

ST-MON 2.04

User's Manual

Copyright 1984 by David C. Wiens
All rights reserved

Sardis Technologies
2261 East 11th Avenue
Vancouver, B.C.
Canada V5N 1Z7

CREDITS

Special thanks go to Will W. -- several of the improvements in this release, especially in the manual, are due to his suggestions.

TRADEMARKS

SBUG-E is a trademark of Southwest Technical Products Corp. (SWTPc)
FLEX is a trademark of Technical Systems Consultants (TSC)
MIKBUG is a trademark of Motorola
OS-9 is a trademark of Microware Systems Corp.
STAR-DOS is a trademark of Starkits

COPYRIGHT INFORMATION

The entire contents of this manual and all information on the supplied EPROM(s) are copyrighted by David C. Wiens. It has been sold to you on a "single end user" basis. It is permissible to make copies of the EPROM data only for backup purposes and to make modified versions. However, if it becomes necessary to run the ST-MON monitor program on more than one computer simultaneously, additional copies or a multi-copy license must be purchased from the supplier.

DISCLAIMER

Although much effort has been made to ensure the accuracy of the software and documentation, David C. Wiens and Sardis Technologies disclaim any and all liability for consequential damages, economic loss, or any other injury arising from or on account of the use of, possession of, defect in, or failure of the supplied material.

This manual last revised August 2, 1985.

ST-MON Monitor ROM Version 2.04

=====

The ST-MON System Monitor program, supplied in a 4K byte EPROM (450 nsec. 2732), enables the ST-2900 computer to do the following after powerup or system reset:

- initialize the system, including I/O chips
- perform self-diagnostics
- communicate with a terminal for programming and debugging functions
- load the FLEX, STAR-DOS, or OS-9 operating system from disk

ST-MON was designed for the ST-2900 CPU board and will not work in any other system.

Although ST-MON has a few similarities to the SBUG-E monitor from SWTPc, all programs written to directly access SBUG-E will have to be modified to use ST-MON's routines.

Well commented source code of ST-MON is available separately for users who need to modify the monitor, or want a better understanding of how it works.

ST-MON, with its lack of breakpoints and register commands, is not intended to be a full fledged debugging monitor -- rather, its main purpose is to allow contents of memory to be examined and changed, and to boot an operating system. If you need debugging capability, and are running FLEX (or STAR-DOS), we highly recommend TSC's excellent "6809 DEBUG" package available from Southeast Media or Frank Hogg Laboratory. Microware's DEBUG utility generally comes free with OS-9, although this program only has a fraction of the capabilities of TSC's.

ST-MON Monitor Powerup/Reset Initialization

=====

After powerup or system reset, ST-MON initializes the 6883 SAM chip and the two serial I/O ports. If any ST-2900 RAM-512 boards are plugged in, the first one has bank #0 activated, the second board (if any) is deactivated. At this point the SAM chip is in memory map 0, Page 0, with the RAM on the CPU board occupying only \$0000-\$7FFF, the ST-MON EPROM occupying \$A000-\$AFFF, and the RAM-512 board, if any, occupying \$E000-\$EFFF.

Next ST-MON performs a checksum test of the EPROM. If the calculated checksum matches the expected value, it doesn't display anything, but continues on to the next step. If the checksum is not OK, it prints "M1" to the console. This could indicate that a memory location in the EPROM is bad, or merely that you modified ST-MON and didn't give it the new correct checksum (refer to Appendices C and D).

Now the dynamic RAM chips on the CPU board are tested. Every single byte of the 64K RAM is tested (possible because no stack is used by the test routine) and the contents of memory are not changed. If an error is discovered, "M2" is displayed and the address of the first bad location found is saved. The location's page number (0 or 1) is stored at \$C000, and the address (\$0000-\$7FFF) within that page is stored at \$C001/\$C002. Later, when the powerup/reset initialization is finished, you can use the "M" command to display these values. If all 64K bytes passed with flying colours, no message is displayed. No news is good news!

At this point the prompt "CW?" should be seen. Usually you will respond with "C" (see Appendix A for an explanation), which copies most of the monitor program from EPROM to RAM, then switches the EPROM out of the memory map to give you a full 64K (minus 256 bytes) of RAM, with the monitor occupying most of \$F400 - \$FEFF. At this point the SAM chip is in memory Map 1.

Finally the identification/copyright notice is displayed and the command mode is entered. An equals sign, "=", at the left hand margin indicates the system is ready to accept a command.

If the "M2" error message appeared a few steps back, you should now run the "T" command to do a thorough memory test.

ST-MON Monitor Commands

=====
 Commands consist of a one character code followed by 0 - 3 arguments. Some arguments are one or two alphabetic characters, but most are entered in hexadecimal, either 2 or 4 digits each. Keying any non-hex digit (such as a carriage return) when a hexadecimal value is required aborts the command. Here is a summary of the available commands:

M Memory examine/change byte by byte
 F return to FLEX
 J Jump to subroutine or program
 T Test memory
 E Examine large blocks of memory
 S Search for byte
 B move Block of memory
 A fill Area of memory
 L Load Motorola "S1" format ASCII "tape"
 D Disk bootstrap load

Several commands below refer to ST-MON locations BDRIVE, DBLSTP, JMPADDR, IPORTL, and OFFSET. These locations are described near the end of this manual in the section "User Referrable RAM Locations". Also, all references in this manual to FLEX apply equally to STAR-DOS Level I.

 "A <begin address> <end address> <value>" -- Fill Area of Memory

The memory fill command allows the user to fill a contiguous area of memory, from the begin address to the end address, inclusive, with the 1 byte data value entered. Note -- be careful not to overwrite ST-MON or its stack. Refer to the memory map several pages below for more details.

Example:

=A 2000 7FFF 3F

This command can come in handy when testing a program that has a bug in it that causes it to go to never-never land, and eats up ST-MON in the process. By filling user memory with \$3F's (SWI opcode), then loading the program-under-test on top of it, the chances are the program, shortly after it starts to go wayward, will hit one of the SWI instructions which causes a return to ST-MON. You have in effect put a short leash on the program so it can't wander as far!

 "B <source-begin address> <source-end address> <destination-begin address>"
 -- Move Block of Memory (ie. duplicate it)

The move block command copies the contents of the contiguous area of memory, source-begin to source-end addresses, inclusive, to a same-sized area starting at the destination-begin address. If the source and destination blocks overlap, the results might not be what was desired. If no overlapping occurs, the contents in the source area remain unchanged. Note -- be careful not to overwrite ST-MON or its stack. Refer to the memory map several pages below for more details.

Example:
=B F800 FEFF 1000

Refer to Appendix E for an example of a use for this command.

