adaptec, inc.

SCSI Development System Reference Manual

SDS-1

Revision 1.2

August 1986

Table of Contents

Section Page

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

LIST OF APPENDIXES

Appendix | Page

LIST OF FIGURES

LIST OF FIGURES

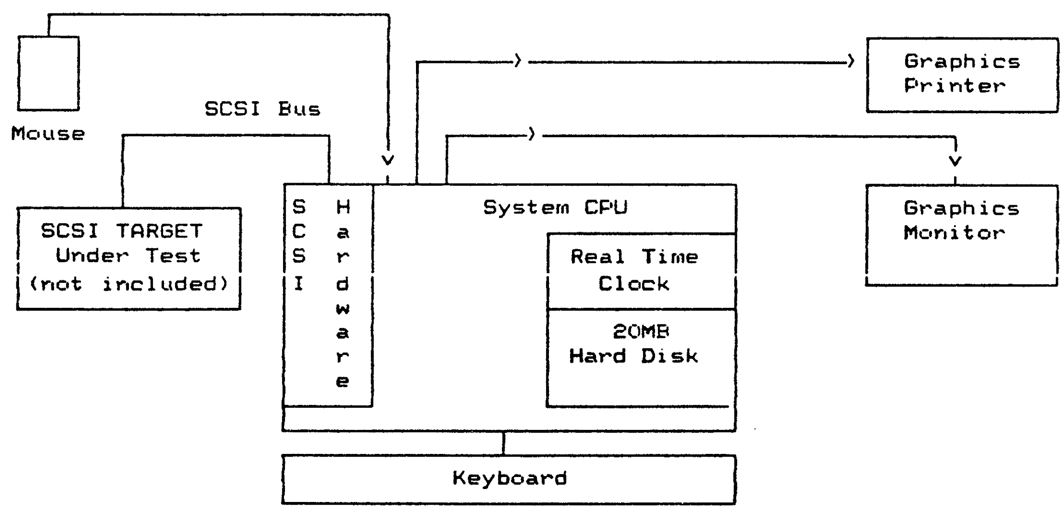
LIST OF TABLES

~INTRO.Ø SDS-1 INTRODUCTION

~INTRO.1 SDS-1 OVERVIEW

The Adaptec SDS-1 (SCSI Development System) is a stand-alone computer system designed to fulfill a number of test needs for SCSI peripheral development and qualification. Figure INTRO-F1 shows a system level block diagram of the SDS-1 which includes a hard disk-based computer with graphics monitor, graphics printer (optional), and mouse interface.

FIGURE ~INTRO-F1. SDS-1 SYSTEM LEVEL BLOCK DIAGRAM



INTRO.2 SDS-1 PRODUCT DESIGN GOALS

The SDS-1 was designed to perform in the following SCSI development environments for SCSI OEMs and System Integrators:

- Initial Engineering Debug:
  During initial product debug, the development Engineer needs a versatile but simple tool to use. The SDS-1 Development System's Menu Interface provides a quick, user-friendly testing capability at the SCSI electrical, message, command, status and sense levels.

- Final Product Debug:
  The time-consuming final steps in product debug require tools which provide flexibility and power to create unique tests which can uncover the "hard-to-find" bugs. The SDS-1 addresses this type of testing within the Menu Interface via a versatile menu-driven test compiler. This feature allows the user to quickly generate simple test sequences which can be executed with a single keystroke. The ultimate in flexibility can be obtained via use of the SDS-1's full Microsoft "C" compiler in the Stand-Alone Test (SAT).

- Engineering Performance Testing:
  Fully documented Engineering Performance tests can be quickly generated via the SDS-1's Menu Interface or SAT utilizing the built-in documentation functions.

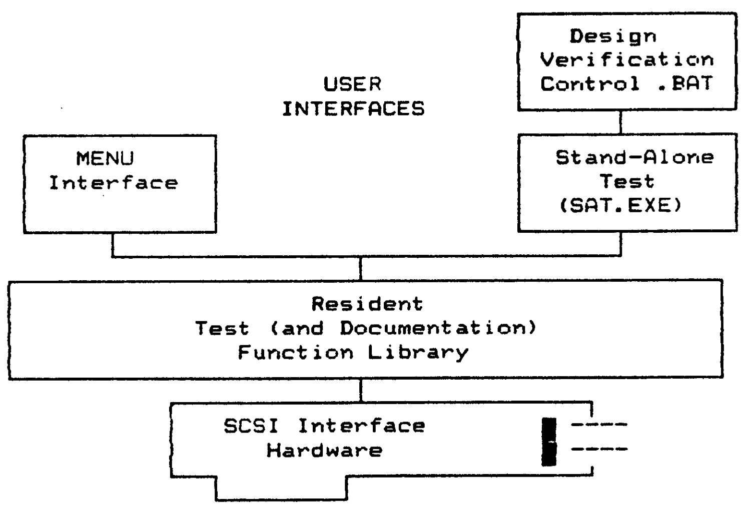
- Design Verification/Regression Testing:
  The SDS-1 provides a systematic approach to a "hands-off" initial design verification and regression testing during the course of the product's life. The Adaptec "Matched Set" (Test Procedure and Test Results Report) Documentation system provides the user with an easy-to-generate Test Procedure and Test Results Report which tracks the test procedure at a section, subsection, paragraph and subparagraph level.

