEN(INDATA$))   ' encode integer in 2 bytes
1090 LSET FILERECORD$  = INDATA$
1100 PUT #2
1110 RETURN
1120 '
1130 '   Load a TI-99 program from host diskette
1140 '
1150 '   The opposite of lines 570-1030.
1160 '   A zero length field signals end of file.
1170 '
1180 CLS : PRINT : PRINT : PRINT : PRINT
1190 PRINT,  "       LOAD TI-99 PROGRAM FROM HOST DISK":PRINT:PRINT
1200 ' Open line to TI-99
1210 OPEN "COM1:9600,N,8,1" AS #1    ' no parity, 8 bits, 1 stop bit
1220 ' Open disk file
1230 PRINT,  "Name of disk file to be read: ";
1240 LINE INPUT FILENAME$
1250 OPEN FILENAME$ AS #2 LEN=45
1260 FIELD #2, 2 AS FIELDLENGTH$, 43 AS FILERECORD$
1270 '
1280 PRINT
1290 PRINT,  "To start transmission, type  ";

1300 COLOR 0,7 : PRINT " >OLD RS232.BA=9600 "
1310 COLOR 7,0
1320 PRINT,  "on the console of the TI-99." : PRINT
1330 PRINT,  "The number of blocks remaining to be trans-"
1340 PRINT,  "mitted is shown at the top of the TI-99 screen."
1350 PRINT
1360 '
1370 IF EOF(1) THEN 1370    ' wait for go-ahead
1380 A$ = INPUT$(1,#1)
1390 IF A$<>CHR$(22) THEN PRINT,"Wrong signal received from TI-99" : STOP
1400 '
1410 GET #2
1420 IF FIELDLENGTH$<>MKI$(4) THEN PRINT,"Not a TI program file" : STOP
1430 PRINT #1,LEFT$(FILERECORD$,4); ' transmit header record to TI
1440 '
1450 IF EOF(1) THEN 1450
1460 A$ = INPUT$(1,#1)        ' wait for go-ahead (chr$(6))

1470 FOR I = 1 TO 6          ' transmit 6  43-character records
1480   GET #2
1490   IF FIELDLENGTH$=MKI$(0) THEN 1540    ' end of file?
1500   PRINT #1,LEFT$(FILERECORD$,CVI(FIELDLENGTH$));   ' semicolon vital!
1510 NEXT
1520 GOTO 1450
1530 '
1540 PRINT : PRINT,  "                 Transmission complete."
1550 PRINT : PRINT,  "              Press any key to continue..."
1560 IF INKEY$="" THEN 1560 ELSE RETURN
1570 '
1580 '      Display list of files
1590 '
1600 CLS : FOR I=1 TO 5 : PRINT : NEXT
1610 PRINT,,"Which disk drive";
1620 INPUT R$
1630 PRINT : PRINT : PRINT : FILES LEFT$(R$,1)+":*.*"
1640 PRINT : PRINT : PRINT
1650 PRINT,"      Press any key to return to main menu..."
1660 IF INKEY$<>"" THEN RETURN ELSE GOTO 1660
1670 '
1680 '      End program
1690 '
1700 CLS
1710 KEY ON
1720 END
```

Randy's Ravings

by Randy Holcomb

This month I was going to rip into the MBX system for an in-depth look but some new software has just come my way that needs to be looked at, so I'm pushing the MBX article back another month while I probe the disassembled ROM listings of the Championship Baseball cartridge and I should have something that will be worth the wait. Now, on to the new software.

