GUILFORD 99'ERS NEWSLETTER

Alles Zu Ende ?

SUPPORTING THE TEXAS INSTRUMENTS TI-99/4A COMPUTER

GUILFORD 99'ERS UG
3202 CANTERBURY DR
GREENSBORO NC
27408

TO:

VOLUME 9 NUMBER 4 OCTOBER-DECEMBER 1992

Bob Carmany, Pres. and Newsletter Editor (855-1538)
Tony Kleen, Sec/Treas (924-6344)   Bill Woodruff,Pgm/Library (228-1892)
BBS: (919)621-2623 --ROS
+-
The Guilford 99'er Users' Group Newsletter is free to dues paying members (One copy per family, please). Dues are $12.00 per family, per year. Send check to:Tony Kleen c/o 3202 Canterbury Dr., Greensboro, NC 27408. The Software Library is for dues paying members only. (Bob Carmany Ed)
+-

NEXT THREE MEETINGS

DATE: October 6, 1992   Time: 7:30 PM. Place: Glenwood Recreation
November 3, 1992   Center, 2010 S.
December 1, 1992   Chapman Street

BITS AND BYTES

By Bob Carmany

It has been almost two years since I came into possession of one of the HV99 Eprommers. To be honest, I really didn't know exactly what I was going to do with it in the beginning. After all, it would eprom a few then-common eproms up to the 16K 27128's in the 21/25V range. Shortly after I started putting F'WEB, ARCHIVER, and a few other programs into eproms (and thus into cartridges), new low/voltage eproms started hitting the market. These 12.5V eproms could be easily "fried" by the unmodified HV99 eprommer. Ron Kleinschafer came to the rescue a little over a year ago with a simple (for him) modification to drop the voltage by meand of a couple of resistors and a switch. Now, it could handle ANY eprom up to the 27128's in ANY commercially produced configuration. It was almost perfect!!

The software worked flawlessly and it was now just a matter of finding something besides program material to stuff into an eprom. I already had the EVEN and ODD console ROMs on disk and ready to go and the next project was to amass a collection of DSRs (Device Service Routines) for the various TI and non-TI peripherals available.

On my last trip to Oz, I "innocently" mentioned to Tony McGovern that a DSR dump program would be nice. About 30 minutes later, he had put the finishing touches on a F'WEB-dependent program that did just that! It was relatively crude --- an unequivocal 8K dump to disk without the required 6 byte header for the eprommer but it worked! A trip to Ron Kleinschafer's and a bit was added to stuff the unique 6 byte header on the DSR dump to allow the code to be nearly ready to use without modification. Along the way, I managed to collect "a few" DSRs from various devices like the HFDC controller and some CorComp devices. I didn't realize until much later that they were raw dumps and lacked the required header for the eprommer. That created a bit of a problem!

Here's what the Eprom screen looks like:

```
16K Eprom (Y/N) : N

RAM Start Address : >
```

RAM Last Address : >

Eprom Start Address : >

Preparing a file for the eprommer was fairly easy. The first thing that was required was to do a dummy read of a portion of memory --preferably an 8K segment -- with the eprom program. That filled the space with >FF which made it easy to find the true end of the file with SUPERBUG (included as a part of the Eprom software. You simply enter the two values as "Start" and "Last" RAM addresses. The first 6 bytes of the file were copied on a piece of paper and then replaced with the 6 byte eprom header: >0000 >2000 >6000.

The next step was to load the file into memory 6 bytes past the starting read address. For example, if you did your dummy read from >C000 to >E000, you would load the DSR file at >C006 with the Eprom software. I hope you are with me so far! Using SUPERBUG, you change the first 6 bytes (>FF) to the 6 byte eprom header (>0000 >2000 >6000) and then change the next 6 bytes back to the originals that you copied down on paper.

The final step is to go through the file and find the end and change the last value in the read to the actual file length. That value is entered as RAM Last Address" and the screen prompts are followed for saving a file. It sounds a lot more complicated than it really is -- the software has all of the prompts on-screen.

I think that my next A/L project is going to be a modification of the DSR dump program to compare the individual bytes of the DSR files with >FFFF and compute and install the correst file length without having to look for it in memory -- I'm basically lazy!

Anyway, after a bit of jiggling and tweaking, I have a fairly substantial collection of DSRs with which to reconsrtuct failed originals. Most of them, you see, fit vaery neatly into a 2532 eprom.