- Product Assurance:
  With its ability to read and compare data up to 1.8 MB/second, the SCSI Development System allows Product Assurance a quick means of obtaining data reliability information. And with its Menu Interface, the SDS-1 provides a user-friendly interface with versatility.

~INTRO.3 SDS-1 DEVELOPMENT SYSTEM PRODUCT FEATURES

The power and flexibility of the SDS-1 is provided by a three-level architectural approach (Figure INTRO-F2): SCSI Interface, Resident Test/Documentation Function Library (RTFL) and User Interfaces. The two user interfaces, MENU and SAT (Stand-Alone Test), provide the user with different levels of flexibility and complexity.

FIGURE ~INTRO-F2. SDS-1 ARCHITECTURE



Some of the features of the SDS-1 are:

User Interfaces
- Menu Interface
  * Menu Driven Test Sequence Generation
  * User Customized Environment
- "C" Compiler for Creating SATs
- Program Debugger
- Batch File Regression Testing
- Adaptec "On-Line Reference Manual"
  * One Second Random Access to Reference Manual
  * All Manual Artwork On-Line
  * Context-Sensitive Reference Manual Access from any User Input Point

SCSI Environment Control
- Hardware or Software Arbitration (or no arbitration)
- Complete SCSI LEVEL 17 Command Set Macro (Test Functions) and Additional Common Command Set (CCS) Functions
- Ability to Create Vendor-Unique SCSI Commands
- Variable Speed/Types of Data Request/Acknowledge Handshake
  * Up to 1.8 MB/Sec Asynchronous to SCSI Development System Test Adapter On-Board 16K Buffer
  * SCSI Development System Memory DMA <==> SCSI Bus
  * Programmed I/O
  * Transmit/Receive State Machine Handshake (auto ACK gen)
- "On-the-Fly" Data Comparison (real time read after write data integrity checking)
- SCSI Parity Generation/Checking Enable/Disable
- SCSI Parity Error Generation Capability
- Microprogramming (allows complex SCSI message system testing/verification)

Architecture
- SDS-1 Backplane Buffers and High-Speed Test Adapter On-Board Buffer
- Automatic Hardware/Software Data Compare Capability

Documentation
- Adaptec Exclusive Test Procedure Generator (can generate test procedures utilizing the design verification batch file as a Table of Contents and the embedded test procedures found in each Stand-Alone Test)
- Test Sequence Run-Time Operators (provide a 1:1 tracking between the execution "Test Results Report" and the "Test Procedure Report" generated by the Report Generator, known as the Adaptec Matched Documentation Set)

INTRO.4 SYSTEM COMPONENTS

The SDS-1 is comprised of the following hardware, software and manual components:

HARDWARE CONTENTS

- SDS-1
  - 64ØK User Ram
  - One 36ØK Floppy Drive
  - One 2ØMB Winchester Drive
  - Real-Time Clock
  - One Serial Port
  - One Parallel Port
  - One SCSI Single or Differential Test Port
  - 8Ø-Column x 25-Line Monochrome Display
- 8Ø-Column Graphics Printer (Optional)
  - Desktop Printer Stand
  - Printer Cable
- Mouse and Mouse Pad

SOFTWARE/MANUALS CONTENTS

- SCSI Development System Software
  - "On-Line Reference Manual"
  - Resident Test/Documentation Function Library (RTFL)
  - Run-Time Batch File Documentation Functions
  - Menu Interface
  - "C" Stand-Alone Test Generation Routines
  - Test Procedure Report Generator
  - SAT/Regression Test Examples
  - Interactive Editor
- SCSI Development System Reference Manual (Hard Copy)
- SAT Library Catalog Binder
- Microsoft "C" Compiler Diskettes and Reference Manual Set
- PC DOS Diskettes and Reference Manual Set
- Mouse Systems PC PAINT PLUS Diskette and Reference Manual Set
- Computer Reference Manuals
- Real Time Clock Utility Diskette and Reference Manual
- Borland Sidekick Diskette and Manual

