Tucson Amateur Packet Radio
8987-309 E. Tanque Verde Rd #337
Tucson, Arizona • 85749-9399
Office: (817) 383-0000 • Fax: (817) 566-2544
Non-Profit Research and Development Corporation

DSP-93

DSP-93 Programming Guide

Contributions by Ron Parsons, W5RKN, Don Haselwood, K4JPJ, and Bob Stricklin, N5BRG.

Introduction

Please send all corrections to Ron Parsons at w5rkn@amsat.org. If a topic relevant to DSP-93 programming is not covered, please feel free to write a draft and submit it for the next revision.

This purpose of this guide is to assist potential programmers new to the TAPR/AMSAT DSP-93 environment insight into the tools and techniques available when developing for the DSP-93. TAPR maintains an active Internet site that supports the DSP-93 project.

FTP: TAPR maintains an ftp site at `ftp.tapr.org`. Updates and archive material relevant to the DSP-93 can be obtained via anonymous ftp from the directory `/tapr/dsp93.`

File Request: You can get copies of all past messages distributed on the dsp-93@tapr.org list by sending mail to '`listserv@tapr.org`', subject of '`list`', message of '`index dsp-93`'. This message will request a listing of all files in the dsp-93 mail archive. To request a specific month, send mail again to '`listserv@tapr.org`', subject of '`request`', message of '`request dsp-93 filename`', where filename is the name of the file in the dsp-93 area you want to request (i.e., nov.94).

Listserv Mail Group: You can join the DSP-93 listserv by sending mail to '`listserv@tapr.org`', subject of '`list`', message of '`subscribe dsp-93 First_Name Last_Name`'. This will subscribe you to the mail list.

Web: You can find the DSP-93 web page on:
`http://www.tapr.org`

Books to Read

Any introduction regarding DSP-93 programming must begin with discussing how to locate and secure reference materials. The first step is to locate your local Texas Instruments distributor and call them. Local distributors have been known to give free access to their literature room. Books that you should be looking for include:

- **TMS320C2x User's Guide By Texas Instruments; Document # SPRU014C**
  This book covers the TMS320C25 DSP chip used in the DSP-93. It covers the chip's electrical properties, memory models, interrupt processing, and, of course, the instruction set.

- **Linear Circuits; Data Conversion, DSP Analog Interface, and Video Interface; Data Book Volume 2 By Texas Instruments; Document # SLYD004A.**
  This book covers the TLC32044CN AIO chip used in the DSP-93. It covers the chip's electrical properties, configuration, etc.

- **Digital Signal Processing Applications with the TMS320 Family; Theory, Algorithms, and Implementations**
  Volume 1 Document # SPRA012A
  Volume 2 Document # SPRA016
  Volume 3 Document # SPRA017
  These books cover various DSP algorithms which may or may not be useful to you.

- **Digital signal processing with the TMS320C25**
  Chassaing, Rulph, published 1990 by Wiley, New York 464 pg
  ISBN 0471510661

- **Digital Signal Processing: A laboratory approach using PC-DSP.** Alkin, Oktay, published 1994 by Prentice-Hall, Inc.
  ISBN 0-13-328139-6
  An introduction to Digital Signal Processing techniques. Comes with a DOS program which can compute FIR coefficients, among other things.

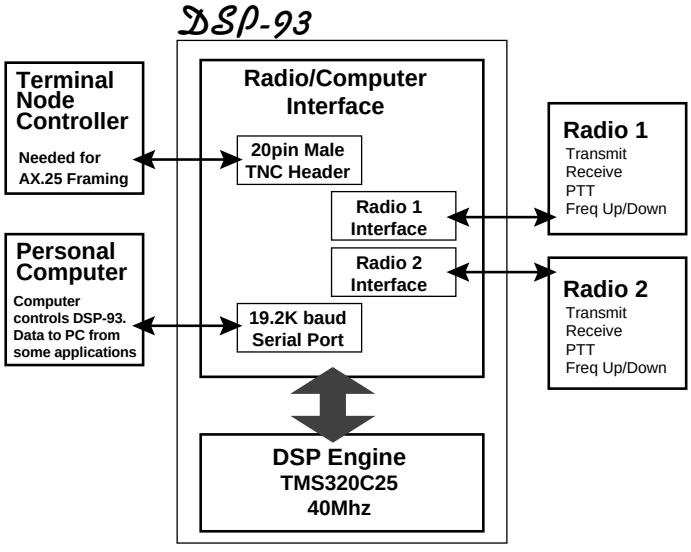
Theory of Operation of DSP-93

DSP-93 Design

The TAPR/AMSAT DSP-93 is designed to provide radio amateurs the wonderful capabilities of Digital Signal Processing in a stand-alone low-cost design. Not just limited to one mode, the DSP-93 can support data, audio, and video modes with the proper software.

The basic system includes a DSP engine board and a radio/computer interface board. The DSP Engine, contains the TMS320C25 DSP, 32K by 16 bits of program and data memory - upgradable to 64K, the clock circuitry (40MHz) and some programmable array logic for system I/O. The Radio/Computer Interface Board, contains two eight pin female mini-DIN connectors for radio interfacing. Incoming radio signals pass through a voltage divider to establish the initial levels, then through an eight channel multiplex chip. The multiplex chip then feeds the single A/D input with either of the radio inputs or one of the six auxiliary inputs. The Texas Instruments TLC32044 Analog I/O chip is used to sample and update the input signal at a rate of up to 45K operations per second and includes aliasing filters. This board also communicates with your computer at speeds up to 19.2K baud using a serial connection and, with special programming, this can be increased to the maximum rate attainable by a 16C550 and your computer.

The modular design of the DSP-93 allows for either of these boards to be replaced with future boards designed for any number of unique applications. It's sort of like adding a new application card to a PC without redesigning the complete PC. The following block diagram shows how the DSP-93 is interfaced.



Code Development

A low cost shareware assembler, TASM TMS320-25 Assembler, is available for code development. A copy of the shareware version of this assembler is included with the DSP-93 kit. If you use this assembler, please pay the small shareware fee.

Memory Map

The following memory map exists after you have entered a 'G' command from the Monitor. The 'G' command is executed after a program download using DSPLOAD.EXE or the windows program D93WE. The unit is operating entirely in static RAM and in the high speed mode at this point.

```
      PROGRAM                 DATA

0000 Interrupts         0000 TMS320C25
     and                      MEMORY MAPPED
     Reserved                 REGISTERS
001F ————                0005 ————
                              Reserved
                         0060 ————
                              On DSP Block B2
                         007F ————

     Reserved
     For Monitor         01FF ————
                         0200 ————
                              On DSP Block B0
                         02FF ————
                         0300 ————
                              On DSP Block B1
                         03FF ————
                         0400 ————
                              Page 8 Data Memory
0FFF ————
1000 TINT
1002 RINT
1004 XINT
1006 TRAP
1008 ————

     User Program            User Data
     Memory Space            Memory Space

 FFFF ————               FFFF ————
```

Memory Mapping

[Don Haselwood, K4JPJ]

The purpose of the following is to provide some detail as to how the DSP-93 implements memory in conjunction with the TMS320. Integral to this is a scheme for providing a Monitor in EPROM and high/low speed operation.

The Harvard architecture used in the TMS320 is quite different from the typical microprocessor. There are two memories utilized during one instruction—data and program. Until experience is gained working with this architecture it is easy to forget this basic principle. There are two memories active at the same time. By having two memories, a single instruction can load a word out of program memory and do something with a word out of data memory, all within the same cycle. It facilitates implementing filters and other digital signal processing algorithms. For example, stepping down a table of constants, such as filter coefficients and doing a multiply and add to accumulator with a data array of signal values can be accomplished at full machine cycle speed with no overhead for instruction fetches, nor double accessing of a single memory.

Program and data memory can change, and keeping track of "what is which" is needed. How this is done is largely a function of how the DSP-93 design uses the TMS320. Therefore, the TMS literature will not give the whole story necessary to understand the DSP-93. Various clues are given in the DSP-93 docs and literature, but this tries to bring it together into one place. The following details show how memory is mapped under various conditions.

The DSP-93 has a number of physical memories. Some memory is internal to the TMS320C25 chip, and other is external. The tables below outline the physical memories, as well as the conditions which determine whether they are used by the TMS320C25 as program or data. Following each table is a discussion of how the configuration is organized and used.

Internal Memory (TMS320C25)

Label	Can be configured as	Size
B0	Data or program *	0100h
B1	Data only	0100h
B2	Data only	0010h

* B0 is in data mode after a hardware reset, Monitor reset, or CNFD instruction. It is program memory after a Monitor 'G' command, or CNFP instruction.

External Memory

How the external memories are configured depends on the state of the XF bit. Hardware reset turns the XF bit ON, as will the SXF instruction. RXF turns it OFF. SRAM U104, U105, U110, and U111 come with the basic kit and provide two 32K memories (one program and one data). Four more SRAM IC's will raise the amount to 64K for both program and data memory.