"D F" -- Disk Boot FLEX Operating System

Before executing this command, check with the "ST-2900 FLEX Conversion Package" manual to see whether or not you first need to change the values in locations BDRIVE (\$FEA1) or DBLSTP (\$FEA2).

The FLEX disk boot option begins the process that loads the FLEX (or STAR-DOS) disk operating system from disk. The routine selects the specified boot drive, side 0, single density, restores to track zero, then reads sector 1. From this it determines the format of the disk and the location of the binary file (usually FLEX.SYS) to be loaded. The binary file is then read and loaded into memory at the addresses specified in the file. When the end of the file is found, the loading process terminates and control is passed to the last transfer address found. For more details on the structure of a binary file, transfer addresses, etc., refer to TSC's "FLEX Programmers Guide" or STAR-KITS' "STAR-DOS Level I User's Manual". Also consult the "ST-2900 FLEX Conversion Package" manual.

The boot routine detects several kinds of errors such as "sector not found" and CRC errors (ie. bad data) and will try several times to re-read the desired sector before giving up. If a hard (ie. unrecoverable) error is detected, control will be returned to the monitor. If there is no disk in the drive, or if the disk drive or interface isn't connected, the boot command will hang until the system reset switch is pressed.

Once the "D F" command has been called, the "D OC" and "D OS" commands are disabled until the next time you respond with "C" to ST-MON's "C/W?" prompt. This is because the memory occupied by those commands is needed by FLEX. The rest of ST-MON remains intact, and in fact you can exit FLEX to return to ST-MON with FLEX's "MON" command, then return to FLEX with ST-MON's "F W" or "F U" commands.

Note -- the disk you boot from must have been formatted by the "FORMAT" utility supplied with the ST-2900 FLEX Conversion Package. Other formatters don't set up track zero / sector 1 with all the values needed by ST-MON's "boot" routine. See Appendix H for more details.

"D OC" -- Disk Boot OS-9 Operating System (from CoCo format disk)

Before executing this command, check with the "ST-2900 OS-9 Conversion Package" manual to see whether or not you first need to change the values in locations BDRIVE (\$FEA1) or DBLSTP (\$FEA2).

The OS-9 disk boot option begins the process that loads the OS-9 disk operating system from disk. The routine selects the specified boot drive, side 0, double-density, restores to track zero, then reads sector 1 (logical sector number 0). From this it determines the location of the "OS9Boot" file. After also finding the location of the "OS9Kernel" file, both of these files are loaded into memory.

The loading routine detects several kinds of errors such as "sector not found" and CRC errors (ie. bad data) and will try several times to re-read the desired sector before giving up. If a hard (ie. unrecoverable) error is detected, control will be returned to the monitor. If there is no disk in the drive, or if the disk drive or interface isn't connected, the boot command will hang until the system reset switch is pressed.

Once the two files have been loaded, ST-MON is temporarily relocated to \$AXXX, and the OS-9 kernel moved to \$FXXX. After the ST-2900 OS-9 data area at \$FEXX is initialized, control is passed to the kernel's "OS9" module. At this point ST-MON is "discarded".

Unlike FLEX, you cannot access ST-MON once you have booted OS-9. OS-9 cannot co-exist with ST-MON because of OS-9's memory requirements, module structure, and other features such as multi-tasking and interrupts. All is not lost, however, as OS-9 includes a separate DEBUG program which has more functions than ST-MON.

Refer to the "ST-2900 OS-9 Conversion Package" manual for more details.

 "D OS" -- Disk Boot OS-9 Operating System (from a standard OS-9 format disk)

This option is identical to the "D OC" command except that the boot disk used is in the standard OS-9 format, rather than the Radio Shack CoCo OS-9 format.

 "E <begin address> <end address>" -- Examine Block of Memory

The examine memory command displays the contents of memory from the begin to the end address, inclusive. The contents are displayed in both hexadecimal and ASCII. Each line shows 16 bytes and is aligned on an even 16 byte boundary (ie. the first displayed byte of each line has an address ending with "0"). Non-printable characters (\$00-\$1F, \$7F) are shown as dots in the ASCII display. A sample display looks like this:

```
=E FE7F FE96
FE70 250B8146 22078037 39326132 61433953 %..F"..792a2aC9S
FE80 542D4D4F 4E20312E 30202863 29203139 T-MON 1.0 (c) 19
FE90 38342044 4357043D 043F3F04 0D0A0000 84 DCW.=.???......
```

The display can be temporarily halted at any time by pressing the "Escape" key on the terminal. At this point, pressing the "Return" key will abort the command; typing any other character resumes the display.

 "F W" -- Return to FLEX Warmstart

The "F W" command is used to return control to FLEX (or STAR-DOS) via its Warmstart address (\$CD03) after having exited FLEX by using the "MON" command, or having pressed the system reset switch.

FLEX must have been active prior to re-entering ST-MON, and must not be damaged, otherwise unpredictable results will occur. If ST-MON was entered via system reset, unpredictable results may occasionally occur after returning to FLEX.

Do NOT use this command if after system reset you responded with "C" to ST-MON's "CW?" prompt (refer to Appendix A for more details).

"F U <address>" - Return to FLEX environment, but to user program

The "F U hhhh" command is used to jump to a user program running in the FLEX (or STAR-DOS) environment. That is, the program uses FLEX routines and/or will eventually return control to FLEX's "Warmstart" address (\$CD03).

A typical use for this command is where you want to modify a few bytes of a program before executing it. Use FLEX's "GET" command to load the program from disk, use "MON" to exit to ST-MON, use the "M" command to make the desired patches, then use the "F U" command to jump to the entry point of the program. If the documentation of the program does not tell you the location of its entry point, you will first have to run TSC's "MAP" command to find out.

In order to use the "F U" command, FLEX must have been active prior to re-entering ST-MON (via "MON" command or system reset), and must not be damaged, otherwise unpredictable results will occur. If ST-MON was entered via system reset, unpredictable results may occasionally occur after the "F U" command is executed. Do NOT use this command if after system reset you responded with "C" to ST-MON's "CW?" prompt. Refer to Appendix A for details.

"J <address>" -- Jump to subroutine or program

The jump-to-subroutine (or program) command lets the user execute a routine beginning at the specified address. The address in "JMPADDR" is ignored. Return to ST-MON can be via an "RTS" instruction at the end of the subroutine, or a jump-to-the-monitor instruction in the routine, coded as
 JMP [\$FEC3]

Note -- do NOT use this to return to FLEX (or STAR-DOS) or to execute a program that will use FLEX routines or will return to FLEX warmstart (\$CD03). Use the "F W" or "F U" commands instead.