~INTRO.5 USING THE SDS-1

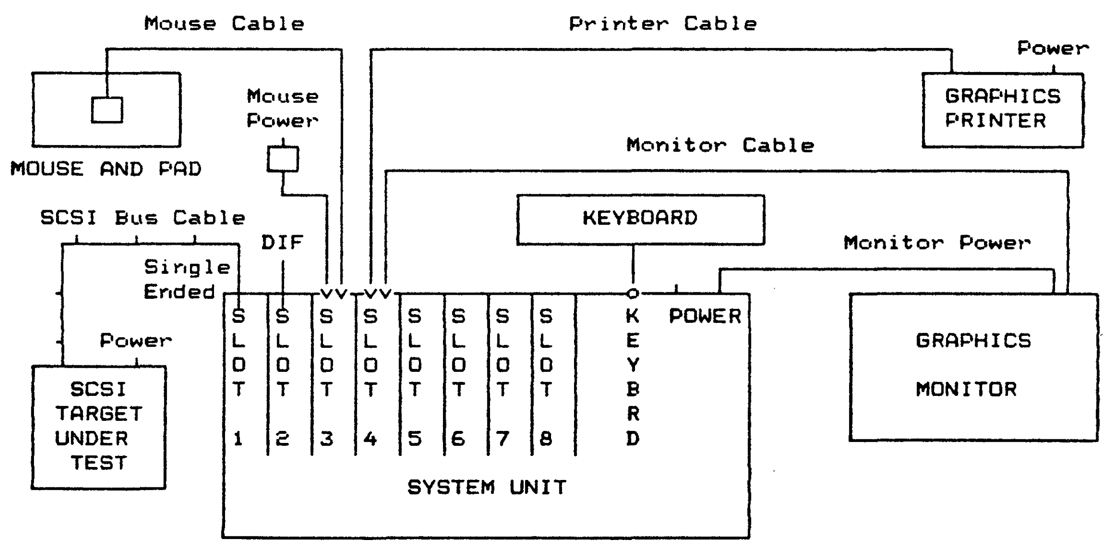
At this point, the user may be reading a magnetic version of the SDS-1 Reference Manual, which is displayed at system boot time, or the hard copy version. The following steps will get the user involved with the SDS-1 and serve as a quick system checkout.

1. SYSTEM SETUP:
   The system components (base unit, monitor and keyboard) should be connected as shown in Figure INTRO-F3. The printer should be connected and on-line. For customers purchasing the SDS-1 without a printer, connect one of the following qualified printers:

   OKIDATA Microline 192
   IBM Graphics Printer

   The mouse should also be connected (refer to Section RPTG.2.1.2.5.1. for mouse hardware setup).

FIGURE ~INTRO-F3. SYSTEM HOOKUP



2. SCSI PERIPHERAL HOOKUP:
   Next, connect an SCSI peripheral disk or tape drive. If using a disk, try to choose a preformatted one. Pin One of the SCSI cable points up. It would be easier to run the example if the initial peripheral requires SCSI bus parity.

3. MENU INTERFACE:
   It is now time to leave the "On-Line Reference Manual" and proceed to the Menu Interface. But before leaving the Help System, scroll the display such that the top line displayed

is **3a. WRITING AND READING:.** Now mark this line with Book Mark 1 by pressing **ALT-1** keys (while pressing the **ALT** key, press the **1** or number one key).

NOTE: This allows the user to reenter the Reference Manual (Help System) at this paragraph from the DOS command line by: C>**SDSHELP BM1** or from the reference manual TOC via the BOOK MARK SECTION and BM1.

If a hard copy of the Reference Manual is not available, the user may want to print out Step 3a: adjust the screen to Step 3a and press **SHIFT-PrtSC.** The user may want to do another print screen since this Step does not fit on a single screen.

To leave the Reference Manual (Help System), enter the **ALT-H** keys (while pressing the **ALT** key, press the **H** key).

3a.WRITING AND READING:
After leaving the Reference Manual, invoke MENU by entering:

C>**MENU SAMPLES**

at the DOS prompt. SAMPLES will initialize the system and place the user in the RANDOM menu. If the initial peripheral does not require parity, the user may reset **parity(1)** in the SETUP Menu to **parity(0)** (refer to MENU.1.1). To get acquainted with the SDS-1 MENU, perform the following operations:

KEYBOARD INPUT FOR DISK	KEYBOARD INPUT FOR TAPE	DESCRIPTION
R	S	If Already in Proper Menu, Skip
T	T	Performs SCSI Bus Reset
N	N	Performs Sense Command
W	W	Write 10 Blocks
	X	Rewind Tape
E	A	Reads 10 Blocks
B	B	Move to BUFFER Menu
Z	Z	Displays Read Buffer
F	F	Displays SCSI State Log

Return to the Reference Manual by pressing the **ALT-H** keys. The user will return to the Reference Manual at **MENU.3 BUFFER MENU.** Return to Step 3a by pressing the **1** (number one) key for Book Mark 1.

NOTE: If the user has followed Step 3a, the Reference Manual (Help System) was entered through MENU. To return back to MENU, enter **ALT-H.**

4. SDS-1 ARCHITECTURE BASICS:
At this point, a SCSI write/read operation has been executed and the SDS-1's Bus State Log has been displayed. Before

proceeding, a few architectural concepts should be understood.