IC number	XF = ON(1) "Slow speed"		XF = OFF(0) "Fast speed"	
SRAM				
U104, U105	Data	0000 - 7FFF(2)	Prog	0000 - 7FFF(2)
U106, U107	Data	8000 - FFFF	Prog	8000 - FFFF(3)
U108, U109	N.A.	.... -	Data	0000 - 7FFF(2)
U110, U111	N.A.	.... -	Data	8000 - FFFF
EPROM				
U102, U103	Prog	0000 - 7FFF(2)	N.A.	.... - ....

(1) - XF line/bit is ON after a hardware reset, after a Monitor reset command ('R'), or after a SXF instruction. RXF turns the bit off, and it is also turned off after a Monitor 'G' command.

(2) - Data addresses below 0400h access internal memory in the TMS320 and therefore are not available for use in the external memory.

(3) - Addresses FF00h - FFFFh access internal memory, B0, when B0 has been configured as program memory (normally after a Monitor 'G' command).

Note that all memory is organized as 16 bit words, and not 8 bit bytes. Address 1000h addresses a 16 bit word, and address 1001h addresses the next 16 bit word. (In a byte oriented machines such as the PDP-11, which can address 16 bit words or 8 bit bytes, the address 1001h would address the second byte of a word, thus making 16 bit word addresses, 1000h, 1002, 1004h, etc.—such is NOT the case here). Along the same lines remember that the accumulator is 32 bits long so be careful about unintended sign extension.

Internal memory, B0, is configured to the data mode after a hard reset. Also, the Monitor in EPROM configures B0 to data when a 'R' (reset) command is executed, and to program when a 'G' command is executed. Switching B0 modes is done with the instructions CNFP and CNFD, (set program, and set data, respectively), or the hardware reset that switches it to data.

External memory is switched via the XF line out of the TMS320. This line is controlled by the XF bit. A hardware reset sets this bit high. The bit can be set/reset by the instructions SXF/RXF, respectively.

When the XF bit is high, such as after power-up reset, the EPROM, which contains the Monitor programs, is active as program memory. This makes it possible to get the machine running with something intelligent. Since inexpensive EPROM is slow, the XF also selects a low speed mode (divides the 40 MHz clock by two, or four depending on jumper J100). In slow speed the processor is slow enough to accommodate the EPROM. XF also makes external memories U104,5,6,7 switch to data mode and U108,9,10,11 not accessible. This arrangement allows the EPROM Monitor to load program into data memory (U104,5,6,7 and B0). This switching is necessary since the TMS320 does not execute instructions which store anything into program memory; the downloaded program is placed into the DSP-93 as data.

With the Monitor program executing out of EPROM, the usual step is to download a program from a general purpose computer. The program being downloaded is stored as data, in data memory, which is U104,5,6,7 at this time. Upon completion of the downloading, the general purpose computer issues a 'G' command which causes the DSP-93 Monitor to turn off the XF bit. This switches U104,5,6,7 from data memory to program memory. The Monitor also jumps to location 1008h (of program memory) to start the program that was just loaded (into what was data memory).

With XF low, program normally runs out of external SRAM, U104,5,6,7, using U108,9,10,11 for data storage. B0, B1, and B2 can also be used. B0 as stated before can be configured either as data or program, though remember that the Monitor sets it to program after the loading process completes and a 'G' command is given. Operating out of internal memory is somewhat faster than external memory and may be needed for time critical operations. The TMS320C2x User's Guide shows timings for instructions according the memory combination being used.

Since B0 can be switched between data and program, it can be loaded with a program which can be executed. During the normal program downloading process, B0 is configured as data, so it can be loaded with program no differently than U104,5,6,7. The addresses where the program loads of course must be correct. If the program has code which was preceded by .ORG directive with the address so that the code assembles in locations 0200h - 02FFh, the loader will put that code into B0. Some code .ORG'ed to 1000h is also needed. When the loading process completes, the Monitor does a 'G' command and jumps to 1008h. The program at 1008h can then jump to B0 and execute the code which was loaded. When B0 is configured as program, it is no longer at locations 0200h - 02FFh, but occupies FF00 - FFFFh. Therefore, the code assembled at 0200h - 02FFh must be capable of executing properly when moved to locations FF00h - FFFFh. The jump from the program in U104,5,6,7 will be to FFxyh, if the beginning of the code loaded into B0 is 02xyh.

External memory is not accessed for data addresses below 0400h, as these are reserved for the TMS320. Also, FF00h - FFFFh of program memory is not accessed when B0 is configured as program.

Note that some addresses are really registers within the TMS320, such as locations 0 and 1 which are used to load/receive the serial shift register data to/from the AIO chip. The TMS manual covers these in detail.

External memory, (prog, fast), 0400h - 1000h holds the Monitor which is used when in the fast mode (since the Monitor in the EPROM is only available in the slow mode). If these locations are blasted, such as with a Monitor Fill command, the Monitor is lost and a hard reset is required.

The TMS320 cannot load and store (i.e. move) data from program memory to program memory. Data-to-data memory can be accomplished, as well as program-to-data and data-to-program. Therefore is not likely that a runaway program will blast the Monitor stored in 0400h - 1000h.

The Monitor uses B0 during the transition from EPROM to the SRAM or from low speed operation to high speed. It is not possible to load the entire 256 words with program, or Fill it via the Monitor. If a reset command is executed by the Monitor, locations 0200h - 0275h are overwritten with "stuff" from the Monitor, wrecking what might have been downloaded into B0. As long as a Monitor reset does not occur between the loading of B0 and utilization of B0 (either as data or program), then the full page can be used. Otherwise, only those locations not used by the Monitor can be used (0276h - 02FFh).

The DSP-93 Monitor

To use the Monitor, you must have the DSP-93 connected to a computer using a terminal interface program, the D93WE windows program or the DSP-93Control application for the Macintosh. You should have your computer set for 19200 baud with 8 data bits, 1 stop bit, and no parity. If this is all in place you will see the asterisk (*) when you power up or press reset.

DSP-93 Firmware - Monitor Operation

When the DSP-93 is powered up the firmware Monitor takes control of the unit. The Monitor conditions the TMS320C25 and then begins polling the serial data link looking for single character instructions to execute. The action taken by the Monitor during each operation will be explained in more detail here.

After initialization is complete, the Monitor enters a polling loop checking the serial port for an input character. Acceptable input characters are; ?, A, D, F, G, H, J, L, M, P, R, S, and T. The Monitor is not case sensitive so the lower case of all the above also applies. An entry of '?' will bring a listing of the Monitor version and the commands as shown here;

```
*?
DSP-93 TAPR / AMSAT REV 2.17
Copyright <c> 1995 by Tucson Amateur Packet
Radio Corporation.

A-AR REGISTERS
D-DUMP MEMORY
F-FILL MEMORY
G-FLIP & RUN PROGRAM @ 1008h
H-INTEL LOADER HIGH BITS
J-JUMP TO XXXX & RUN
L-INTEL LOADER  LOW BITS
M-MODIFY WORD
P-FIRMWARE PROGRAMS
R-RESET
S-SHOW WORD
T-TEST
```

Let's review the operation of each of the commands in order. An example will be shown when applicable for the command. Also note the results of the command will vary depending on the state of the DSP-93 or previous commands issued.

A-AR REGISTERS

This command displays the value contained in the eight 16 bit registers on the TMS320C25. The first register is used by the Monitor and so the value in register zero will normally be 0400 hex. This is what you should see when you enter 'A' after power up.

```
*a 0400 0ECD FFFF FFFF FFFF FFFF FFFF 0AFF
```

D-DUMP MEMORY

The dump memory command displays the specified block of program or data memory. The TMS320C25 uses a Harvard Architecture memory format which means it has separate 16 bit memory for program space and data space. Therefore when you are working with memory you must specify whether you want to use program or data memory. Monitor commands use a 'P' for program and a 'D' for data memory. After entering a 'D' for DUMP and a 'P' or a 'D', you enter the hex starting address and the ending address of the memory block you want to see. The Monitor is not very forgiving so you must enter exactly four hex characters a space and four more hex characters. The memory contents will then be presented in block form with one 16 bit data address and eight 16 bit data values on each line. This will continue throughout the range specified. The last block may run beyond the ending value you specified to round out a block of eight values. When you are operating the DSP-93 in the low speed mode memory

dumps for program memory will be the contents of the EPROM and the scratch pad on the TMS320C25 and the data memory will come from the first bank of static RAM, U104 through U107. When you switch to the HIGH speed mode, memory dumps will come from the static RAMs U104 through U107 for program and from U108 through U111 for data.

```
*d  [P]ROGRAM or [D]ATA
p
    XXXX XXXX
    1000 1010
1000  FF80 0476 FF80 0476 FF80 0476 FF80 0476
1008  C808 CA40 6060 E760 FF80 0530 FF80 1008
1010  FF80 10AF FF80 10E8 FF80 1008 CE01 C808

*d  [P]ROGRAM or [D]ATA
d
    XXXX XXXX
    1000 1050
1000  FF80 0476 FF80 0476 FF80 0476 FF80 0476
1008  C808 CA40 6060 E760 FF80 0530 FF80 FFFF
1010  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
1018  0101 0202 4800 2000 0102 0220 002A 020A
1020  0800 0041 0040 8025 0000 0000 0100 0004
1028  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
1030  FFDF FFF7 FFFF FFFF FFFF FFFF FFFF FFFF
1038  035C 0040 0000 0000 8000 4000 0003 0001
1040  0001 8000 0041 4940 6200 0200 0003 0100
1048  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
1050  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
```