99/4 AUTO SPELL-CHECK
Spelling Checker for TI-Writer
Dragonslayer American Software
2606 Ponderosa Drive
Omaha, NE 68123
$49.95 + $3 Shipping/Handling

Rumors of a spelling checker program for TI-Writer were floating on the TI Forum on CompuServe for the past several months, and lo and behold, these rumors have come true! This piece of software comes on 2 disks with a 20,000 word vocabulary and facilities to create your own external dictionaries. The program is loaded by selecting menu option 3 from the TI-Writer screen (See, there IS something that uses this option after all!) or via menu option 5 in the Editor/Assembler main menu. It will display the title screen, then ask you to hit enter and the software then asks you to enter the name of the file you wish to process. The spelling checker software will handle any DISPLAY/VARIABLE 80 file with one noticeable restriction: no ESCape characters are allowed in the text file. The manual suggests that you replace all occurrences of ESCape by using "!!". Once the file name has been entered the checker will ask you to place disk A in drive 1 and hit enter. The software will begin scanning the text file (which is in memory) with the first dictionary file (DICT1) and begins stacking words which are not in its resident dictionary. After it processes the first dictionary file, the software asks you to install the second dictionary (DICT2) in drive 1 and hit enter. The second part of the dictionary file will then be processed against the text file and like the first part will stack words not recognized. The last phase of the scanning involves asking for a user-supplied dictionary which contains a custom vocabulary which you can create and maintain. If no user dictionary is supplied the software moves to the word review list, otherwise you supply the name of the user dictionary and continue scanning. The software is set up so a number of user dictionaries can be processed against the text file.

Reviewing the bad word stack. Once scanning the file is completed, the menu appears for processing of the bad words that have been stacked during the scanning process. The program will show the word that it could not recognize and give you the options of disregarding the word, adding the word to the user dictionary, view the word in context (show the location of the word in the text), change the word and replace it in the text, and move backwards and redisplay the bad word stack. Once you

Continued On Page 124

Randy's Ravings Continued From Page 122

have finished running thru the bad word stack and have made your changes (if any) it will then update the text file in memory. Once the update is complete the spelling checker asks you for the output text file name. The instruction book suggests that you choose a different name than the original file name. Once the new text file is written the software will ask you for the name of the user dictionary you wish to have updated. When providing the name be sure NOT to use the main dictionary names of DICT1 or DICT2. The software will read in the old user dictionary, merge the new entries into the dictionary file out.

After using the Checker, it is a good idea to review the text file created by the spelling checker as it may have had to split a line of text in order to make your change. At this stage, it's just a simple matter of entering TI-Writer and using the REFORMAT command.

User Comment. Outside of a few rough edges (the manual could use some clarification and some of the prompts have some funny characters) the package works quite well. I have used it numerous times and found that all phases of the checker worked as stated. The user dictionary feature is a definite plus as you can have separate vocabularies for specific industries (I'm building one for computer related words). And it is easy to use. If you have TI-Writer or any other word processor that produces DISPLAY/VARIABLE 80 text files the 99/4 Auto Spell-Check is worth looking into. The package requires an expanded system with 32K and disk.

Par Fore
Golf Game For TI 99/4A
Endless Mt. Software
Distributed By
M.W.S. Computers
22 E. Tioga Street
Tunkhannock, PA 18657
$19.95 Cassette, $24.95 Disk

This is the first golf game that I have seen for the 99/4A and it's a curious one at that. What you see is an aerial view of the playing surface of either 9 holes (cassette version) or 18 holes (diskette version, requires expansion memory) of a golf course complete with the normal sand traps, water holes and tall trees that you find on a typical golf course. The game can have 1 or 2 players (speech optional) and can use either the joystick or keyboard. You use the joystick to change clubs (up and down) and to determine the angle of the shot you move the joystick left and right. A vertical bar which appears next to the ball indicates the direction that the ball will take--almost. You have to read the instruction book carefully or you will do what I wound up doing and that's hitting in the opposite direction (there is no arrow on the direction bar to indicate where you are going.) How much force you hit the ball with is determined by how long you hold down the fire button on the joystick and listening to a tone which varies in frequency. Holding it down all the way gives you 100% power, stopping short by releasing the fire button gives you a percentage (which is hard to determine WHAT percentage you are dealing with.) Once the ball is hit, you see it and the shadow of the ball cruise across the screen towards the destination. Depending on the options set on the game things, can go good or bad for you - if you select on target versus pinpoint accuracy and if you have a tendency to hook or slice it gets multiplied. Of course, hitting some of the objects can mess you up or be of benefit--on one hole I aimed straight across the treetops and once the pinball action stopped I was on the green and birdied the hole.

This ranks as one of the more interesting games that has been written in Extended BASIC and is a refreshing alternative to all the space-warp(ed) games available. ●