INTRO.5.1 SDS-1 ARCHITECTURAL CONCEPTS

INTRO.5.1.1 BUFFERS

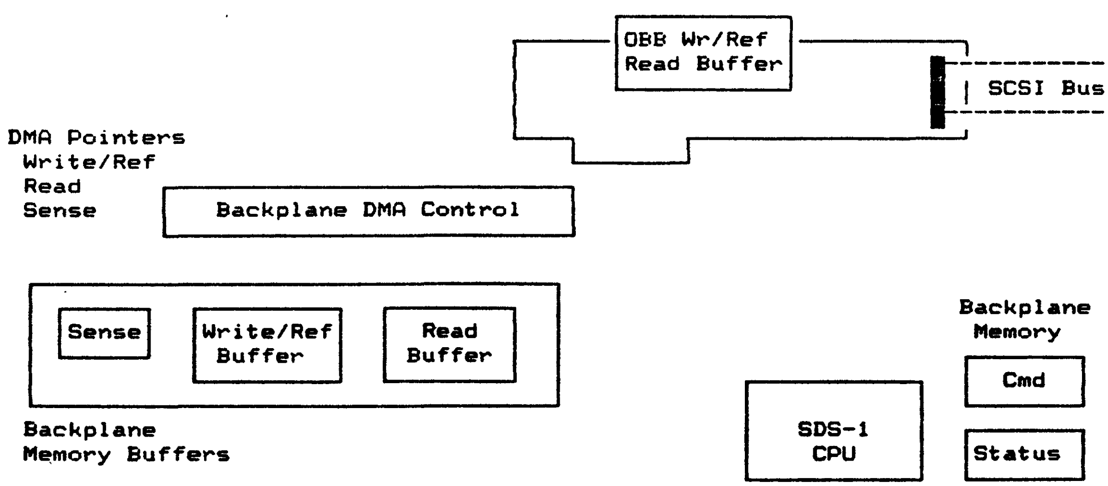
Figure INTRO-F4 shows the basic buffer structure of the SDS-1. Initially, we will focus on the backplane memory buffers: Write/Reference, Read and Sense. All SCSI DATA OUT transfers are taken from the SDS-1 write buffer. The starting location of the transfer is set by the Write DMA pointer. The SDS-1 provides a number of different buffer fill functions which allow the user to create any data pattern in the write buffer. Unless changed by the user, the write/reference buffer is the target buffer for all fill functions.

With the exception of the **sense()** command, all SCSI DATA IN transfers write data into the SDS-1 read buffer. The starting location of the transfer is determined by the Read DMA pointer.

The sense buffer is dedicated to SCSI sense DATA IN. Each **sense()** command writes data into this buffer starting at DMA address Ø.

The SDS-1 manages buffer wraparound for the backplane buffers. This means that if a transfer exceeds the size of the buffer, the SDS-1 will automatically stop the transfer at the buffer limit, reset the correct DMA pointer to the start of the buffer, and continue the transfer.

FIGURE ˜INTRO-F4. SDS-1 BUFFER ARCHITECTURE

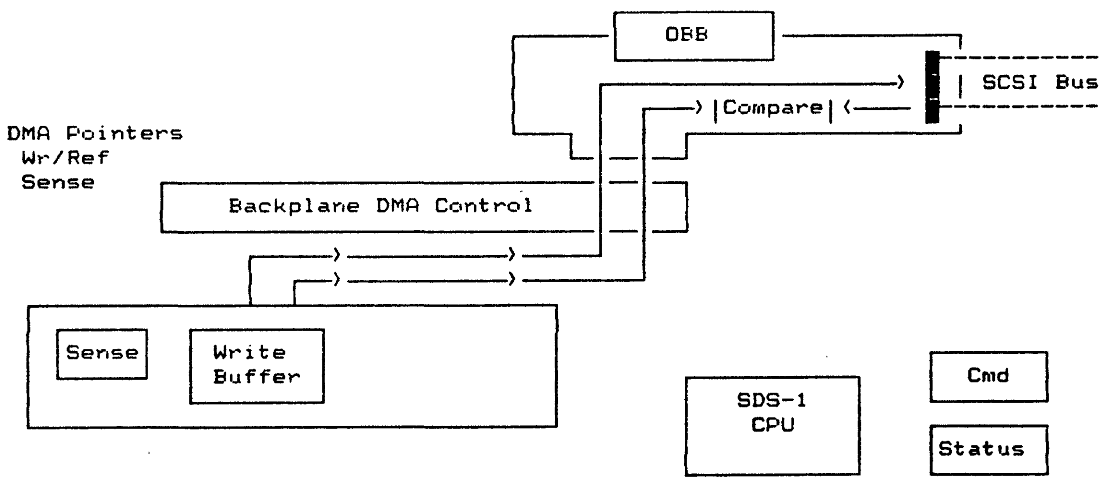


The SDS-1 provides numerous transfer modes: Programmed I/O (PIO), Transmit/Receive (TR), Direct Backplane Memory Access (DMA) and High-Speed Direct Memory Access to the SDS-1 On-Board Buffer (HS). In addition, various methods of data comparison can be specified. The following section describes the most commonly used method, Hardware Compare.

INTRO.5.1.2 HARDWARE COMPARE

When operating in a hardware compare mode, the SDS-1 transfers SCSI DATA OUT information from the WRITE/reference buffer (see Figure INTRO-F5) using the DMA pointer. During SCSI DATA IN phases (with the exception of a **sense()** command), the SCSI bus data is held on the SCSI bus and compared against the write/ref buffer (via a hardware comparator) using the Write DMA pointer as an index into the write/REFERENCE buffer. Since data is read from the write/REFERENCE buffer via DMA, this is a very fast operation.