F-FILL MEMORY

The Fill memory command works like the data dump command except in this case you are filling memory with a 16 bit hex value. If you write over any program areas you will kill the Monitor. The reset switch will restore order.

```
*f  [P]ROGRAM or [D]ATA
p
    XXXX XXXX XXXX
    2000 2200 fe50

*f  [P]ROGRAM or [D]ATA
d
    XXXX XXXX XXXX
    3000 3300 fe60
```

G-FLIP & RUN PROGRAM @ 1008h

The 'G' command is used after you have loaded a program and you are ready to run it. The details of this command are covered in the software development sections of this document. For now you need to know the 'G' command will cause the DSP-93 to began executing the program loaded in memory. The Flip referred to here means that the Monitor is flipping from the EPROM over into the static RAM.

H-INTEL LOADER HIGH BITS

This is an Intel style hex loader for placing bits in data memory. To execute properly the DSP-93 must be in the LOW speed state. I called it Intel style because Intel loaders normally handle only 8 bit code. The same concepts are used however. After receiving the 'H' command, the DSP-93 is expecting data for the upper 8 bits of the 16 bit words. The data is expected in the following sequence; count (8 bits), byte address (16 bits) (the byte address is twice the word address), Intel hex code command (8 bits), some quantity (count) of data (8 bits) and finally an 8 bit checksum. Here are a few typical lines of a loadable file:

```
:08000000FF04FF04FF04FF04EC
:0E002000FF04FF0AFF0AFF04C8CA60E7FF04DE
:20420000FF04C8CA60FE05CA60FE05CA60FE05FE05FF04CECECE8CA60C885CE5D004D024
```

If no checksum errors are found the data is placed into memory in the address locations specified. If an error occurs a checksum error message is transmitted via the serial port. When the loader is finished, control is returned to the Monitor.

J-JUMP TO XXXX & RUN

JUMP and run executes in the same manner as the 'G' command except execution begins at the specified address. The interrupt vectors must still be in place if you are going to allow interrupts to occur.

L-INTEL LOADER LOW BITS

The 'L' command works exactly like the 'H' command except it deals with the lower 8 bits of the 16 bit words.

M-MODIFY WORD

This command is used to change the contents of a 16 bit memory location in program or data memory. The command can be used to force the D/A output of the TLC3204X to a particular value. This can be done by modifying location 0001. D/A changes will only occur if the AIO chip is active.

```
*m  [P]ROGRAM or [D]ATA
d
     XXXX
     2000 FE50
          fddd
```

P-PROGRAM
This command is used to launch one of the firmware programs located in the DSP-93 EPROM. To use the command, hit reset, then enter a 'P' and a '?' to see the list of programs available. Then enter your selection. The code for the application will be loaded into the DSP-93 and execution of the code will begin. If you do not want to run a program after entering the 'P', use the escape key or reset. The list of available programs may change from version to version.

R-RESET
Entering an 'R' will cause the DSP-93 to go through a soft reset. The results of this should be equivalent to a hardware reset. If you have entered the 'G' command and you are working in static RAM, the DSP-93 will bounce back to the EPROMs just as if you hit the reset button. Some of the areas in data RAM are initialized during a reset cycle. For example some code to keep the DSP-93 Monitor alive is placed from 1000h to 100Ch. This is mentioned in case you want to use the dump command to look at a program you just loaded after entering a reset.

S-SHOW WORD
This command is used to display the contents of a particular memory location. It can be used to check the current state of the A/D output for example by checking location 0000h. Once again, the AIO chip must be active for direct A/D access.

```
*s  [P]ROGRAM or [D]ATA
d
     XXXX
     2000 FDDD
```

T-MEMORY TEST
This command tests the current data memory for errors. The routine starts at 0FFFFh. If it gets an error there, it assumes you do not have the upper 32K of memory installed and the routine jumps to location 7FFFh and continues testing from there. The memory is tested by writing 0000h, 5555h, and 0FFFFh into every location and reading it back. The test will loop through all locations and give a '*' prompt if no errors occur. If an error occurs, the location of the error will be reported along with the value written and the value returned. Both RAM banks can be tested by issuing the 'T' command in slow mode (i.e., after a reset) and again in fast mode (i.e., after a 'G' command).

```
*t  MEMORY TEST

ERROR LOCATIONS:
FFFF  5555  D004
FFFE  5555  D004
FFFD  5555  D004
etc.......
```

Sine/Cosine table

In versions of the Monitor prior to Version 2.17, the sine/cosine table located in EPROM was copied to RAM memory when the 'G' command was issued. Beginning with Version 2.17, although the table is still in EPROM, it is no longer copied to RAM. This 1) frees up some Monitor code area 2) eliminates the possibility the sin table will write over code the user loads, and 3) allows user to locate the sin table in data memory which is the place he will really need it.

The table in EPROM is a 540 degree, 0.25 degree step, sine/cosine table is located in EPROM from 767Fh to 7EEFh and is scaled by 2^15. Sine begins at 767Fh, cosine begins at 77E7h.

To copy this table to your program's data space, we must play a little trick on the DSP-93. When a user program is running, the EPROM is switched out of the memory space and RAM is used for both program and data memory. To get access to the EPROM from a running program, the XF bit must be toggled, a code fragment to copy the data from EPROM to RAM copied to memory bank B0, then B0 switched to program memory and the data copy code executed. Note that the same process of loading B0 then branching to B0 after the loading and switch to high speed can be used to run code out of internal program memory, so as to achieve maximum speeds.

Code to copy the sine/cosine table to the user program's program memory follows. There is also a call to PM_BLK_MV to copy the block from program memory to the program's data memory.

```
ROM_START  .EQU 767fh       ;  Start address in Program (EPROM) to copy from
ROM_END    .EQU 7eefh       ;  End address in Program (EPROM) to copy from
RAM_START  .EQU 4000h       ;  Start address in RAM to copy to
LENGTH     .EQU ROM_END-ROM_START+1   ;  number of words to copy

B0DADDR    .EQU 200h        ;  Pointer to B0 in data
B0PADDR    .EQU 0FF00h      ;  Pointer to B0 in program

...

BEGIN
        LDPK    USER_PAGE
        CNFD                ;  B0 is data memory

; Move BLOCKMOVE code to page B0 (data)
        LARP    AR0         ;  AR0 indirect addr pointer
        LRLK    AR0,B0DADDR    ;  point AR0 to block B0
        RPTK    BLOCK_END-BLOCKMOVE-1     ;  repeat
        BLKP    BLOCKMOVE,*+;  move blockmove function
; BLOCKMOVE is now in B0 (data)
        CNFP                ;  B0 is program memory
; BLOCKMOVE is now in B0 (program)

        B       B0PADDR     ;  a branch to the moved BLOCKMOVE

; Data copied from EPROM ROM_START is now in program RAM at RAM_START

RESUME

;      ================================================
;      Move a block of data from PROG MEM to DATA MEM
;      ================================================

        LDPK    USER_PAGE
        LARP    AR1
        LRLK    AR1,RAM_START  ; Start of new location in DATA
        LALK    RAM_START      ; Start of block in Program memory
        SACL    063h           ; 063h of page 8, Start of block
        LALK    RAM_START+LENGTH-1        ; End of block in Program memory
        SACL    064h           ; 064h of page 8, End of Block
        LDPK    BASE_PAGE
        CALL    PM_BLK_MV

...

BLOCKMOVE
        SXF                 ;  XF bit ON to put EPROM in program memory
        LRLK    AR1,LENGTH-1     ;  set AR1 to length
        LRLK    AR0,RAM_START    ;  point AR0 to destination
        LARP    AR0         ;  AR0 indirect addr pointer
        LALK    ROM_START   ;  set ACC to source address
BLOOP   TBLR    *+,AR1      ;  move data
        ADDK    1           ;  increment ACC (source address)
        BANZ    B0PADDR+BLOOP-BLOCKMOVE,*-,AR0  ;  fixed-up address
        RXF                 ;  XF bit off to make data RAM into Prog RAM
        B       RESUME
BLOCK_END
```

The Parts of a DSP-93 Program

One of the best ways to learn DSP-93 programming is to read, study, and understand existing programs. A lot of source code is provided on the system diskettes. Use this valuable resource.