In the course of my explorations into the world of DSRs, I have found some interesting bits of information. Not all Disk Controllers are alike. By comparing the individual files, I have found at least three different versions in the TI Disk Controller alone. That might explain some of the idiosyncracies of different systems. I haven't had the opprotunity to look at the Myarc or Corcomp DSRs in volume to see if there are variations there but I suspect that there are modifications and different versions.

Incidently, here are a few CRU addresses of various devices. If you know of any more, let me know. There might be a DSR that I don't yet have lurking out there somewhere!

Product	CRU Address
Myarc HFDC	>1000 ->1F00 (16 different)
TI Disk Controller	>1100
CorComp Disk Controller	>1100
Myarc Floppy Disk Controller	>1100
RS232/PIO #1	>1300
RS232/PIO #2	>1500

Device	CRU Address
HORIZON RAMdisk	>1000->1700 (8 different)
Myarc RAMdisk	>1000
CorComp RAMdisk	>1000, >1400
Foundation RAMdisk	>1E00
Quest RAMdisk	>1000, >1400, >1600
GRAM Karte	>1000->1F00 (16 different)
P-GRAM Card	>1000->1700 (8 different)
Mechatronics 80-column	>1000
DIJIT AVPC 80-column	>1400
Mechatronic EPROMmer	>1900
HV99 EPROMMER	>1900
Mechatronic 128K+Printer	>1400
TI Thermal Printer	>1800
Corcomp Triple Tech	>1D00
P-code for Pascal	>1F00

For further information, you can use any CRU address for your peripherals that isn't already in use. That is why some of the devices have the capability of being addressed at more than one address. TI peripherals are generally at odd addresses starting at >1100.

After a few delays, I finally got the third Quest RAMdisk up and running. A local firm, Electronics South, programmed the GAL chips for me. They really have a first-rate operation. It took about 15-20 seconds per chip and there were NO problems with anything.

Trying to get the HORIZON to interface with the three Quests was a real problem, though. I have packed it away for future use. It seems that the ROS 8.14B doesn't co-exist with much of anything. I have had some feedback that it isn't just the Quest cards that get "fragged" by the HORIZON either!

As most of you know, Myarc is no longer in business. Cecure Electronics, 7759 Scepter Dr. #7, Franklin WI, is dong repairs on the Myarc stuff that is still around including the HFDC. It appears that they do custom repairs, design, and fabrication of circuit boards, etc. They also got glowing reviews in a past issue of MICROpendium. It looks like an address that you might want to keep!

RF MOD REPAIR

Many TI owners are not as affluent as I am and use a TV instead of a monitor. Occasionally, the RF modulator needs to be fine tuned to eliminate annoying background noise, such as humming or buzzing. A simple internal adjustment on the modulator will often alleviate this problem. The following procedure is to be done when all equipment is on and operating. (If you have the old version of the TI900 Video Modulator, this procedure will not work.)

You will need a small, flat, thin-bladed screwdriver.

1> Turn the volume of the TV all the way down, but do NOT turn it off.

2> Select the Master Title Screen on the computer.

3> Using the title screen color grid, fine tune the TV to the best color picture you can.

4> Using the screwdriver, pry off the lid of the modulator by lifting under one edge of the lid near the indentation holding it on.

5> Lift off the lid and turn the TV volume up to half.

6> Insert the screwdriver blade into the slot of the small box labelled CV1 and turn it slightly until the background noise is at a minimum. (This should take less than 1/8th of a turn.)

7> After bending the modulator lip edge back into place, put it back over the modulator box and press it firmly in place until it snaps.

This should take care of the any problems with background noise.

SCROLL DEMO

Here's a little demo which will allow you to scroll part of a screen and only takes up two lines of program code.

```
100 CALL SCREEN(15):: PRINT : : : :"T HIS PROGRAM WILL ACCEPT
ANY INPUT AND SCROLL UP 1 LINE."
110 PRINT : :"BUT ONLY THE BOTTOM HALF F OF THE SCREEN WILL SCROLL. THE
TOP HALF WILL STAY INTACT."
120 PRINT :"____________________________ ____": : : : : : : : : : :: B$=RPT$("
",252)
130 ACCEPT AT(24,1)SIZE(28):A$ :: A$= A$&RPT$(" ",28-LEN(A$))::
B$-SEG$(B$,29,224)&A$ :: DISPLAY AT(15,1):B$ :: GOTO 130
```

If you want to scroll down, change the "29" to a "1" in line 130. To change the location where scrolling occurs, change the DISPLAY AT. If it is higher than 15, it will split the screen so that the top and bottom will scroll and the middle stays the same. (You will also need to change the ACCEPT AT so that it lines up with the scrolling screen)

You are limited to 9 lines because strings are limited to 255 characters and 9 lines takes up 252.