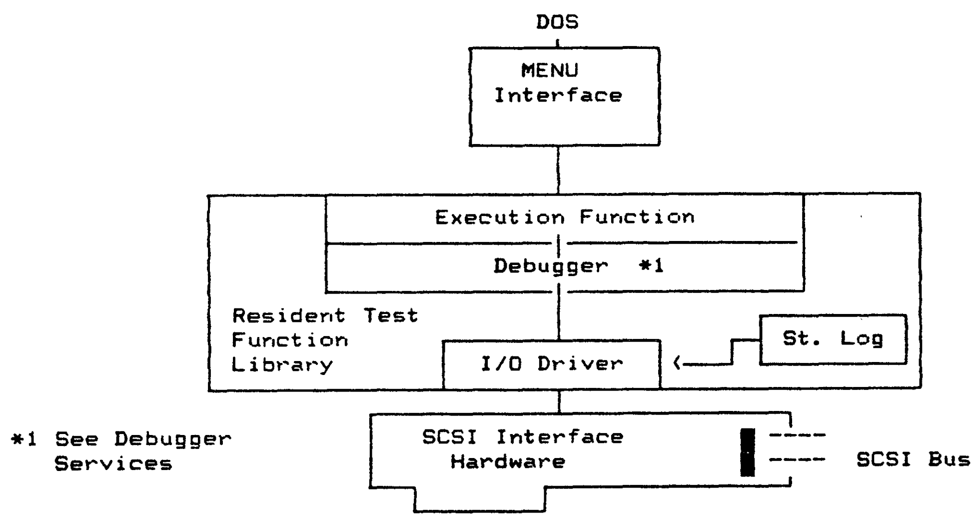
FIGURE ~INTRO-F5. SDS-1 HARDWARE COMPARE



INTRO.5.1.3 I/O DRIVER/MENU EXECUTION ENVIRONMENT

Another concept to understand is the Menu Interface/I/O Driver execution environment and its relationship to the Resident Test Function Library (RTFL). Figure INTRO-F6 shows the basic architecture of the SDS-1. The Menu Interface is a special application designed to give the user easy access to the SDS-1 Resident Test Function Library. This library lives in system memory, just like DOS and is accessible via a fixed entry point. The Menu Interface simply allows the user to make function calls to the library in the order chosen. Certain functions within the function library (such as **writer()** or **readr()**) interact with the SCSI bus. This interaction is accomplished via an I/O Driver. As shown, the I/O Driver can report its status to the SCSI Bus State Log.

FIGURE ~INTRO-F6. MENU/I/O DRIVER EXECUTION ENVIRONMENT



The Resident Test Function Library Debugger is heavily utilized by the SDS-1's Menu Interface. The Debugger provides the following services to the MENU:

Service	See Figure INTRO-F7
Execution Statistics Display	Left-Hand Frame
Buffer/DMA Pointer Display	Lower Center Frame
SCSI Command Block Display	Right-Hand Frame
I/O Driver Control Flags	"
SCSI Status Byte Display	"
SCSI Sense Data Display	"
RTFL Function Execution History (Trace Display)	Lower Frame

In addition to the display services, the RTFL Debugger also provides a Debugger to aid in the debug of MENU Function Key (FKEY) sequences or Stand-Alone Tests.

FIGURE ~INTRO-F7. RTFL DEBUGGER DISPLAY

```
+------------------------------------- I/O DRIVER STATUS ----------------------------------------+
| I/O  Ops:       2F | uc0:               |  I/O Command Parameters       stat: 00 __              |
| TGT Chks:        0 | uc1:               | CDB: 08 00 00 c0 40 00        sense: (old)             |
| INT D Er:        0 |--------------------|      00 00 00 00 00 00        00 00 00 00              |
| Bytes Wr:    F0400 | Wr/Ref: BPM        | xfer: DMAHC      a.s.:OFF     __ __ __ __              |
| Bytes Rd:    50000 |              0000  | s.l.ON  arb.HDW sel.SMA       __ __ __ __              |
| Bytes Cp:    20000 | Rd Buf:            | b.p.OFF b.w.OFF               __ __ __ __              |
| Cmp Ers.:        0 |                    | ha: 0   iid: 7   tid: 4       __ __ __ __              |
+------------------------------------- TRACE DISPLAY --------------------------------------------+
writer(0580,40) overbcw(05c0,0100,0000,4000) writer(05c0,40)
overbcw(0600,0100,0000,4000) writer(0600,40) overbcw(0640,0100,0000,4000)
writer(0640,40) overbcw(0680,0100,0000,4000) writer(0680,40)
overbcw(06c0,0100,0000,4000) writer(06c0,40) paragph() ackdelay(2100)
fillpr(009f,0000,0200) savebuf(OBBIMG.TST,0000,0200) writer(0a00,2)
paragph() dmarst(R) ackdelay(0) readr(0000,0040) paragph() dmarst(R)
ackdelay(15) readr(0040,0040) paragph() dmarst(R) ackdelay(255)
readr(0080,0040) paragph() dmarst(R) readr(00C0,0040) paragph() dmarst(R)
readr(0300,001F) readr(031F,0020) readr(033F,0001) paragph() ackdelay(0)
dmarst(R) readr(0900,0001) readr(0901,0010) readr(0911,000F)
readr(0920,0020) group() xfermode(DMAHC,4000) paragph() fillk(00,0000,4000)
```

This concludes the basic SDS-1 architectural concepts. The following outline is provided in order to guide the user through the use of the SDS-1 Menu System into Stand-Alone Test Generation and onto SCSI Design Verification Testing.