Not every program will use all these parts, and not all that do use them will use them exactly as shown. If they did, there would be only one program!

Header (Instructions, Copyrights, Disclaimers, etc.)

It's a good idea to have a header the tells what the program is, how to assemble and execute it, who wrote it and a Copyright and Disclaimer statement. See the sample program for more examples.

Example:

```
;  Experimental modem
;
;  To assemble, use the command
;     "asm expmodem.asm -dRn -dGn"
;  where Rn is R1 or R2 for radio ports 1 or 2
;     and
;  where Gn is one of G0, G1, G2, G3, G4, G5,
;    or G6 to set the gain
;
; Copyright © 1995 Ronald G. Parsons W5RKN
;
```

The Include Files

There are currently five include files that define most of the constants that you will use when programming the DSP-93. It is *strongly* encouraged that all programs include these files and use the constants therein. This will make your program easier to understand, and will reduce programming errors. These files are:

```
;  Module Name:
;       MACROS.INC
;  Purpose:
; *  This macro defines the origin in the
; *  code segment that the initialized data
; *  is to begin.
;
; *  This macro defines storage in the code
; *  segment while also creating a symbol
; *  representing the location the data will
; *  reside in data memory after an
; *  initialization block move.
;
```

```
;  Module Name:
;       MONITOR.INC
;  Purpose:
;    This file defines the DSP-93 Monitor
;    functions and addresses. Symbolic names
;    for all of the entry points have been
;    assigned.  This file should be used
;    instead of the absolute addresses since
;    a linker does not exist. The file may
;    also contain any macros defined in the
;    future that can help in using the
;    Monitor functions.
```

```
;  Module Name:
;       PORTS.INC
;  Purpose:
;    This file defines the DSP-93 I/O ports
;    and I/O bits.  The file assigns symbolic
;    names for each of the ports and bits.
;    The user should make use of the TASM
;    bitwise AND (&) and OR (|) operators to
;    manipulate the bits.  The programmer is
;    discouraged from using hex values in
;    programs, as other users may have to
;    read the listings, and the symbolic
;   names assign greater meaning to the code.
```

```
;  Module Name:
;       SERIAL.INC
;  Purpose:
;    Define symbols used in configuring the
;    serial port for the DSP-93. The symbols
;    in this file describe the 16550 UART.
;    Programmers should use this file instead
;   of using magic numbers in their programs.
```

```
;  Module Name:
;       REGS.INC
;  Purpose:
;    This file defines the register
;    replacements for the TASM assembler. The
;    register file AR0 through AR7 are
;    defined as well as the memory mapped
;    registers defined by the processor.
```

The following lines of code should be in your source after the header:

```
          .NOLIST
#include  "MACROS.INC"
#include  "REGS.INC"
#include  "PORTS.INC"
#include  "MONITOR.INC"
#include  "SERIAL.INC"
          .LIST
```

Memory Location Equates

Storage variable locations in internal and/or external memory must be defined in your program. These variables are defined using an .EQU directive to assign values to labels.

For example:

```
;  global variables
BUFI      .EQU   060h    ; AIO input buffer
BUFO      .EQU   061h    ; AIO output buffer
DO        .EQU   062h    ; data output buffer
```

Program Constant Equates

Constants used in your program should be assigned a label and that label be given a value using an .EQU directive. It is *strongly* encouraged that all programs use labels for constants rather that using constants in the body of the program. Note use of the CMDA_VAL macro below.

For example:

```
CMDA      .EQU      CMDA_VAL(6)     ;  AIO load RA/TA = 6 for SCF @ 833.333 kHz
CMDB      .EQU      CMDB_VAL(20)    ;  AIO load RB/TB = 20 for 41,666 samples/
sec

;  AIO cofig:  A/D gain 2, sync, no HPF or sin corr
CMDC      .EQU      AIO_CONFIG|AIO_GAIND71|AIO_GAIND60|AIO_SYNC

TOP       .EQU      10000     ;  ramp top value
```

Program Origin

The TASM assembler does not, by default, set the DSP-93 starting address of 1000h. So be sure to include the directive:

```
;  TASM does not assume the necessary 1000h starting address
          .ORG      1000h
```

Interrupt Vectors

The first four instructions of the program handle the various interrupt vectors and must branch to the appropriate labels. The Timer and Trap interrupts will not occur in the DSP-93 programs so they branch to the program starting label GO. Program execution begins at 1008h, just following the TRAP interrupt vector.

```
                            ;  Define Vectors
          B         GO        ;  branch to program start (TINT) Timer
          B         RINT      ;  AIO receive interrupt service routine
          B         XINT      ;  AIO transmit interrupt service routine
          B         GO        ;  branch to program start (TRAP)

GO        DINT                ;  program starts here
```

Initialization of the Math, Serial and Memory Model

There are various parameters for specifying how mathematical operations, the DSP chip serial IO and memory models will be handled. Your program should set these values at the beginning of the program.

For example:

```
;
; Initialization
;
GO      DINT                ;  disable interrupts

MATH_INI
        SSXM                ;  set sign extension mode
        SOVM                ;  saturate on overflow
        SPM     0           ;  set P shift = 0

        CNFD                ;  on-chip RAM configured as data memory
        SFSM                ;  frame sync SIO to AIO
        RTXM                ;  make FSX a CPU input
```

Initialization of the AIO

The initialization of the AIO chip is probably the most confusing aspect of DSP-93 programming. However, using the recipes below should enable you to start the chip sampling at the rate you desire without problem.

The AIO chip is reset and enabled by manipulating the IO data lines D15 and D14. In doing so, the variable CFG in which the value to be output to the RADIO_GAIN port must be in external memory, i.e. page 8 or greater. I would suggest copying the following code and defining constants in your program for appropriate values of GAIN and REC_IN, the software selectable gain and radio port.

```
          LDPK      USER_PAGE ;  data mem page 8 (external)

AIO_RST   LALK      AIO_RESET|GAIN|REC_IN
                            ; D14 low to reset AIO
                            ; D15 high to tri-state lines to/from AIO
                            ; Set gain and radio port
          SACL      CFG       ;  store in AIO CFG buffer
          OUT       CFG,RADIO_GAIN   ;  OUT to config port

          CALL      WAIT4     ;  delay 100 ms

AIO_INI   LALK      LED_202|GAIN|REC_IN ;  LED 202 on
                            ; D14 low to reset AIO
                            ; D15 low to enable DR & DX to CPU
                            ; Set gain and radio port
          SACL      CFG       ;  store in AIO CFG buffer
          OUT       CFG,RADIO_GAIN   ;  OUT to config port

          CALL      WAIT4     ;  delay 100 ms

AIO_ENA   LALK      AIO_ENABLE|GAIN|REC_IN
                            ; D14 high to end AIO reset pulse
                            ; D15 low to enable DR & DX to CPU
                            ; Set gain and radio port
          SACL      CFG       ;  store in AIO CFG buffer
          OUT       CFG,RADIO_GAIN   ;  OUT to config port

          CALL      WAIT4     ;  delay 100 ms
```

Immediately after resetting and enabling the AIO chip, it must be configured to specify the AIO gain, sync, filters and the values of RA/TA and RB/TB which set the sample conversion frequency. See "AIO Port Programming, Setting the AIO conversion frequency" below. I would suggest copying the following code and defining constants in your program for appropriate values of CMDA, CMDB and CMDC. See Program Constant Equates example above.

```
;
; AIO CONFIG
;
          LDPK      BASE_PAGE ;  data mem page 0 (internal)


          LALK      IMR_XINT  ;  mask SIO TX interrupt active
          SACL      IMR       ;  store INT mask

          ZAC                 ;  zero A
          SACL      BUFO      ;  store in BUFO

          EINT                ;  enable TX interrupt
          IDLE                ;  wait for TX interrupt

          LALK      AIO_CONFIG ;  load AIO command flag
          SACL      BUFO      ;  store in BUFO
          IDLE                ;  wait for TX interrupt

          LALK      CMDC      ;  load AIO config command
          SACL      BUFO      ;  store in BUFO
          IDLE                ;  wait for TX interrupt

          LALK      AIO_CONFIG ;  load AIO command flag
          SACL      BUFO      ;  store in BUFO
          IDLE                ;  wait for TX interrupt

          LALK      CMDB      ;  load TB/RB AIO command next
          SACL      BUFO      ;  store in BUFO
          IDLE                ;  wait for TX interrupt

          LALK      AIO_CONFIG ;  load AIO command flag
          SACL      BUFO      ;  store in BUFO
          IDLE                ;  wait for TX interrupt

          LALK      CMDA      ;  load TA/RA AIO command last!!
          SACL      BUFO      ;  store in BUFO
          IDLE                ;  wait for TX interrupt

          DINT                ;  disable interrupts
          LALK      IMR_RINT  ;  mask SIO RX interrupt active
          SACL      IMR       ;  store mask in IMR
```

Initialization of the DSP

The initialization of the DSP consists of initializing of any program values your program may use. For example:

```
DSP_INI   ZAC                ;  zero A

          SACL      DO       ;  zero buffer
          SACL      SO       ;  zero buffer
```

Handling Interrupts

The transmit and receive interrupts, generated by the AIO chip, must be handled by your code. This requires two functions RINT and XINT, pointed to by the interrupt vectors at the beginning of your program. Data received from the AIO are stored in the variable BUFI and processed, in this example, by the function DSP. Data to be sent to the AIO is stored in the variable BUFO and will be processed when a transmit interrupt occurs.

```
RINT      LAC       DRR       ;  get received AIO data sample from DRR
          SACL      BUFI      ;  store DDR in AIO input buffer
          CALL      DSP       ;  call DSP process
          EINT
          RET


XINT      LAC       BUFO      ;  get data from AIO output buffer
          SACL      DXR       ;  store data in DXR to transmit to AIO
          EINT
          RET
```

TINT and TRAP interrupts can be handeled in the same way.