This gem comes from the lightpen author, Edwin McFall. I hope you can

find some use for it.

BASIC TIPS

1. HOW TO DISABLE THE "FUNCTION-QUIT" HARDWARE RESET: TI Basic and Extended Basic has two ways to exit, one by typing in "BYE" which will properly close all files, or by pressing "Function=(QUIT)". The latter method really should not be used at all since files will not be closed and unpredictable things can happen if function quit is pressed while files are open. Unfortunately, many of us had the nasty experience of accidentally hitting "Function Quit" with the result that everything in memory was lost and files were scrambled. IF you have Extended Basic and 32K memory, the following will disable "Function Quit": CALL INIT::CALL LOAD(-31806,16). This can be typed in as a direct command, or could be the first line of an extended basic program.

2. HOW TO SPEED UP EXTENDED BASIC. While XB offers faster execution speed for some applications compared to console basic, XB can be speeded up even further by disabling sprite graphics (naturally this works only if the program does not use sprite graphics). The program statement is: CALL INIT::CALL LOAD(-31878,0). There are several different releases or versions of Extended Basic and the speed-up effect will be more pronounced with some versions than with others. 32K memory is required.

3. HOW TO RECOVER MEMORY IN TI BASIC/EXTENDED BASIC WITH DISK DRIVE ATTACHED. The TI operating system automatically sets aside memory to serve three concurrent open files. A minimum of 534 bytes of memory are taken up by general expansion overhead plus 518 more bytes for each of the three files opened up by default, or a total of just about 2K. If you know that you will have only one file open, key in the following DIRECT COMAND: CALL FILES(1)(Press ENTER) NEW (Press ENTER). This sequence will recover 1K of precious memory. Please note that this sequence can be keyed in as a command only and cannot be used as a program statement. Don't forget the NEW or results will be unpredictable. This prodecure can be used with both TI Basic or Extended Basic. With TI Basic and attached disk this is more essential than ever since TI Basic will only address 16K and you can ill afford to lose much of that.

MEMORY CALC

In a way computer merchandising and advertising resembles the "Horsepower-War" in automobiles of not too many years ago. Manufacturers tout their wares as 64K or 128K machines but fail to tell that the operating system is disk-based and after it is read in, only 48 or even 32K of user space remains....

Much to the credit of TI, the 99/4A operating system from the very beginning was in the form of solid-state read-only memory chips which does not require a separate disk-based program to be read-in. The advantage to the user is that such a system is very efficient, very fast and very dependable. Compared to other systems the 99/4A seems to lack horsepower (memory), which really is not true as the following tabulation will show.

A 99/4A system can have three types of memory chips: RAM (random-access read/write chips, available for user programs and data), ROM (Read-only memory, mainly operating system), and GROM (a type of

read-only chip programmed by TI in Graphics Programming Language (GPL), mainly Basic and Extended Basic interpreters). Here is how the numbers stack up:

```
Console Operating System...........ROM........8K
RS232 Device Service Routines......ROM........2k
Disk Device Service Routines.......ROM........8K
Extended Basic Support.............ROM.......12K
TI Basic Interpreter (Console).....GROM......18K
Extended Basic Interpreter.........GROM......24K
Speech Support (Synthesizer).......ROM.......32K
Console Random Access Memory.......RAM........8K
Video Display Memory...............RAM.......16K
Memory Expansion (Assembler Rout.).RAM........8K
Memory Expansion (X-Basic).........RAM.......24K
```

This table shows that a 99/4A system with diskette drive, speech synthesizer, and RS232 card, using Extended Basic controls a total of 160K as RAM, ROM or GROM...not bad at all.

HYPHENATOR

A TI-Writer Utility

One of the nice features of TI-Writer is the ability to type in word-wrap mode, which speeds up typing by allowing you to concentrate on text without having to worry about exceeding the right margin.

There is a draw-back, though, to word-wrap in that longer words which would exceed the right margin are scrolled to the next line in their entirety.

The disadvantage of this system becomes obvious when text is printed out using the FORMATTER when there is a tendency for the right margin to have the "jaggies".

Using the right-margin-flush feature (.AD) of FORMATTER provides only a partial cure since now FORMATTER inserts blank spaces between words to fill up the line. The amount of white space inserted varies with the number of characters that need to be filled with the result that text can be rather blotchy in appearance.

The only sure way to improve the appearance is to re-edit word-wrap text and to hyphenate as much as possible where lines break.