TOPIC	REFERENCE MANUAL SECTION/SUBSECTION
MENU System	Menu Interface
Individual Menus	Menu Interface/"menu name"
Parameter Save/Load	Menu Interface/Other/Exit
FKEYs	Menu Interface/FKEY Menu
Saving/Loading	"
Debugging	"
Host Emulation (I/O Driver operation)	I/O Driver
SAT Generation	Stand-Alone Test
I/O Driver Environment	I/O Driver
Debugger	Debugger
State Log	State Log
Microprogramming	Microprogramming
Function Library	Function Library Appendix A
Design Verification	Design Verification
Report Generator	Design Verification Report Generator

INTRO.6 HELP SYSTEM

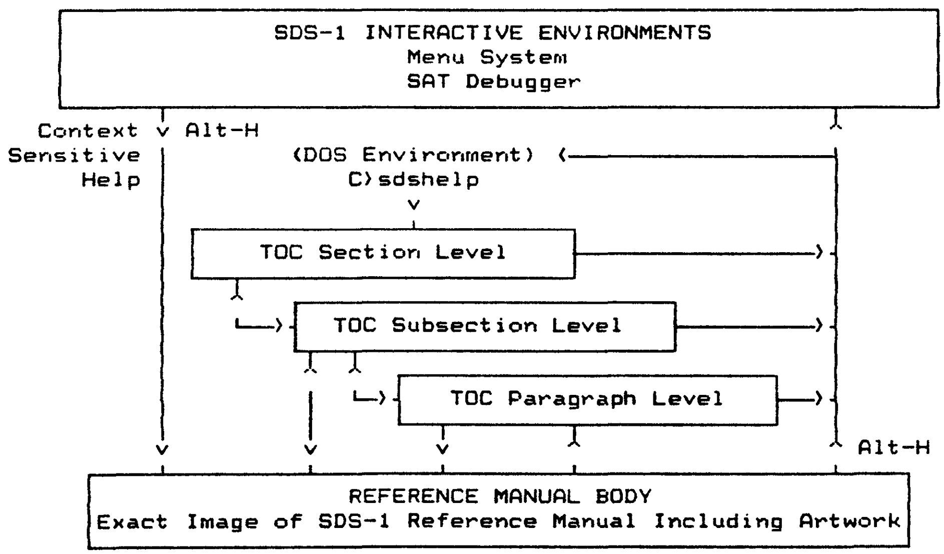
The SDS-1 Help System provides two features to the user. First, the Help System contains the full SDS-1 Reference Manual (including artwork) with an electronic Table of Contents (TOC) which can access any page in less than a second. The other feature is that the Help System is integrated with the SDS-1 Menu Interface and SDS-1 Debugger to provide the user with "context-sensitive help" at any user input point. That is, whenever the SDS-1 requires a user input, the user can press **ALT-H** keys to get more information from the Reference Manual.

Figure INTRO-F8 shows a diagram of the Help System starting with the two methods of entry, context-sensitive (**ALT-H**) or direct entry (DOS environment). To utilize the direct entry feature of the SDS-1 Help System, the user enters "SDSHELP xxxxxxxx" at the DOS prompt, where "xxxxxxxx" is the reference manual entry point. Below are some examples:

C>**SDSHELP writer**
C>**SDSHELP INTRO.5**
C>**SDSHELP**

The first command will take the user to the **writer()** function description page in Appendix A. The second command will take the user to Section INTRO.5 and the last command will take the user to the Electronic Table of Contents (TOC). The electronic TOC is the default if the reference manual entry point is not found.

FIGURE ~INTRO-F8. SDS-1 HELP SYSTEM



Once in the Reference Manual Text, the user can move from one end of the manual to the other. The three-level TOC provides top-down access to all SDS-1 subjects.

The user can also set electronic bookmarkers (ALT-1,ALT-2,..., ALT-9,ALT-Ø) and can return to a bookmarker via a single keystroke (1,2,...,9,Ø). Bookmarkers can be viewed from the Section Level of the TOC. They are preserved until the next system boot or reset.

Below is a list of commands used in the magnetic version of the Reference Manual:

IN THE TOC BODY	KEYS TO USE:
Select Section, Subsection or Paragraph:	Up or Down Arrow
Expand Section, Subsection or Paragraph:	Carriage Return
Contract Subsection or Paragraph:	ESC
Swap to Original Screen (before Help):	Space Bar
Exit Help Screen:	ALT-H