Serial Port IO and Exiting to the Monitor

If your program is to read the DSP-93's Serial Port while executing, the following function will provide that capability. The function should be called from someplace in your code that is executed repeatedly. The character read is returned in the variable CHARREAD. If no character was available, ACC will be zero upon return.

In any case, to be a user-friendly DSP-93 program, include this function always. If an upper- or lower-case R is sent to the DSP-93 Serial Port, the program will exit to the Monitor.

```
;
; This is a modification of the ROM based INBIT code that is used to
; reset if a 'R' is received, or to return if no key is waiting.

CK_SERIAL
        LALK    UART_SEL_LSR    ; Select the line status register
        SACL    CHARREAD        ; Use temp memory
        OUT     CHARREAD,UART_CTRL   ; Select the register to read
        ZAC
        ZAC

        IN      CHARREAD,UART_READ ; Get the Line CTRL on SIO (get the selected reg)
        LACK    UART_LSR_RBF    ; Set bit for receiver holding register
        AND     CHARREAD        ; Test the 0th bit Line CTRL
        BZ      CK_RET          ; Receive Holding Register (not) Empty
```

```
        ZAC
        SACL    CHARREAD
        OUT     CHARREAD,UART_CTRL
        IN      CHARREAD,UART_READ
        LAC     CHARREAD
        ANDK    05Fh            ; match upper or lower case
        SUBK    'R'             ; Reset DSP-93
        BZ      RESET
CK_RET
        RET                     ; Continue
```

Wait Functions

There are two "wait" routines in the Monitor (See MONITOR.INC). There are also three "wait" functions that are commonly included in DSP-93 programs. These have delays of:

WAIT4 104 msec
WAIT2 52 msec
WAIT1 26 msec

```
WAIT4
          CALL      WAIT2
          CALL      WAIT2
          RET

WAIT2
          CALL      WAIT1
          CALL      WAIT1
          RET

WAIT1     LARK      AR2,0FFh
WAIT_A    LARK      AR3,0FFh
WAIT_B    LARP      AR3
          BANZ      WAIT_B    ; Loop through count
          LARP      AR2
          BANZ      WAIT_A    ; Loop through count
          RET
```

Defining Data Tables

Tables of data such as filter coefficients, strings, etc. may be defined within your program. For example:

```
          .MSFIRST
;
; Initialization table. This is used to configure the AIO at initial
; startup.
;
INITAB    .WORD       3          ; 0 load AIO configuration
          .WORD       CFGC       ; 1
          .WORD       3          ; 2 load TB register
          .WORD       CFGB       ; 3
          .WORD       3          ; 4 load TA register
          .WORD       CFGA       ; 5
          .WORD       0          ; 6 terminator
```

Using Pre-defined Data Tables

There are two tables of waveforms provided with the DSP-93 source code. For example:

```
; ————————— waveform tables ————————————-
;
; Converts phase position (0..255) to amplitude of that phase position
;
; Contains 4 tables, each 256 words long.
;
; Table 0 = Sine(theta) scaled by 2^15
; Table 1 = Triangle(theta)
; Table 2 = Square(theta)
; Table 3 = Sawtooth(theta)

          .MSFIRST

WTABLE    ; address of wavetable data in memory

#include  “wavetabl.dat”

; ————————— sin/cos tables ————————————-
;
; Sin/Cos table.  0..511 is Sin(theta), 512 words = 2 PI
;                 64..639 is Cos(theta), 512 words = 2 PI
;
; 512 words long for each table, total 640 words long
; The table is scaled by 2^11

          .MSFIRST

CTABLE    ; Sin/Cos() table, 512 words = 2 PI

#include  “sinco512.dat”
```

The End

Don’t forget the following directive at the end of your program.

```
          .END                              ; end of program
```

AIO Port Programming

Setting the AIO conversion frequency

The AIO conversion frequency (sampling frequency) is set during the AIO configuration. There are two values that determine the conversion frequency, TA and TB. The conversion frequency (in Hertz) is:

$$\frac{10{,}000{,}000}{2 \cdot TA \cdot TB}$$

The values are most easily set using the macros CMDA_VAL and CMDB_VAL.

```
CMDA      .EQU      CMDA_VAL(6)     ;  AIO load RA/TA = 6 for SCF @ 833.333 kHz
CMDB      .EQU      CMDB_VAL(20)    ;  AIO load RB/TB = 20 for 41,666 samples/sec
```

See “Linear Circuits; Data Conversion, DSP Analog Interface, and Video Interface; Data Book Volume 2” for more information on setting the conversion frequency. The AIO chip has a specified upper limit on the conversion frequency of 19.2 kHz, but the chip will operate considerably in excess of this. For example, the 9600 bps FSK modems use a conversion frequency 41666 Hz.

A table of conversion frequencies for various TA and TB is on the following page.

Hz	TB —>											
TA	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**
25	8000	8333	8696	9091	9524	10000	10526	11111	11765	12500	13333	14286
24	8333	8681	9058	9470	9921	10417	10965	11574	12255	13021	13889	14881
23	8696	9058	9452	9881	10352	10870	11442	12077	12788	13587	14493	15528
22	9091	9470	9881	10331	10823	11364	11962	12626	13369	14205	15152	16234
21	9524	9921	10352	10823	11338	11905	12531	13228	14006	14881	15873	17007
20	10000	10417	10870	11364	11905	12500	13158	13889	14706	15625	16667	17857
19	10526	10965	11442	11962	12531	13158	13850	14620	15480	16447	17544	18797
18	11111	11574	12077	12626	13228	13889	14620	15432	16340	17361	18519	19841
17	11765	12255	12788	13369	14006	14706	15480	16340	17301	18382	19608	21008
16	12500	13021	13587	14205	14881	15625	16447	17361	18382	19531	20833	22321
15	13333	13889	14493	15152	15873	16667	17544	18519	19608	20833	22222	23810
14	14286	14881	15528	16234	17007	17857	18797	19841	21008	22321	23810	25510
13	15385	16026	16722	17483	18315	19231	20243	21368	22624	24038	25641	27473
12	16667	17361	18116	18939	19841	20833	21930	23148	24510	26042	27778	29762
11	18182	18939	19763	20661	21645	22727	23923	25253	26738	28409	30303	32468
10	20000	20833	21739	22727	23810	25000	26316	27778	29412	31250	33333	35714
9	22222	23148	24155	25253	26455	27778	29240	30864	32680	34722	37037	39683
8	25000	26042	27174	28409	29762	31250	32895	34722	36765	39063	41667	44643
7	28571	29762	31056	32468	34014	35714	37594	39683	42017	44643	47619	51020
6	33333	34722	36232	37879	39683	41667	43860	46296	49020	52083	55556	59524
5	40000	41667	43478	45455	47619	50000	52632	55556	58824	62500	66667	71429
4	50000	52083	54348	56818	59524	62500	65789	69444	73529	78125	83333	89286