"L" -- Load Motorola "S1" format "tape"

The "load" command is used to input data from a serial port and store it into memory. In times past this data would have come from cassette or paper tape readers. Nowadays, you might connect the ST-2900 CPU board to another computer via an RS-232 serial link and want to transfer a program from the other system to the ST-2900.

Before using the "L" command, the IPORTL vector at \$FEAC/\$FEAD must point to the desired input port (it usually contains \$FF20 which points to port A; changing it to \$FF28 points to port B). The 2 byte variable "OFFSET" at \$FEB4/\$FEB5 must also be set to the desired value, usually \$0000. Any non-zero value will be added to the block address (see item "c" in Appendix B). After typing "L", the load process continues until the "S9" end-of-data code is received. If a checksum error is detected, or a byte of data is not properly stored (bad memory, etc.), a "?" will be printed and the loading process aborted.

Although ST-MON does not contain a matching "punch" routine to output "S1" format data, programs to perform this function are available to run under the FLEX operating system. OS-9 has BINEX and EXBIN commands to send and receive "S1" data.

See Appendix B for details on the Motorola S1/S9 (MIKBUG) block format.

 "M <address>" -- Memory Examine/Change

The memory examine/change command lets the user look at and modify the contents of memory, one byte at a time, beginning with the specified address. Each byte is shown on a separate line, with its address, its current contents in hexadecimal, and its current contents in ASCII (if printable). At this point you can key one of several sub-commands:

- a) typing a valid 2 digit hex value changes that memory location to the new value (a "?" is displayed if not successfully changed because of bad RAM or because it was ROM, or it was changed OK but cannot be read back because it was a write-only register in an I/O chip), then advances to display the next location. Note -- if the address was in the \$FF00-\$FFFF range, the command is exited after the one location was changed.
- b) typing an apostrophe followed by one ASCII character stores that ASCII character into the memory location.
- c) typing a comma or space advances to the next location without changing the current location
- d) typing a backspace or minus sign or up arrow displays the previous location instead of the next one -- the current location is not changed
- e) typing a period or carriage return exits the command -- the current location is not changed
- f) if you type anything else you will get a question mark, and the current location will be redisplayed.

Entire machine language programs can be entered this way.

 "S <begin address> <end address> <search value>" -- Search for byte

The search command finds all occurrences of the specified 1 byte data pattern between the specified begin and end addresses, inclusive. Each "hit" is displayed on a separate line, with its address, the two bytes before the search value, the search value, and the two bytes after. Note that the address is that of the middle byte. A sample display looks like this:

```
=S C100 C12F BD
C114 0000 BD CD24
C11A C840 BD CD2D
```

The display can be temporarily halted at any time by pressing the "Escape" key on the terminal. At this point, pressing the "Return" key will abort the command; typing any other character resumes the display.

"T <begin address> <end address>" -- Test Memory

The memory test command performs up to 65,535 passes over the specified address range. A "*" is printed for each pass. In the first part of each pass, test patterns are written to the entire specified block of memory. After pausing for 200 msec. to flush out potential dynamic memory refresh problems, the test patterns are read back. Any locations with errors are displayed on the terminal to indicate the address, and which bits were in error. For example:

```
=T 3000 3FFF*****35A7 60 **
```

In this example, memory from \$3000 to \$3FFF (inclusive) is being tested. A problem with memory at address \$35A7 is indicated. The first 8 passes (represented by 8 asterisks) were OK, then on the ninth pass two bits (represented by the value \$60) were in error. Each of the 8 bits in a byte in memory are stored in a different memory chip (U3 through U10) on the ST-2900 CPU board, with the high order bit in chip U3, and the low order bit in U10. Thus the bit error code of \$60 (binary %01100000) indicates chips U4 and U5 are partially bad.

Since a full 65,535 passes over a large block of memory can take many hours, the test can be aborted at the end of any pass by keying any character on the keyboard.

Note -- be careful not to test the memory currently occupied by ST-MON itself or its stack. Refer to the memory map a few pages below for more details. Unlike the automatic memory test after powerup/reset, this command DOES destroy the contents of the tested memory. Appendix E shows you how to test the memory that is normally occupied by ST-MON and its stack.

Advanced Programmer's Guide

=====

ST-MON, after initialization, resides in approximately 2.5K of RAM, from \$F4FD to \$FEFF, including variables and vectors. RAM from \$C000-\$C07F is reserved for the stack, and the stack pointer is initialized to \$C07F. The location of ST-MON's stack can be changed at any time by changing the contents of STACK (\$FEA3/\$FEA4) from the default value of \$C07F to any other desired value, then re-entering ST-MON at the address specified by the MONITOR vector (at \$FEC3/\$FEC4).

The memory space occupied by the D,J,T,E,S,B,A,L commands (\$F4FD - \$FC7C) can be used for other purposes if these commands are not needed -- just change TABSIZ (at \$FEA5) to zero. The "M", "F W", and "F U" commands will still be accessible.

On the other hand, the command table has room to add up to four more RAM-based commands to the monitor (perhaps they would be loaded from disk after the disk operating system has been loaded?). The contents of memory location CMDTAB (\$FEBF/\$FEC0) indicate the location of the command table (\$FC59 for version 2.04).

If you need more information than is provided in this manual, source code for ST-MON is available separately for a modest fee.

Several routines below refer to memory locations TABSIZ, CMDTAB, ICLSW, OCLSW, IPORT, IPORTL, OPORT, OPORTA, ECHOFL, and UCFLAG. These locations are described near the end of this manual in the section "User Referrable RAM Locations".

User Callable Monitor Routines

A table of monitor routine addresses is provided at locations \$FEC1-\$FEE6 for use by user programs.

Here is a list of user callable routines:

Addr	Name	Description
FEC1	LOAD	Load Motorola S1/S9 format data from serial port
FEC3	MONITOR	Re-enter ST-MON
FEC5	INCHEK	Check if input character waiting
FEC7	INCH8	Input character from terminal (8 bits)
FEC9	INCHN	" " " " , no echo
FECB	INCHE	" " " " , echo
FECD	INCHTE	" " " " , echo if flag set
FECF	IN1HEX	Input 1 hexadecimal character from terminal
FED1	IN2HEX	" 2 " " " "
FED3	IN4HEX	" 4 " " " "
FED5		(reserved - see write-up below)
FED7	OUTCH	Output character to terminal
FED9	OUT1SP	Output one space character to terminal
FEDB	OUT1HX	Output 1 hexadecimal character to terminal
FEDD	OUT2HX	" 2 " " " "
FEDF	OUT4HX	" 4 " " " "
FEE1	PCRLF	Output CR, LF, nulls to terminal
FEE3	PDATA	Output data string " "
FEE5	PSTRNG	Call PCRLF, then output data string to terminal

Two special purpose routines, ROMMAP and RAMMAP, are described in Appendix F.