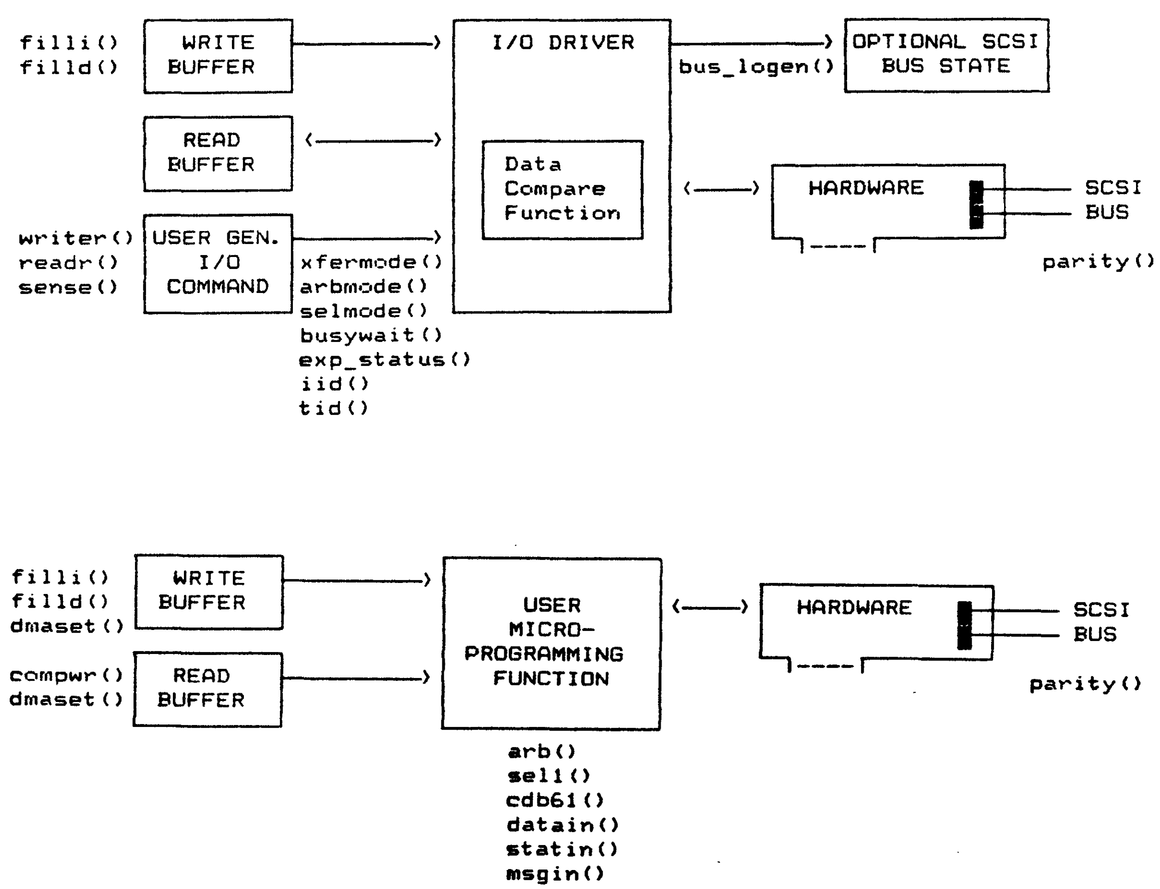
IN THE REFERENCE MANUAL TEXT BODY	
Scan through Text:	Up or Down Arrow or Page Up or Down
Return to TOC:	ESC
Set Bookmarker:	ALT-Ø through ALT-9
Go to Bookmarker:	Ø through 9
Swap to Original Screen (before Help):	Space Bar
Exit Help System:	ALT-H

~MENU.Ø MENU INTERFACE

~MENU.1 INTRODUCTION/OVERVIEW

The Menu Interface allows execution of the Test/Documentation Functions via an interactive menu-driven system. It supports both the I/O Driver and Microprogramming execution environments as shown in Figure MENU-F1.

FIGURE ~MENU-F1. MENU INTERFACE EXECUTION ENVIRONMENTS



There are currently eight menu displays or screens (SETUP, BUFFER, RANDOM, SEQUENTIAL, I/O DRIVER, MP, FKEY and OTHER/EXIT), each of which contains a set of functions that can be executed with a single keystroke. In addition to executing these functions, there is the ability to edit individual function parameter(s). The Menu Interface also has the flexibility to custom-build function sequences that can be executed by a single keystroke (F1 through F10). The function key (FKEY) sequences are discussed in more detail in section MENU.8 .

The Menu Interface is invoked by the command:

C>**MENU** <filename>

where the file name is optional. The file name is the name of a file that has the stored parameter value set which had been saved from a previous Menu Interface session. If the file name was not specified, default parameter values will appear on the menu screens.

Once the menu initialization process is done, one of the menu screens will be displayed. This is the default screen which can be modified by the init_menu function in the OTHER/EXIT menu. To display the other menu screens, use the **Left** or **Right Arrow** keys or the menu code which is highlighted on the Menu Page Select Line displayed at the bottom of the screen. The current menu screen is noted in inverse video.

The Menu Interface screen contains three major areas: Debugger Window, Menu Screen and Trace Display. The top portion of the screen is the Debugger Window which provides the user with information such as statistics, counters, buffers, SCSI command bytes, sense display and other status. The Debugger Window is discussed in more detail in the DEBUG.2.1.3 section. The lower portion of the screen is the Menu Screen which displays the current menu with its functions available for execution. The Trace Display is swapped with the Menu Screen; it shows the execution history of the Test/Documentation Functions that have been executed (refer to DEBUG.2.1.4).

MENU.1.1 PARAMETER SETUP IN THE EDIT MODE

The edit mode is used to set up or modify parameter values. To enter this mode, hold down the **CTRL** key while pressing **E** (will be written as **CTRL-E** or **^E**) at the menu screen. To exit this mode, enter **^E** again.