Hz	TB —>											
TA	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
25	15385	16667	18182	20000	22222	25000	28571	33333	40000	50000	66667	100000
24	16026	17361	18939	20833	23148	26042	29762	34722	41667	52083	69444	104167
23	16722	18116	19763	21739	24155	27174	31056	36232	43478	54348	72464	108696
22	17483	18939	20661	22727	25253	28409	32468	37879	45455	56818	75758	113636
21	18315	19841	21645	23810	26455	29762	34014	39683	47619	59524	79365	119048
20	19231	20833	22727	25000	27778	31250	35714	41667	50000	62500	83333	125000
19	20243	21930	23923	26316	29240	32895	37594	43860	52632	65789	87719	131579
18	21368	23148	25253	27778	30864	34722	39683	46296	55556	69444	92593	138889
17	22624	24510	26738	29412	32680	36765	42017	49020	58824	73529	98039	147059
16	24038	26042	28409	31250	34722	39063	44643	52083	62500	78125	104167	156250
15	25641	27778	30303	33333	37037	41667	47619	55556	66667	83333	111111	166667
14	27473	29762	32468	35714	39683	44643	51020	59524	71429	89286	119048	178571
13	29586	32051	34965	38462	42735	48077	54945	64103	76923	96154	128205	192308
12	32051	34722	37879	41667	46296	52083	59524	69444	83333	104167	138889	208333
11	34965	37879	41322	45455	50505	56818	64935	75758	90909	113636	151515	227273
10	38462	41667	45455	50000	55556	62500	71429	83333	100000	125000	166667	250000
9	42735	46296	50505	55556	61728	69444	79365	92593	111111	138889	185185	277778
8	48077	52083	56818	62500	69444	78125	89286	104167	125000	156250	208333	312500
7	54945	59524	64935	71429	79365	89286	102041	119048	142857	178571	238095	357143
6	64103	69444	75758	83333	92593	104167	119048	138889	166667	208333	277778	416667
5	76923	83333	90909	100000	111111	125000	142857	166667	200000	250000	333333	500000
4	96154	104167	113636	125000	138889	156250	178571	208333	250000	312500	416667	625000

IO Port Programming

Which Port does What

The TMS320C25 has 16 IO ports, many of which are implemented in the DSP-93. Data is written to an IO port with the OUT instruction and read with the IN instruction. For example:

```
        OUT     CHARREAD,UART_CTRL
        IN      CHARREAD,UART_READ
```

where in these examples, the data are sent from (read into) the variable CHARREAD and written to the IO port UART_CTRL or read from the port UART_READ. The ports are defined by the labels:

```
PORT_00h    .EQU    0       ;  Not defined for basic system
UART_WRITE  .EQU    01h     ;  UART Write
HS_CLOCK    .EQU    02h     ;  HIGH SPEED CLK & H_AIO
UART_CTRL   .EQU    03h     ;  UART CONTROL REGISTER
AIO_SELECT  .EQU    04h     ;  H_AIO SELECT OUTPUT
UART_READ   .EQU    05h     ;  UART READ  (READ ONLY)
TNC_OUTPUT  .EQU    06h     ;  TNC OUTPUT
RADIO_GAIN  .EQU    07h     ;  RADIO PORT SELECT AND GAIN & AIO ENABLE
PORT_08     .EQU    08h     ;  Not defined for basic system
PORT_09     .EQU    09h     ;  Not defined for basic system
TNC_INPUT   .EQU    0Ah     ;  TNC INPUT  (READ ONLY)
RADIO_CTRL  .EQU    0Bh     ;  RADIO PORT DIGITAL CONTROL
PORT_0C     .EQU    0Ch     ;  Not defined for basic system
PORT_0D     .EQU    0Dh     ;  Not defined for basic system
PORT_0E     .EQU    0Eh     ;  Not defined for basic system
PORT_0F     .EQU    0Fh     ;  Not defined for basic system
```

Hardware Port IO Assignments

[Bob Stricklin, N5BRG]

The following hardware port assignments are used by the DSP-93. These assignments are identified and labeled in an include file named PORTS.INC. When interfacing with any of the ports you should include this file with your code and use the labels provided. There is little chance you may damage the DSP-93 when writing code but if you error in assign port information and execute the code you may key your transmitter if it is connected. This may cause injury to you or damage to your equipment.

```
PORT CONNECTIONS AND PROGRAM ASSIGNMENTS

PORT 00h
  CODE        CONNECTION       NAME       DESCRIPTION
  D15-D0      NOT DEFINED FOR BASIC SYSTEM

PORT 01h     UART WRITE
  CODE        CONNECTION       NAME       DESCRIPTION
  D7-D0       U212-1-U212-8    SIO_TX     SIO TRANSMIT BYTE (8 BIT)
  D8-D15      Not defined

PORT 02h     HIGH SPEED CLK & H_AIO
  CODE        CONNECTION       NAME       DESCRIPTION
  D15-D0      NOT DEFINED FOR BASIC SYSTEM

PORT 03h     UART CONTROL REGISTER
  CODE        CONNECTION       NAME       DESCRIPTION
  D0-D7       Not defined
  D8-D9-D10   A0-A2 on U212 UART control registers
  D11-D15     Not defined

PORT 04h     H_AIO SELECT OUTPUT
  CODE        CONNECTION       NAME       DESCRIPTION
D3 D2 D1 D0
 X  X  X  1

PORT 05h     UART READ  (READ ONLY)
  CODE        CONNECTION       NAME       DESCRIPTION
  D7-D0       U212-1-8         SIO_RD     SIO READ BYTE (8 BIT)
  D8-D15      Not defined

PORT 06h     TNC OUTPUT
     CODE          CONNECTION   NAME       DESCRIPTION
 D11 D10 D9 D8
   X   X  X  1     TNC-1         *CDI       TNC *CDI   AND LED L215
   X   X  1  X     TNC-6         *KEYI      TNC *KEY I AND LED L216
   X   1  X  X     TNC-10        *KEYI      TNC *KEY I AND LED L217
   1   X  X  X     TNC-9          CTSI      TNC  CTS I

     CODE          CONNECTION   NAME       DESCRIPTION
 D15 D14 D13 D12
  X    X   X   1   TNC-11       XCLKI       TRANSMIT CLOCK
  X    X   1   X   TNC-13       RCLKI       RECEIVE CLOCK
  X    1   X   X   TNC-17       RDI         RECEIVE DATA
  1    X   X   X   TNC-20       TDI         TRANSMIT DATA
```

PORT 07h RADIO PORT SELECT AND GAIN & AIO ENABLE

CODE	CONNECTION	NAME	DESCRIPTION
D2 D1 D0			
0 0 0	I/O201 PIN 3	AUX IN	AUXILIARY AUDIO IN RADIO 1
0 0 1	I/O202 PIN 3	AUX IN	AUXILIARY AUDIO IN RADIO 2
0 1 0	Jx03 PIN 14	AUX #2	AUDIO BUSS #2
0 1 1	Jx03 PIN 12	AUX #1	AUDIO BUSS #1
1 0 0	I/O202 PIN 5	XMIT 2	TRANSMIT MONITOR RADIO 2
1 0 1	I/O201 PIN 8	REC IN	RECEIVE AUDIO RADIO 1
1 1 0	I/O201 PIN 5	XMIT 1	TRANSMIT MONITOR RADIO 1
1 1 1	I/O202 PIN 8	REC IN	RECEIVE AUDIO RADIO 2
D5 D4 D3			
0 0 0	FEEDBACK	GAIN=	Use MEASGAIN to determine your
0 0 1	FDBK R201	GAIN=	units gain.
0 1 0	FDBK R204	GAIN=	
0 1 1	FDBK R205	GAIN=	
1 0 0	FDBK R206	GAIN=	
1 0 1	FDBK R207	GAIN=	
1 1 0	FDBK R208	GAIN=	
1 1 1	FDBK R203	GAIN=MAX	
D7 D6			
X 1	U215-10	L212	PANEL LED 212
1 X	U215-11	L211	PANEL LED 211
D14	U210-9	*AIO_EN	ENABLE FOR AIO DR & DX SIO
D15	U208-2	*AIO_RST	Reset for AIO converter (*RESET)

PORT 08h

CODE	CONNECTION	NAME	DESCRIPTION
D15-D0	NOT DEFINED FOR BASIC SYSTEM		

PORT 09h

CODE	CONNECTION	NAME	DESCRIPTION
D15-D0	NOT DEFINED FOR BASIC SYSTEM		

PORT 0Ah TNC INPUT (READ ONLY)

CODE	CONNECTION	NAME	DESCRIPTION
D11 D10 D9 D8			
X X X 1	TNC-2	CDO	TNC *CD O
X X 1 X	TNC-5	RTSO	TNC *RTS O
X 1 X X	TNC-8		TNC PIN 8
1 X X X	TNC-7	CONNO	TNC CONN 0

D15 D14 D13 D12			
X X X 1	TNC-14	RCLKO	REC CLOCK
X X 1 X	TNC-16	XCLKO	TRANSMIT CLOCK
X 1 X X	TNC-18	RDO	RECEIVE DATA
1 X X X	TNC-19	TDO	TRANSMIT DATA

PORT 0Bh RADIO PORT DIGITAL CONTROL

CODE	CONNECTION	NAME	DESCRIPTION
D2 D1 D0			
X X 1	I/O201 PIN 2	*PTT	PUSH TO TALK RADIO 1
X 1 X	I/0201 PIN 7	*FREQ UP	FREQUENCY TUNE UP PULSE
1 X X	I/0201 PIN 1	*FREQ DN	FREQEUNCY TUNE DOWN & 470 TO PIN 6
1 X X	I/0201 PIN 4	*ICOM	470 OHM RESISTOR TO PIN 5
D5 D4 D3			
X X 1	I/O202 PIN 2	*PTT	PUSH TO TALK RADIO 2
X 1 X	I/0202 PIN 7	*FREQ UP	FREQUENCY TUNE UP PULSE
1 X X	I/0202 PIN 1	*FREQ DN	FREQEUNCY TUNE DOWN & 470 TO PIN 6
1 X X	I/0202 PIN 4	*ICOM	470 OHM RESISTOR TO PIN 5
D7 D6			
1 X	U215-13	L214	PANEL LED L214
X 1	U215-12	L213	PANEL LED L213