Calling User Routines

Routines in the ST-MON monitor should be called through the vector table listed above. Calling the routines directly is not recommended because future releases of ST-MON will have different internal addresses. On the other hand, the vector table addresses shown above will be unchanged. The normal way to call a routine in ST-MON is to use an indirect subroutine call.

```
eg.   OUTCH   EQU   $FED7
      LDA     #'W
      JSR    [OUTCH]
```

LOAD - \$FEC1

Executes the ST-MON "L" command.

MONITOR - \$FEC3

Re-enters the ST-MON monitor. Resets the stack pointer and prints the sign-on message, sets switches ICLSW and OCLSW to point to IPORT and OPORT, sets ECHOFL to "echo", and sets UCFLAG to "convert to upper case". Prompts for command.

Unlike SBUG-E, ST-MON does not re-initialize the serial ports or RAM interrupt vectors at this entry point.

INCHEK - \$FEC5

Checks serial port A or B (depending on values in ICLSW/IPORT/IPORTL) to see if a character has been received and is waiting to be read. All registers are preserved, and the Z condition code is set to EQ if no character was waiting, and reset to NE if a character is waiting.

INCH8 - \$FEC7

Gets one 8 bit character from serial port A or B (depending on values in ICLSW/IPORT/IPORTL). Does not echo it. The character is returned in Register A; all other registers are preserved. If the port was set to only 7 data bits, the character returned will be truncated to 7 bits, with the high order bit set to zero.

If the DUART's status register for that port has the "overrun error" flag set, an ASCII "bell" control character (\$07) is output via routine OUTCH, then the error flag is reset. To disable the sending of the "bell" character, execute the following ST-MON command:

=A FDEC FDED 12

INCHN - \$FEC9

Gets one character from serial port A or B (depending on values in ICLSW/IPORT/IPORTL). Does not echo it. The character is returned in Register A with the high order bit set to zero, and with lower case a-z converted to upper case if UCFLAG set. All other registers are unchanged. The note under INCH8 regarding the "bell" character also applies to this routine.

INCHE - \$FECB

Gets one character from serial port A or B (depending on values in ICLSW/IPORT/IPORTL). The character is returned in Register A with the high order bit set to zero, with lower case a-z converted to upper case (if UCFLAG set), and is echoed to serial port A or B (depending on values in OCLSW/OPORT/OPORTA). All other registers are unchanged. The note under INCH8 regarding the "bell" character also applies to this routine.

INCHTE - \$FECD

Gets one character from serial port A or B (depending on values in ICLSW/IPORT/IPORTL). The character is returned in Register A with the high order bit set to zero, with lower case a-z converted to upper case (if UCFLAG set), and is echoed (if ECHOFL set) to serial port A or B (depending on values in OCLSW/OPORT/OPORTA). All other registers are unchanged. The note under INCH8 regarding the "bell" character also applies to this routine.

INIHEX - \$FECF

Gets one hexadecimal digit (via routine INCHTE) and returns the binary value in the lower 4 bits of Register A. If an invalid hexadecimal character is entered, the carry bit will be set on exit. All other registers are unchanged.

IN2HEX - \$FED1

Gets two hexadecimal digits (via routine INCHTE) and returns the 8 bit binary value in Register A. If an invalid hexadecimal character is entered, the carry bit will be set on exit. All other registers are unchanged.

IN4HEX - \$FED3

Gets four hexadecimal digits (via routine INCHTE) and returns the 16 bit binary value in Register X. Register A returns garbage. If an invalid hexadecimal character is entered, the carry bit will be set on exit. All other registers are unchanged.

- \$FED5

DO NOT USE -- this vector is reserved for future use. However, in order to be backwards compatible with ST-MON 1.01 which used this for the OUTCH8 routine, this vector currently points to the OUTCH routine below. If you wrote any programs to use the ST-MON 1.01 OUTCH8 routine, you should change them to use the ST-MON 2.04 OUTCH routine instead, as future releases of ST-MON will use this vector for other purposes.

OUTCH - \$FED7

Outputs one character to serial port A or B (depending on values in OCLSW/OPORT/OPORTA). The character to be output is passed in Register A. All registers are unchanged. The high order bit is transmitted "as-is", although it will be truncated if the serial port is set to only 7 data bits. The CTS line of the serial port selected must be asserted -- the character will not be transmitted until it is.

OUT1SP - \$FED9

Outputs one ASCII space character (via routine OUTCH). All registers are unchanged.

OUT1HX - \$FEDB

Outputs one hexadecimal digit (via routine OUTCH). The 4 bit binary value is passed in the lower 4 bits of Register A. Upon return, Register A contains garbage; all other registers are unchanged.

OUT2HX - \$FEDD

Outputs two hexadecimal digits (via routine OUTCH). The 8 bit binary value is passed in Register A. Upon return, Register A contains garbage; all other registers are unchanged.

OUT4HX - \$FEDF

Outputs four hexadecimal digits (via routine OUTCH). The 16 bit binary value is passed in Register X. Upon return, Register A contains garbage; all other registers are unchanged.

PCRLF - \$FEE1

Outputs a carriage return, line feed, and four nulls (via routine OUTCH). All registers are unchanged.

PDATA - \$FEE3

Outputs a character string (via routine OUTCH). The string must end with a byte containing hexadecimal 04 (which is not sent). The starting address of the string is passed in Register X. Upon return, the address in Register X points to the location after the \$04 byte, and Register A contains garbage. All other registers are unchanged.

 PSTRNG - \$FEE5

Identical to routine PDATA except that PCRLF is called internally before the string is output.

User Referable RAM Locations

Addr	Len	Name	Description
FE9F	1	ARTOPC	Current copy of DUART OPCR register contents
FEA0	1	ARTOPX	Current copy of DUART OPR settings
FEA1	1	BDRIVE	Drive number to boot from (0-3)
FEA2	1	DBLSTP	Double-step boot drive (\$00 = no, not \$00 = yes)
FEA3	2	STACK	Stack pointer value at initialization or when ST-MON re-entered
FEA5	1	TABSIZ	Maximum number of entries in secondary command table
FEA6	2	JMPADDR	Transfer address of user program (used internally)
FEA8	1	ICLSW	Switch: \$00 = use IPORT, not \$00 = use IPORTL
FEA9	1	OCLSW	Switch: \$00 = use OPORT, not \$00 = use OPORTA
FEAA	2	IPOINT	Port address for normal input (default = \$FF20 for Port A, use \$FF28 for Port B)
FEAC	2	IPOINTL	Port address for "L" command (default = \$FF20)
FEAE	2	OPOINT	Port address for normal output (def = \$FF20)
FEBO	2	OPOINTA	Port address for alternate output (def = \$FF20)
FEB2	1	UCFLAG	Flag re convert to uppercase (0=no, non-zero=y)
FEB3	1	ECHOFL	Non-zero = echo rec'd charac, zero = don't echo
FEB4	2	OFFSET	Address offset for "L" command
FEB6	1	ARTACR	Current copy of DUART ACR register contents
FEB7	1	ARTINT	Current copy of DUART IMR register contents
FEB8	1	VIAINT	Current copy of VIA interrupt enable flags
FEBD	2	BEGFLX	Address of first location of ST-MON when running FLEX
FEBF	2	CMDTAB	Address of secondary command table
FEE7	4	SWI3	Jump instruction to SWI3 handler
FEEB	4	SWI2	" " " SWI2 "
FEF3	4	FIRQ	" " " FIRQ "
FEF7	4	IRQ	" " " IRQ "
FEFB	4	SWI	" " " SWI "
FEFB	4	NMI	" " " NMI "