Unfortunately, the EDITOR of TI-Writer is not quite up to that task:

At the most, the EDITOR can display 80 characters per line whereas the FORMATTER and most printers can handle Elite (up to 96 characters) or Compressed (up to 132 characters) per line. In such a case the EDITOR is of no help.

A further hindrance is that the EDITOR will display imbedded print commands which is helpful in creating text but a serious obstacle in fine tuning right margins. Typical examples are string commands to turn super- or subscript on or off or the "ampersand" or "at" commands of TI-Writer for underlining and double strike.

Quite often for ease in typing and editing users elect to fix the right margin at 40 characters to do away with horizontal scrolling. An attempt to judge the final appearance of text by resetting tabs to final form and using the "Reformat" command can be misleading since previously entered indentations are then ignored.

HYPHENATOR is an editing utility for TI-writer that succeeds in addressing all these problems:

HYPHENATOR can handle print widths , or right margins of up to 160 characters.

HYPHENATOR properly accounts for imbedded print commands, be they the TI-Writer "at" or "ampersand" type or special character mode (CTRL U) transliterate symbols.

HYPHENATOR makes it possible to change margin settings within a document for quoted text that needs to be indented further.

HYPHENATOR recognizes a double "ampersand" or "at" symbol as a character to be printed rather than as a non-printing control character.

HYPHENATOR allows for the FORMATTER idiosyncracy of inserting two blanks following a period even though only one space might have been keyed in.

The program is a stand-alone utility that can be loaded using the LOAD AND RUN option of the Editor/Assembler or Mini-Memory cartridge. After loading, HYPHENATOR will prompt for the name of the input file (the name of the document created with TI- Writer EDITOR) and a name for an output file which HYPHENATOR will create in TI-Writer format. The use of either a single disk drive or two disk drives is supported. The original text file will not be altered in any way.

Once the proper files are set up, HYPHENATOR will read in a paragraph of text which can be up to 5280 characters long (a full page, single-spaced).

According to the margin and indentation information for which HYPHENATOR has prompted, the first block of text will be displayed (five lines) with an end-of-line marker exactly on that character which would be the last character to be printed by FORMATTER, with all non-printing characters, extra spaces, etc. already accounted for.

If the end-of-line marker points to a space or the last character of a word, no further action is necessary except for pressing the <ENTER> key to bring up the next line.

If the EOL marker points to the middle of a word, a decision needs to be made whether hyphenation is possible. If yes the editing cursor <FCTN S> should be moved to the last character prior to the hyphen and a hyphen symbol keyed in. HYPHENATOR will supply the necessary prompts to complete the job.

If hyphenation should not be possible, moving the cursor to the first blank and pressing <ENTER> would complete the job.

Once all the lines of a particular block have been edited a screen message

will prompt for writing the block out to the disk file.

For speed and convenience, HYPHENATOR has a number of imbedded defaults. Thus empty lines or lines with only format control characters are written to the output file without user intervention.

An "Oops" feature can be invoked at any time by pressing <CTRL 1> to go back to the beginning of the paragraph. This comes in handy if there should be any second thoughts about a line just completed.

<CTRL 3> and <CTRL 4> toggle the screen display color which make it possible to display many combinations of screen and text color.

<CTRL 2> invokes the margin/indentation set option to change these values at any time.

<CTRL 9> writes out the remainder of an input file without further editing to the output file. This comes in handy where only a portion of text needs that final touch.

Any time a line of text is displayed on the screen, minor editing is possible. Thus "recieve" can be changed to "receive". The limitation is that the new text must have the same length as the original text.

HYPHENATOR is written in Assembler and thus is very fast. A test with a 59 sector compressed print document could be "fine-tuned" in under twenty minutes.

The use of a pocket dictionary in conjunction with HYPHENATOR is strongly recommended. Due to the memory limitations of the 99/4A system, HYPHENATOR can only show WHERE to hyphenate. The "IF" and "HOW" is up to the user. This is where a pocket dictionary comes in handy.

All-in-all, HYPHENATOR is an excellent utility along the lines of Tom Kirk's AUTO SPELL CHECK to make a good product, such as TI-Writer, even better.

Exploring BASIC Programs

By Tim MacEachern

The program listed below demonstrates how BASIC programs are stored in the 99/4A. The program as listed will work in Extended BASIC with the Memory Expansion card or peripheral attached. A similar program can be run in normal BASIC with the Editor/Assembler or Mini Memory module inserted. To convert this program to normal BASIC simply change the calls to subroutine 'PEEK' in lines 200,240 and 260 into calls to subroutine 'PEEKV'. That is, add a 'V' between the 'PEEK' and the '(' in each line. This program will not work properly in Extended BASIC unless you have the memory expansion.