PORT 0Ch

CODE	CONNECTION	NAME	DESCRIPTION
D15-D0	NOT DEFINED FOR BASIC SYSTEM		

PORT 0Dh

CODE	CONNECTION	NAME	DESCRIPTION
D15-D0	NOT DEFINED FOR BASIC SYSTEM		

PORT 0Eh

CODE	CONNECTION	NAME	DESCRIPTION
D15-D0	NOT DEFINED FOR BASIC SYSTEM		

PORT 0Fh

CODE	CONNECTION	NAME	DESCRIPTION
D15-D0	NOT DEFINED FOR BASIC SYSTEM		

Debugging DSP-93 Programs

Debug functions in the Monitor

[Bob Stricklin, N5BRG]

The Monitor ROM includes a debug routine developed by Tom McDermott, N5EG. This routine can be called as a development aid when you are generating new DSP code or if you are just studying existing code. To use the routine include the MONITOR.INC file at the beginning of a program you are assembling with the following statement:

```
#include  "MONITOR.INC"
```

The MONITOR.INC file has three key statements which are used by a call to the debug routine. The statements are:

```
DEBUG       .EQU    0400h      ; Start of N5EG Debug Routine
```

This establishes the label DEBUG in your program. With this in place you can issue a CALL DEBUG in your program to use the DEBUG routines.

```
DEBUGPC     .EQU 007Eh      ; Pass Counter used by CALL DEBUG
```

This statement establishes a memory pointer. DEBUGPC should be referred to as the passcount/ mode value. DEBUG assumes this memory pointer will be located in page eight (8). You must store a value at this memory location before calling debug. The value tells debug how you want it to work. Here are the possible values to store:

```
          = 0      : causes the debug to trap always.
          = n      : value gets decremented by one each call, it stops
                     decrementing at zero and causes debug trap.
          = 0FFFFh : causes the debug to trap always, then waits for
                     keyboard input (any key) to continue.

DEBUGML     .EQU 007Fh      ; Starting data memory location used by CALL DEBUG
```

This statement also establishes a memory pointer. DEBUG assumes this memory pointer will be located in page eight (8) also. The pointer is for the first of eight memory locations which will be sent to the serial port. This will be handy if you want to watch some of your memory variables change as your program executes.

Debug uses the AR7 register as holding location for the current memory stack pointer. The Monitor places a value of 0AFFh in this register for the beginning of the stack location. The stack will be utilized by moving down towards 0h. The debug routine uses only a few of the memory stack locations, about 25, when it is called. Since AR7 is used to hold the location of stack you can not use it in your program if you intend to call debug.

You can see an example illustrating the assembly code used to call debug by reviewing the MON_TST.ASM routine.

Here is an example of the output from that routine:

```
PC=1018    A=00000060  AR=  2000  0EFC  FFFF  101A  FFFF  FFFF  FFFF  0AE8
                P=00000000  T=0000  ST0=EE08  ST1=17E0
0060:    0000  0000  0000  0000  0000  33E0  0E08  37E0
```

On the first line you have the program counter, accumulator, and eight AR registers. On the second line you see the P, T, ST0 and ST1 registers. On the third line you see the starting target memory location then the values sorted in eight memory locations starting with the target and including a total of eight locations. All of this will be lined up a little better in a later version of the Monitor.

Debugging using LOC.ASM

[Bob Stricklin, N5BRG]

The code in LOC.ASM was developed for finding problems. It may be used to determine when program execution stops. An author of code which seems to be stopping mysteriously should integrate this code into his program for testing. Then one of the users with a DSP-93 which hangs may run the modified program and report back with the results. Results can be reported by dumping memory in the DSP-93 using the Monitor. The memory dump can be captured and posted for evaluation by the author of the code. If we can determine when the problem is occurring the solution may also appear.

The program called LOC.ASM included in the LOC.ZIP file. This file may be found in the DSP-93 area on ftp.tapr.org. The file includes three subroutines for tracking programs while they execute. The subroutines are included in a program which just flashes LEDs as an example.

The routines included are TRACE, TRACE2, and TRACK.

TRACE writes the program memory location of each CALL which calls TRACE to the DSP-93 serial port. This allows continuous monitoring of the program while it is executing. To use it add the required code to your program and then add in some CALL TRACE statements. Run your program.

TRACE2 continuously updates a storage location with the program memory location of the most recent CALL which called TRACE2. Use this by adding the appropriate code and then run your program. If your program hangs or you stop it, you can use the Monitor SHOW command to look at memory location 0474h. This memory location will contain the program memory address of the last CALL TRACE2 executed. The location 0474h was used in the example program LOC.ASM but you may want to use a different location.

TRACK keeps a running count of the number of times your program passes a CALL which calls TRACK. When the CALL TRACK statement is encountered, the TRACK subroutine increments the value of a memory location which is 3000h higher in memory than the program memory location of the CALL TRACK statement. Review the assembly listing to determine the program location for the CALL TRACK statements you want to follow.

As supplied here when you run the LOC program it zeros memory in your DSP-93 from 4000h to 7FFFh. Then it calls TRACK, TRACE, and TRACE2. You will see 1018 start to scroll on your Monitor. This is the CALL TRACE statement which is being executed. If you review the listing below you will see this is the location of CALL TRACE2 is 1018.

After resetting out of the program you can hit 'G' in the Monitor and then 'S' then 'D' and enter 0474 to see a four digit hex value. This value is the last CALL TRACE2 executed before you hit the reset. For example it may be 103F but it could be any of the program address locations which have a CALL TRACE2.

Now hit the 'D' in the Monitor and dump 'D' data memory locations 4000 4050. You will see values in four locations;

```
4000  0000 0000 0000 0000 0000 0000 0000 0000
4008  0000 0000 0000 0000 0000 0000 0000 0000
4010  0000 0000 0000 0000 0000 0000 0003 0000
4018  0000 0000 0000 0000 0000 0000 0000 0000
4020  0000 0000 0000 0000 0000 0000 0000 0000
4028  0000 0000 0003 0000 0000 0000 0000 0000
4030  0000 0000 0000 0000 0000 0000 0000 0000
4038  0000 0000 0000 0000 0000 0003 0000 0000
4040  0000 0000 0000 0000 0000 0000 0000 0000
4048  0000 0000 0000 0000 0000 0000 0000 0000
4050  0002 0000 0000 0000 0000 0000 0000 0000
```

The locations are 4016, 402A, 403D, and 4050. These correspond to 1016, 102A, 103D, and 1050 in the test program LOC.ASM. So we can see all but one of the locations was executed three times and 1050 was only executed two time before the program was reset.

If you use this code you should double check the memory locations used to be sure they do not conflict with your program. Also make sure the page pointer tracks your code properly and the ARP register integrity is maintained. In most cases the values are preserved. The test code assumes your program is less than 3000 words in length. If this is not the case adjust the starting value up from 4000h.

Monitor IO Routines

A collection of Monitor routines which should help speed code development are documented here. Refer to the file MONITOR.INC included with the release disks for the exact location of these routines. This file should be included with your program in the following manner:

```
#include "MONITOR.INC"
```

An effort was made to preserve the state of the TMS320C25 when one of these routines is called. The register may change if they are involved in the routine but in most cases the processor will return with pointers and registers unchanged.

Here is a little more detail on each of the Monitor routines.

GET4HEX and GETVAL16

These two routines are the same. Collects a 16 bit hex value from the serial port. The value should be presented as four ASCII characters (0 through F digits). The four hex digits are converted to a sixteen bit HEX word and stored in a single memory location for later use. Look at MONITOR.INC for the storage location and a suggested label. This memory storage location is for the current page of memory established by the LDPK instruction.. The LDPK is normally set to page 8 with the instruction LDPK 8.

GET2HEX

This routine collects an 8 bit hex value from the serial port. The value should be presented as two ASCII characters (0 through F digits). The two hex digits are converted to an eight bit HEX word and stored in a single memory location for later use. Look at MONITOR.INC for the storage location and a suggested label. This memory storage location is for the current page of memory established by the LDPK instruction.. The LDPK is normally set to page 8 with the instruction LDPK 8.

GETCHAR

This routine collects an 4 bit hex value from the serial port. The value should be presented as one ASCII character (0 through F digits). The hex digit is converted to an four bit HEX word and stored in a single memory location for later use. Look at MONITOR.INC for the storage location and a suggested label. This memory storage location is for the current page of memory established by the LDPK instruction.. The LDPK is normally set to page 8 with the instruction LDPK 8.