Note that the interrupt jump table allows the use of 4 byte instructions, so you can use a "JMP [\$hhhh]" instruction, if desired. ST-MON does not use any of these interrupts -- initially all entries in the interrupt jump table merely return control to ST-MON at its MONITOR entry point -- so all six are available for your own use. FLEX, however, reserves SWI3 and IRQ. Refer to the ST-2900 FLEX Conversion manual for more details.

Memory Map of ST-MON Version 2.04

```

-----
FFE0 - FFFF interrupt vectors, stepping stone code
FF60 - FFDF SAM registers, etc.
FF00 - FF5F I/O devices
FE6D - FEFF Message strings, variables, vectors
FD71 - FE6C I/O routines
FC7D - FD70 Main logic of ST-MON and commands M, F
FC59 - FC7C Command table for D, J, T, E, S, B, A, L commands
FA3B - FC58 Commands J, T, E, S, B, A, L
F85E - FA3A FLEX disk boot routines
F4FD - F85D OS-9 disk boot routines
C080 - F4FC User memory
C000 - C07F Stack
0000 - BFFF User memory

```

APPENDIX A - "CW?" prompt after powerup/reset

```

=====
Both the "C" (Coldstart) and "W" (Warmstart) replies result in the following:
a) initialize variables ARTACR, ARTINT, ARTOPC, ARTOPX, VIAINT to match the
   current values in the DUART and VIA registers because those chips have
   been reset.
b) switch the 6883 SAM to Map 1 and jump to the RAM copy of ST-MON via the
   "stepping stone" routine at $FFE5.

```

What the "C" option does in addition, prior to step (a) above, is copy ST-MON from EPROM to RAM. This includes resetting the values of all ST-MON variables and the interrupt jump table to their default values.

If you were running FLEX or some other other program that changed any of these variables or interrupt jumps from their original default values, and you got the "CW?" message because you pressed the system reset switch, and want to return to the wamstart location of the aborted program, use the "W" response. This will leave those values as they were before reset.

However, if you suspect that a wayward program has destroyed part of ST-MON, use the "C" response to restore ST-MON. You will then have to re-boot the operating system because the variables and interrupt jump table no longer contain the right values.

APPENDIX B - re Motorola S1/S9 block format

=====

Each "S1" record can contain from 1 to 30 bytes of data. Here is an example of one record:

S11301008E0132FE0137C603960AA1022705095A59

- a) The "S1" is the start-of-block code. Any carriage returns, line feeds, nulls, etc. that precede the "S1" will be ignored.
- b) The "13" is the byte count in hexadecimal (=19 in decimal). The count includes the 2 address bytes, the data bytes, and the checksum byte.
- c) The "0100" is the starting address (in hex) of where the block is to be stored in memory.
- d) The "8E01...095A" is the 16 bytes of data, in hex.
- e) The "59" is a one byte checksum, in hex. It is calculated by adding together the count byte, the two address bytes, and the data bytes, then taking the complement of the sum.

All the values in sections b to e are represented by 2 hex digits per byte. A record consisting of only an "S9" start-of-block code represents an end-of-file code.

The S1/S9 format is frequently used today to send data to EPROM programmers via an RS-232 serial interface.

APPENDIX C - Configuring the Serial Ports

Immediately after powerup or system reset, ST-MON initializes various attributes of both serial ports on the CPU board. While you can subsequently change ANY of these settings by having your software write the appropriate codes into the DUART registers, you also need to be able to change some of the initial settings (such as baud rate) to match your terminal's requirements. Two methods have been provided.

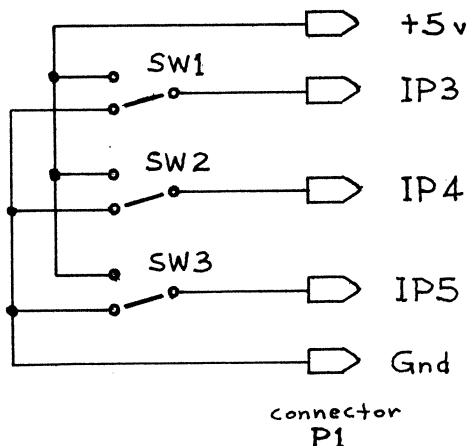
Method #1

The 2681 DUART has several normally unused input lines. As supplied, ST-MON 2.04 uses three of them, IP3, IP4, IP5 to let you select from one of 4 baud rates and 7 vs 8 data bits for port A. At this point you should look at Appendix E in the ST-2900 CPU manual. Note that there are 3 wiring blocks -- 4 pin, 20 pin, 1 pin. The IP3, IP4, IP5 lines need to be jumpered to ground or +5 volts as follows:

port A desired baud rate	IP4	IP3
9600 bps	gnd	gnd
19200 bps	gnd	+5
300 bps	+5	gnd
1200 bps	+5	+5

port A desired data bits	IP5
7 data bits	gnd
8 data bits	+5

We recommend that you connect IP3, IP4, IP5 (at the 20 pin wiring block) to any 3 of the 4 "spare" lines at the 4 pin wiring block. These spare lines, as well as ground and +5v lines, go to connector P1 where they can be routed via ribbon cable to three external SPDT switches. This allows you to quickly change these settings without messing with changing internal jumpers. The switches would be hooked up like this:



Method #2

If you need to use IP3, IP4, or IP5 for some other purpose, or if you need a baud rate other than any of the four shown above, or you want to change the default baud rate of port B (initialized to 300 bps), a second method has been provided. This will require access to an EPROM programmer. A new EPROM needs to be burned, an identical copy of the original except for the first 3 bytes which need to be changed as follows:

- 1) Byte 0 contains the code for the baud rate of port A. The codes for the most commonly used rates are:

\$11 = 110 bps	\$88 = 2400 bps
\$44 = 300 bps	\$99 = 4800 bps
\$55 = 600 bps	\$BB = 9600 bps
\$66 = 1200 bps	\$CC = 19,200 bps
\$FF = baud rate depends on setting of IP3/IP4 (see above)	
- 2) Byte 1 contains the code for the baud rate of port B. These values are the same as above except that a code of \$FF should not be used.
- 3) Byte 2 contains the code for selecting 7 or 8 data bits for port A:

\$00 = 7 data bits
\$01 = 8 data bits
\$FF = number of data bits depends on setting of IP5 (see above)

Changing any of these three bytes does NOT require updating the checksum (referred to in Appendix D) as the checksum calculation ignores these three locations.