The techniques used in this program are intended to make it as easy to understand as possible, while still showing how the DEF statement in BASIC can be used to do all the hard work for you. For instance, lines 100 to 130 of the program create a function HEX which will convert a string of hexadecimal (base 16) digits into a decimal number. As can be seen in lines 150 and 170, this allows us to write the actual hexadecimal addresses as used by assembler language programmers.

Line 130 takes the string of hexadecimal digits given to it and pads it with leading zeroes to make sure that there are four hex digits. Then

function HEX4 is called to evaluate this four-digit hex number. In line 120, HEX4 splits the number into two two-digit hex numbers and combines them to get the proper decimal result. Similarly, line 110 splits a two-digit hex number into two one-digit numbers. Line 100 then is used to figure out the value of each separate hexadecimal digit.

Using nested DEF statements as in this program can simplify development of a working program, but be warned that DEF statements take considerably longer to run than the exact same code put directly into your lines wherever needed. Still, you may find it convenient to write some programs that consist solely of DEF statements ! After such a program is RUN in normal BASIC (or in Extended BASIC without the memory expansion), the defined functions will be available to use in BASIC's calculator mode. For instance, if your program consisted of lines 100 to 130 only, it would provide a conversion function from hex to decimal that you could use while in calculator or direct command mode.

Let's get back to the program. Line 140 defines a function that is used to convert a 16-bit unsigned number (from 0 to 65535) into a 16-bit signed number (from -32768 to 32767). For some strange reason BASIC insists on signed numbers for addresses passed to PEEK, PEEKV, LOAD and POKEV. So whenever an unsigned address is calculated function MA is used to convert it to a signed number. This function works by comparing its argument to the largest positive value allowed. If the number is too big the comparison yields a value of -1. The rest of the expression then caused 65536 to be subtracted from the argument value, giving the correct result. If the original number is okay (from 0 to 32767) the comparison yields a result of 0 and the value of the function is the same as the value of its parameter. It seems complicated to write functions like this, but try to figure them out - you may find them fascinating.

BASIC stores your program in two sections. In the top of memory it stores each line of the program, not necessarily in the correct order. As a matter of fact, each time you edit a line, it becomes the last line in this area, with all other lines packed together above it. Each statement is made up of three parts. The first byte is the length of the rest of the statement in memory. The last byte is zero, and in between are bytes that represent the particular BASIC statement you have written. BASIC keywords are translated into a single byte each (known as a token) while strings and numeric constants are represented as a leading token (199 or 200) followed by a length byte, followed by the ASCII character values of the string. By running this program you can determine how other elements of a BASIC program are stored.

Underneath the statements (that is, lower in memory) is a list of statement numbers and pointers to the first token in each statement. Each statement in your program has a four-byte entry in this list. The bottom two bytes store the statement number. The top two bytes are a pointer to the first token in the statement (the byte following the length byte). This program goes through this list and prints out each token in the statements of your program.

Pointers to the top byte in the statement pointer list and the bottom byte in the list are stored in the scratchpad RAM and read by lines 150 to 180. The loop that starts in line 190 examines each statement in the program. If you have gotten this far in the article, you will understand how the rest of the lines in the program print out each token of each line.

TIM MCEACHERN
PO BOX 1105
DARTMOUTH, N.S.

CANADA B2Y-4B8

```
100 DEF HEX1(X$)=POS("123456789ABCDEF",X$,1)
110 DEF HEX2(X$)=HEX1(SEG$(X$,1,1))+HEX1(SEG$(X$,2,1))
120 DEF HEX4(X$)=HEX2(SEG$(X$,1,2))6+HEX2(SEG$(X$,3,2))
130 DEF HEX(X$)=HEX4(SEG$("0000"& X$,LEN(X$)+1,4))
140 DEF MA(X)=X+65536*(X>32767)
150 CALL PEEK(MA(HEX("8332")),A,B)
160 TOSL=MA(A6+B)
170 CALL PEEK(MA(HEX("8330")),A,B)
180 BOSL=MA(A6+B)
190 FOR PTR=TOSL-3 TO BOSL STEP -4
200 CALL PEEK(PTR,A,B,C,D)
210 PRINT "STATEMENT #";A6+B
220 PRINT "TOKENS:"
230 SPTR=MA(C6+D)
240 CALL PEEK(SPTR-1,L)
250 FOR I=0 TO L-1
260 CALL PEEK(SPTR+I,X)
270 PRINT X;
280 NEXT I
290 PRINT : :
300 NEXT PTR
310 END
```