HEXOUT

This routine assumes a single HEX digit is stored in the accumulator. This digit is converted to ASCII and sent to the serial port.

INBIT

Retrieves incoming data from the serial port. Data is placed in the indirect address location pointed at by AR(ARP).

OUTBIT

Sends the lower 8 bits of AR(ARP) to the serial port. This routine uses memory location 0060h of the current page for temporary storage. Location 0060h is used to store the UART register information while the routine is sending the data. The register information from the UART is used to determine if CTS is set or cleared and if the transmit data buffer is full.

OUTBIT2

Sends the lower 8 bits of location 0062h of the current page to the serial port. This routine uses memory location 0060h of the current page for temporary storage. Location 0060h is used to store the UART register information while the routine is sending the data. The register information from the UART is used to determine if CTS is set or cleared and if the transmit data buffer is full.

PRVAL08

The 8 bits in a defined memory location of the current page are converted to ASCII and sent to the serial port. MSB is sent first. See MONITOR.INC for the proper memory location and label to use for this routine. This routine uses HEXOUT.

PRVAL16

The 16 bits in a defined memory location of the current page are converted to ASCII and sent to the serial port. MSB is sent first. See MONITOR.INC for the proper memory location and label to use for this routine. This routine uses HEXOUT.

RESET

Jump to this location to restart the Monitor.

SD_CRLF

Sends a single carriage return and line feed to the serial port.

SD_STR

Calling this routine with a 16 bit address stored in the current AR(ARP) register will cause a string to be sent to the serial port. The C-style string should begin at the AR(ARP) memory pointer and end with 00. Only the lower 8 bits of the memory words will be sent. The string data must be in data memory space.

Here is an example of a string and the calling routine.

```
STR1    .string " D S P - 9 3   T A P R / A M S A T   R E V   2 . 1 5"
        .word   00
S_END
LRLK    AR0,2000h  ; String will be placed here in data memory
RPTK    S_END-STR1 ; Number of characters to move
BLKP    STR1,*+    ; Moves string from program to data memory
LRLK    AR0,2000h  ; point to memory location
CALL    SD_STR
```

SP1 through SP10

These routines will send the indicated number of spaces (20 Hex) to the serial port.

MWAIT1 and MWAIT_A

The WAIT routines are shown here. You can enter the routine at MWAIT1 or MWAIT_A with your own value set for AR2. With a 40 MHz clock the wait from MWAIT1 will be about 32 milliseconds while the wait from MWAIT_A will be about N mS where N is the value in AR2. Use MONITOR.INC for the entry locations for MWAIT1 or MWAIT_A.

```
MWAIT1
        SST1    MON6       ; Saves ST1 in 65h page 0
        SST     MON7       ; Saves ST0 in 66h page 0
        LRLK    AR2,020h
        CALL    MWAIT_A
        LDPK    0h         ; Set page 0
        LST1    MON6       ; Restores ST1 from 65h page 0
        LST     MON7       ; Restores ST0 from 65h page 0
        RET
MWAIT_A LRLK    AR3,02710h ; 02710h = 10,000
        LARP    AR3
WAIT_B  BANZ    WAIT_B,*-  ; Loop through count
        LARP    AR2
        BANZ    MWAIT_A,*-  ; Loop through count
        RET
```

MP_BLK_MV

This routine will move a block of data memory to program memory. You must provide pointers to the beginning and end of the block in program memory space and the beginning of the block in data memory space. This routine would be useful if you are dynamically creating program code in data memory and need to move it to program space.

```
;
;       ================================================
;       Move a block of data from DATA MEM to PROG MEM
;       ================================================
        LDPK    USER_PAGE      ; Must save data in page 8
        LARP    AR1
        LRLK    AR1,NEWMEM     ; point to DATA mem start location
        LALK    BLK_START      ; Start of block in PROG memory
        SACL    MON4           ; 063h of page 8
        LALK    BLK_END        ; Copy all the block in PROG mem down to the end
        SACL    MON5           ; 064h of page 8
        LDPK    BASE_PAGE      ; Must enter in page 0
        CALL    MP_BLK_MV
```

PM_BLK_MV

This routine will move a block of program memory to data memory. You must provide pointers to the beginning and end of the block in program memory space and the beginning of the new block in data memory space. This routine would be useful if you want to include a sin table in your program. Your program would be loaded into the DSP-93 in what will become program memory. After starting you program you would use this routine to move the table to data memory. The routine may also be used to move string data from program memory to data memory for later use. The following example shows how the routine would be called.

```
;
;       ================================================
;       Move a block of data from PROG MEM to DATA MEM
;       ================================================

        LDPK    USER_PAGE      ; Must save data in page 8
        LARP    AR1
        LRLK    AR1,NEWMEM     ; Start of new location in DATA
        LALK    BLK_START      ; Start of block in Program memory
        SACL    MON4           ; 063h of page 8, Start of block
        LALK    BLK_END        ; End of block in Program memory
        SACL    MON5           ; 064h of page 8, End of Block
        LDPK    BASE_PAGE      ; Must enter in page 0
        CALL    PM_BLK_MV
```

Monitor Memory Usage

The following data memory locations in the current page or absolute location are used by the Monitor.

All the following values are in HEX.
When * is indicated under absolute it means the current AR(ARP) and AR value are used.

APPLICATION	INDEXED MEMORY	ABSOLUTE MEMORY	REGISTERS USED
Reset, Initialization & Setup + Non callable Monitor code.	0,1,2,3,4,5,6,7,8,9 60,61,63,64,65,66,67	0200 - 020E 0400 0B00 - 0FFF 1000 - 100E 8018 - 801E	AR0,AR1

FUNCTION	PAGE 0	PAGE 8	CURR. PAGE	ABSOLUTE MEMORY	REGISTERS USED
DEBUG	65,66, 67,68, 69,6A	6,60,62, 7E,7F		x to 0AFF	AR7
MWAIT1	65,66			*+	AR2 & AR3
MWAIT_A				*+	AR2 & AR3
PM_BLK_MV	66,67	63,64,65		*+	current
MP_BLK_MV	66,67	63,64,65		*+	current

FUNCTION	PAGE 0	CURR. PAGE	ABSOLUTE MEMORY	REGISTERS USED
GET2HEX	65,66,67,69,6A, 6B,6D	1,2		none
GET4HEX	65,66,67,69,6A, 6B,6C,6D,6E,6F	1,2,3		none
GETCHAR	69,6A,6B,6D	1		none
HEXOUT	69,6A,6B,6D			none
HEXIN	69,6A,6D			none
INBIT	65,66,6D		*	current
OUTBIT	65,66,6D		*	current
OUTBIT2	69,6A,6B,6D			none
PRVAL08	65,66,69,6A,6B,6C,6D			none
PRVAL16	65,66,69,6A,6B,6C,6D			none
SD_STR	69,6A,6D		*+	current
SD_CRLF	69,6A,6B,6D			none
SP(n)	69,6A,6B,6D			none

Notes:

Items in [...] are anachronisms and not supported.

ACC is not preserved by the following routines

DEBUG Preserves all register values except AR7 which points to
the stack. Uses stack memory which starts at location 0AFFh and
moves down less than 0FFh.
GET2HEX gets and echos two ASCII, converts to hex into ACC and GET2HEXMEM (2h current page)
GET4HEX gets and echos four ASCII, converts to hex into ACC and GET4HEXMEM (3h current page)
GETCHAR gets and echos one ASCII, converts to hex into ACC and GETCAR_MEM (1h current page)
INBIT gets data into *
HEXIN gets data into ACC
HEXOUT sends data from ACC
OUTBIT2 sends data from ACC [OUTBIT2_MEM (62H page 8)]
OUTBIT sends data from *
PRVAL08 sends data from ACC [PRVAL_MEM (6H current page)]
PRVAL16 sends data from ACC [PRVAL_MEM (6H current page)]
SD_STR send zero-terminated string at * [(sends CRLF after string)]
SD_CRLF sends ASCII CR and LF
SP(n) sends n spaces
MWAIT1 delays 100 msec
MWAIT_A delays (AR2) times 1 msec
PM_BLK_MV block move
PM_BLK_MV block move

Assembling a DSP-93 Program

How to assemble a file using TASM under DOS

If you are using DOS, the command line to assemble a TASM source file, say XXX.ASM, is:

```
TASM -3225 -la1 -g0 XXX.ASM
```

This will result in a listing file XXX.LST being generated and an object file XXX.OBJ.

For further options when using TASM, see the TASM documentation.

How to assemble a file using D93WE under Windows

See the D93WE Help file for information on using D93WE as a code development environment.

How to assemble a file using DSP-93Control under MacOS

See the DSP-93Control ReadMe file for information on using DSP-93Control as a code development environment. The assembler available with DSP-93Control is very limited. Only simple programs without include files may be assembled.