Other Considerations

Determining whether to use 7 or 8 data bits can be confusing as not everyone attaches the same meaning to terms such as "no parity", etc. Use the following table as a guide:

ST-2900 Port A	Terminal
7 data bits	7 data bits, no parity bit, 1 stop bit
7 " "	7 data bits, no parity bit, 2 stop bits
7 " "	7 data bits, "mark parity" bit, 1 stop bit
7 " "	7 data bits, "mark parity" bit, 2 stop bits
7 " "	8 data bits (high order bit = 1), no parity bit, 1 stop bit
7 " "	8 data bits (high order bit = 1), no parity bit, 2 stop bits
8 data bits	8 data bits (high order bit = 0), no parity bit, 1 stop bit
8 " "	8 data bits (high order bit = 0), no parity bit, 2 stop bits
8 " "	8 data bits, "mark parity" bit, 1 stop bit
8 " "	8 data bits, "mark parity" bit, 2 stop bits
8 " "	7 data bits, "space parity" bit, 1 stop bit
8 " "	7 data bits, "space parity" bit, 2 stop bits

ST-MON initializes both serial ports to "no parity bit" and 2 stop bits and sets port B to 8 data bits. In addition, both ports have the "CTS enable Tx" bit set. This means that a character will not be transmitted unless the CTS input line (IP0 or IP1) for that port is "asserted".

For more detailed information study the 2681 DUART data sheet (in the ST-2900 CPU manual) and the source code of ST-MON.

APPENDIX D - Calculating a New Checksum

Any time so much as one byte (except for the first three bytes) of the ST-MON EPROM is changed, the checksum must also be changed if you don't want to see the "M1" error message each time you power up or reset the system.

The following example assumes you are running the FLEX (or STAR-DOS) operating system and will be programming a 4K byte EPROM (2732 or 2732A, 450 nsec. or faster). If you are using a 2K byte (2716) or 8K byte (2764) EPROM instead, several values mentioned in steps 1, 3, and 4 need to be different:

2732	2716	2764
----	----	----
AFFF ->	A7FF	BFFF
AFEE ->	A7EE	BFEE
AFEF ->	A7EF	BFEF

- 1) Fill memory with \$FF:

```
+++MON
=A A000 AFFF FF
=F W
```

- 2) Load new version of ST-MON into RAM at \$A000:

```
+++GET STMON204.BIN
```

- 3) Enter and run a machine language program to calculate the new checksum:

```
+++MON
=M 1000                (to load checksum routine)
                      108EA003
                      8E5555
                      E6A0
                      3A
                      108CAFEE
                      26F7
                      BFAFEE
                      39
=M J 1000              (to execute the routine)
=M AFEE                (to display new 2 byte checksum)
=F W
```

- 4) Re-assemble ST-MON with new checksum value at \$AFEE/\$AFEF, then burn new EPROM.

APPENDIX E - Testing RAM Occupied by ST-MON and its Stack

=====

One disadvantage of a monitor program in RAM instead of EPROM is that you cannot do a memory test of that part of RAM occupied by the monitor itself. That is, you can't unless the program is relocatable. ST-MON is written in position independent code (PIC) so can be moved to wherever is desired, and run from there. However, four things need to be considered before relocating ST-MON:

- 1) The location of the stack (default is \$C000-\$C07F) may need to be changed
- 2) If user programs that call ST-MON's I/O routines (via the address vectors at \$FEC1-\$FEE6) will be run after ST-MON is relocated, you have two choices:
 - a) if you want to test the RAM at \$FEC1-\$FEE6, you will have to modify the user programs to use the address vectors in the relocated copy of ST-MON instead. Also, the contents of those relocated vectors must be changed to point to the new ST-MON locations.
 - b) if you don't want to test the RAM at \$FEC1-\$FEE6, you do not need to modify the user programs, but the address vectors at \$FEC1-\$FEE6 must be changed to point to the new ST-MON locations.
- 3) If the area occupied by the interrupt jump table (\$FEE7-\$FEFF) is to be tested, no interrupts (hardware or software) may be used, as this jump table cannot be moved without changing the EPROM interrupt vectors (\$FFF2-\$FFFF). On the other hand, if interrupts will be used, the jump instructions must be updated to point to the new ST-MON locations, and \$FEE7-\$FEFF should not be included in the memory test.
- 4) The disk boot commands ("D F", "D OC", "D OS") should not be run from the relocated ST-MON, as the boot process and/or the booted operating systems require ST-MON to be in its usual location.

The following instructions will relocate ST-MON from \$FXXX to \$AXXX, and move the stack to \$BF80-\$BFFF. The address vectors and interrupt jump table will not be updated, so no interrupts should be used. Once the memory testing is done, the system must be reset (answer "CW?" prompt with "C") to restore ST-MON to its normal location.

- 1) Copy ST-MON from \$FXXX to \$AXXX as follows:
=B F400 FEFF A400
- 2) Update variable STACK in the new copy of ST-MON:
=M AEA3
BFFF
- 3) Use the "M" command to display the contents of location "MONITOR" (\$FEC3/\$FEC4).
- 4) Use the "J" command to jump to the new copy of ST-MON. The address you specify is the value you displayed in step #3 above, except change the first hex digit from an "F" to an "A" (eg. \$FC7D becomes \$AC7D).
- 5) Now that the relocated copy of ST-MON is being used instead, you can use the "T" command to test memory locations \$C000-\$FEFF.

APPENDIX F - "Stepping Stone" Code and ROM-DISK's

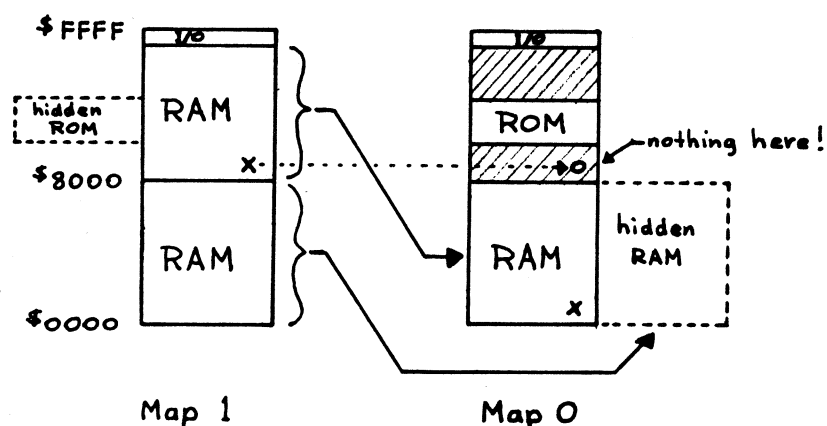
The ability of the 6883 SAM chip to switch EPROM in and out of the 6809 microprocessor's 64K addressing range is a powerful feature. Operating systems such as FLEX, and especially OS-9, benefit from having 63.75K of contiguous RAM. Yet within microseconds you can also access large amounts of non-volatile data and programs stored in EPROM.

But switching between the 6883 SAM chip's two memory maps can be tricky. If you're not careful, your program could "have the rug pulled out from under its feet and fall flat on its face", so to speak.

For purposes of illustration, let's say you are running FLEX and want to put several often used utilities such as P, CAT, COPY, LIST, DELETE, and RENAME into EPROM instead of constantly loading them from disk. Since ST-MON only occupies approx. 3K of the EPROM on the ST-2900 CPU board, a 2764 (8K bytes) chip has room for almost 5K of such routines.

We won't go into the details of implementing such a ROM-DISK system, whether it is structured similar to a RAM-DISK, or modified from RESIDENT (a utility written by Leo Taylor). Suffice it to say that if you want to run those stored utilities without first modifying them, they will somehow have to be copied from EPROM to their normal load address in RAM, every time you call them. Some, like COPY, load into low memory (\$0000 and up), while others, like CAT, use FLEX's utility command space (\$C100-\$C6FF).

The code to retrieve these programs from EPROM will likely reside somewhere in memory areas \$BXXX, \$EXXX, or \$FXXX, as all other memory is reserved for FLEX itself, or programs that run under it. When the ROM-DISK code switches you from the all-RAM memory map to the ROM/RAM map to access the normally hidden EPROM, the RAM containing the ROM-DISK code suddenly jumps down to the other half of the memory map. The CPU tries to execute the next instruction of the switching code, only to find it is no longer there! The diagram below illustrates what happened (the "x" represents the location of the instruction that caused the memory map switch):



What do we do now? I hope you don't mind an unusual, but hopefully illuminating analogy. Imagine yourself standing on a small rug. Your best "friend" comes along and, succumbing to a sudden impulse, tries to pull the rug out from under your feet (literally!). In that same split-second you notice a chin-up bar above you. Quick as a flash you grab the bar and lift

yourself up off the rug, just as your friend jerks the rug. Then, realizing you've outsmarted him, he lets go of the rug, which is now lying an arms-length away. Now you swing over, jump, and land on the same rug -- but in its new location. By momentarily grabbing onto a fixed object and lifting your feet off the moving area, you foiled the prank, and ended up still standing upright on the rug.

ST-MON 2.04 has two such "chin-up bars" -- call them "stepping stones" if you prefer. The top 32 bytes of the EPROM on the CPU board occupy addresses \$FFE0-\$FFFF, regardless of which memory map you are in. Two tiny "stepping stone" routines occupy part of this immovable area, the first one for switching from memory map 1 to map 0 (all-RAM to ROM/RAM), the second for going the opposite direction. They look like this:

```

                ORG  $FFE0

FFE0 B7FFDE  ROMMAP  STA  $FFDE      switch to ROM/RAM map
FFE3 6EC4                JMP  0,U        jump to next instruction
FFE5 B7FFDF  RAMMAP  STA  $FFDF      switch to all-RAM map
FFE8 6EC4                JMP  0,U        jump to next instruction

```

To call these routines, load register U with what will be the address of the next instruction to execute after the memory map is switched, then jump to \$FFE0 or \$FFE5. The stepping stone code switches the map for you, then jumps to the specified address. Before calling "ROMMAP" make sure that the SAM chip's "PAGE" register (\$FFD4/\$FFD5) is set as desired.

The code for the preceding example might look like this:

```

ROMMAP EQU  $FFE0
RAMMAP EQU  $FFE5
                ORG  $E000
BUFFR   RMB  256      intermediate sector buffer
ROMADR  FDB  $AC00    contains address of desired sector in ROM

START    LDX  ROMADR    get address of sector in EPROM
         LDY  #BUFFR-$8000 load address of buffer in RAM
         PSHS CC        disable interrupts
         ORCC #50
         LDU  #MOVESC-$8000 load switched address of routine
         STA  $FFD5      ensure page 1 for map 0
         JMP  ROMMAP    switch map and jump to routine

MOVESC   CLRB          loop to move 256      \
NEXTTB   LDA  ,X+      . bytes from ROM      |
         STA  ,Y+      . to RAM buffer      | this code executed
         DECB          > at $6XXX
         BNE  NEXTTB   | ($EXXX-$8000)
         |
         LDU  #RETURN
         JMP  RAMMAP    switch map          /
RETURN   PULS CC      restore previous interrupt flags
         ...          continue processing

```

For more information on the SAM chip's memory maps and registers, you should obtain the complete Motorola data sheet for the "SN74LS783/MC6883".

APPENDIX G - Updating DUART Registers ACR, IMR, OPCR, OPR

The 2681 DUART chip has many write-only registers. Only one of these (OPR) allows you to directly change one bit without affecting other bits in the register. In order to change individual bits in the other registers, a copy of the current contents of the register must be stored elsewhere. The four DUART registers that typically require bit by bit manipulating are "ACR" (\$FF24), "IMR" (\$FF25), "OPCR" (\$FF2D), and "OPR" (\$FF2E/\$FF2F).

While the OS-9 operating system is active, these registers and their corresponding copies in memory are maintained by special subroutines. For more details refer to the "ST-2900 OS-9 Conversion Package" manual in the section "Additional Information for Advanced Programmers".

If no operating system has yet been booted, or if you are running FLEX (or STAR-DOS), the register copies in memory are in ST-MON's ARTACR, ARTINT, ARTOPC, and ARTOPX locations. Although the ST-2900 FLEX Conversion Package provides subroutines to maintain ACR/ARTACR and IMR/ARTINT, improved subroutines to maintain all four registers and locations are shown below.

**** NOTE --** the "ACR", "IMR", "OPCR", "OPR" registers in the 2681 DUART and the ARTACR, ARTINT, ARTOPC, ARTOPX locations in ST-MON should not be updated directly by user programs. ALWAYS use one of the 7 subroutines shown below. Unfortunately, the original ST-2900 FLEX Conversion Package violates this rule; it doesn't update ARTOPX, and directly accesses the OPR register. Future releases will correct this, but if you have the old version (ie., the FLEX Conversion manual is dated 1984), do NOT use the SETOPR routine while you are running FLEX.

The subroutines can either be incorporated into each user program as needed, or stuffed into some unused corner of memory for use by all programs. The seven entry points are as follows:

Name	Description
DACRON	set on bits in DUART ACR register
DACROF	set off bits in DUART ACR register
DINTON	set on bits in DUART IMR register
DINTOF	set off bits in DUART IMR register
DOPCON	set on bits in DUART OPCR register
DOPCOF	set off bits in DUART OPCR register
SETOPR	set bits in DUART OPR register

To call any of the first six routines, load register A with a value where bits to be turned on or off are 1's, bits not to be changed are 0's, then do a subroutine call. For example, to turn off bits 4 and 5 in the DUART's IMR register:

```
LDA #%00110000 disable RxRDYB & TxRDYB interrupts
JSR DINTOF
```

and to turn on bit 2 in the DUART's ACR register:

```
LDA #%00000100 enable delta IP2 interrupt
JSR DACRON
```

Upon return, register A contains the new value the DUART register has just been updated with; all other CPU registers are unchanged.

The SETOPR routine has a different calling sequence. If the data sheet for the 2681 DUART confuses you by describing different register addresses for setting vs resetting output bits, and even has the audacity to suggest you remember that the output lines OP0-OP7 are the complement of the values in the internal OPR register, relax! Just tell the SETOPR routine which output lines you want changed and what the output levels should be -- it sorts out the rest for you.

To call SETOPR, load register B with a mask re which bits are to be changed (1=change, 0=no change) and register A with the desired output data (bits specified as "no change" by the mask are "don't cares" here). For example, if you want to change OP7 to "1", OP6 to "0", and OP2 to "1":

```
LDB #%11000100
LDA #%10000100
JSR SETOPR
```

On exit, all CPU registers are unchanged except for register A.

```
*****
*
* SUBROUTINES TO UPDATE DUART'S ACR, IMR, OPCR, OPR REGISTERS *
*
*****
```

```
ARTACR EQU $FEB6
ARTINT EQU $FEB7
ARTOPC EQU $FE9F
ARTOPX EQU $FEA0
DUART EQU $FF20
```

```
*
* UPDATE THE 2681 DUART 'ACR' REGISTER
* SETTING INDIVIDUAL BITS ON OR OFF
* IN - A the bits to be affected are 1's, others are 0's
* OUT - A new ACR contents
* CC,B,X,Y,U unchanged
*
```

```
3401 DACRON PSHS CC
1A50 ORCC #$50 disable interrupts
12 NOP
BAFEB6 ORA ARTACR
2008 BRA DA50
3401 DACROF PSHS CC
1A50 ORCC #$50 disable interrupts
43 COMA
B4FEB6 ANDA ARTACR
B7FEB6 DA50 STA ARTACR save new register contents
B7FF24 STA DUART+4 update DUART register
3581 PULS CC,PC restore interrupts and return
```

```
*
* ENABLE/DISABLE INDIVIDUAL INTERRUPT SOURCES
* FROM THE 2681 DUART
* IN - A the bits to be affected are 1's, others are 0's
* OUT - A new IMR contents
* CC,B,X,Y,U unchanged
*
```

```

3401  DINTON  PSHS  CC
1A50          ORCC  #50
12          NOP
BAFEB7      ORA   ARTINT
2008        BRA   DI50
3401  DINTOF  PSHS  CC
1A50          ORCC  #50
43          COMA
B4FEB7      ANDA  ARTINT
B7FEB7  DI50  STA   ARTINT
B7FF25      STA   DUART+5
3581        PULS  CC,PC

```

*

```

* UPDATE THE 2681 DUART 'OPCR' REGISTER
* SETTING INDIVIDUAL BITS ON OR OFF
* IN - A the bits to be affected are 1's, others are 0's
* OUT - A new OPCR contents
*       CC,B,X,Y,U unchanged
*
```

```

3401  DOPCON  PSHS  CC
1A50          ORCC  #50
12          NOP
BAFE9F      ORA   ARTOPC
2008        BRA   D050
3401  DOPCOF  PSHS  CC
1A50          ORCC  #50
43          COMA
B4FE9F      ANDA  ARTOPC
B7FE9F  D050  STA   ARTOPC
B7FF2D      STA   DUART+13
3581        PULS  CC,PC

```

*

```

* SET DUART OUTPUT LINES OP0-OP7
* IN - A value to set output lines to (complement of OPR)
*       B mask re which bits are to be affected
* OUT - CC,B,X,Y,U unchanged
*       A unchanged except "don't care" bits changed to zeros
*
```

```

3407  SETOPR  PSHS  CC,A,B
A462          ANDA  2,S      mask out "don't care" bits
A761          STA   1,S
1A50          ORCC  #50      disable interrupts
53           COMB          calc new OPn values
F4FEA0       ANDB  ARTOPX
EA61         ORB   1,S
F7FEA0       STB   ARTOPX   save new output line values
A862         EORA  2,S      calc which bits to set in OPR
E661         LDB   1,S      retrieve which bits to reset in OPR
FDF2E       STD   DUART+14  update DUART OPR register
3587        PULS  CC,A,B,PC  restore interrupts and return

```

APPENDIX H - Structure of the boot sector(s) on a FLEX disk

=====

The first two sectors on track zero of a FLEX format disk are reserved for a bootstrap loader program.

On many other systems that run FLEX, the monitor command to boot FLEX is a very short routine that merely reads the 1 or 2 sector bootstrap loader program from track zero and passes control to it. The bootstrap loader program, in turn, does the actual loading of FLEX.SYS into memory.

ST-MON's "D F" command, on the other hand, is intelligent enough to load FLEX.SYS (or whatever program is being booted) all by itself. All it requires sector 1 of track zero to contain are 4 values:

byte #	description
-----	-----
0-1	(not used)
2	number of sectors per track (each side) of tracks 1 and up, usually \$0A (10) for single density, and \$12 (18) for double density
3	code indicating if disk is recorded on one or two sides \$00 = single sided; \$FF = double sided
4	code indicating in what density tracks 1 and up of the disk are recorded: \$00 = single density, \$FF = double density
5-6	link address as set by the FLEX "Link" command
7-255	(not used)

Bytes 2-4 are unique to the ST-2900, and are set by the FORMAT program that is part of the ST-2900 FLEX Conversion Package. FORMAT / INIT / NEWDISK programs from other sources do NOT set these three bytes to the values needed by ST-MON.

....