While in the edit mode, the cursor will appear in the current parameter field which is displayed in inverse video. A help reference line with a brief description of the current field will also appear at the bottom of the screen. To move the current parameter field to the previous or next field, use the **Up** and **Down Arrow** or **Return** keys. The **Home** key will move the current parameter field to the first parameter field at the top of the menu screen and the **End** key will move it to the last function.

The **PgUp** and **PgDn** keys will move up or down a line to the first parameter field in the line. A summary of the edit mode keys are displayed at the bottom of the screen.

To edit the parameter value, type in the new value in the parameter field. The values maybe in decimal, hexadecimal or alphanumeric. Some parameters are strings which are noted by double quotes. If the value is to be hexadecimal, an "x" or "X" must appear in the field before the value. For example, decimal 256 is 0x100 in hex, the "X" must be present so that the Menu Interface will interpret this value as a hex value. There is some range and type checking, so that an error will appear if the value is not within its limits or if it is an illegal value. This error will continue to show until a legal value is entered. The displayed value in the parameter field is the value to be interpreted by MENU, so be sure the correct value is shown.

Some of the parameters are toggles. To modify these values use the **Left** or **Right Arrow** keys. Refer to the bottom of the screen for other instructions on editing parameters.

MENU.1.2 FUNCTION EXECUTION

When parameters have been setup, the user may execute these functions. If the user is in the edit mode, be sure to exit that mode. On the menu screen, each function has a highlighted or intensified character preceding the function name. This is the execution key code associated with the function. When this key is entered, the function will execute using the displayed parameter value(s). Only functions in the current menu can be executed.

Some execution key codes are shown in inverse video. These functions are toggles. Their purpose is to set flags or variables. They are not part of the Test/Documentation Function Library.

MENU.1.3 TRACE DISPLAY

When executing a Test/Documentation Library function, the menu display is replaced by the trace display which shows the function name and parameter(s) that have been executed. Internal Menu functions (functions that are not part of the Test/Documentation Library) do not appear in the trace display.

When in the menu display, the trace display can be viewed by pressing the **Space Bar**. Pressing the **Space Bar** again will return the user back to the menu display.

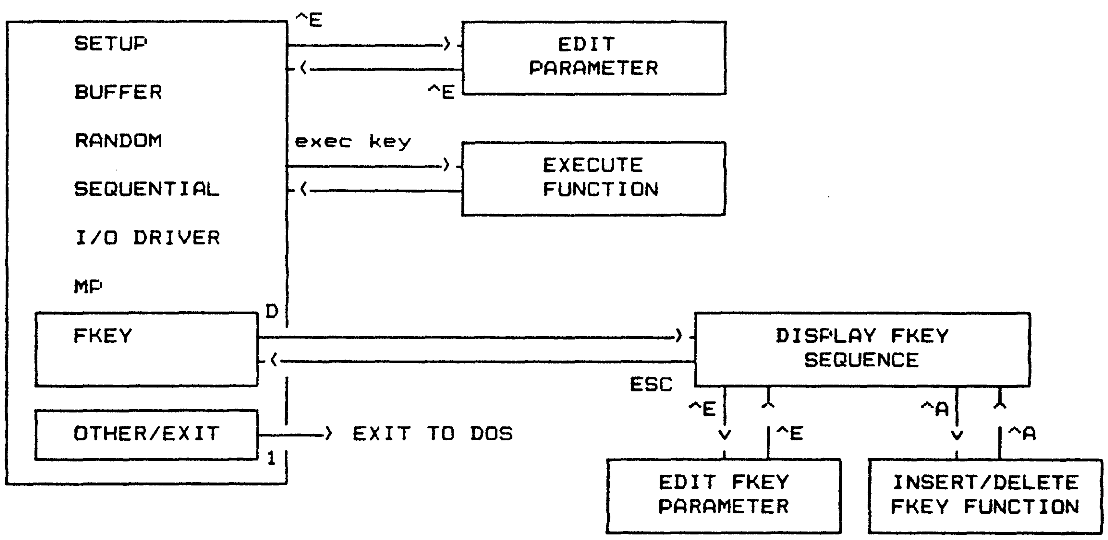
MENU.1.4 SETTING ERROR ACTION

In the Menu Interface, the two error actions available are: LOGC (log and continue) and LOGH (log and halt). These are set by the **iea()** and **eea()** functions in the SETUP menu.

MENU.1.5 MENU INTERFACE STATES

Figure MENU-F2 is a diagram of the Menu Interface states which display the various states and modes that can be accessed through the different menus.

FIGURE ~MENU-F2. MENU INTERFACE STATES



~MENU.2 SETUP MENU

The SETUP menu contains functions that control the execution environment of SCSI execution functions. A typical SETUP menu screen is shown on the next page.

FIGURE ~MENU-F3. SETUP MENU SCREEN

```
------------------------------ I/O DRIVER STATUS ------------------------------
 I/O  Ops:        0 | uc0:              |  I/O Command Parameters   stat: __
 TGT Chks:        0 | uc1:              | CDB: __ __ __ __ __ __    sense:
 INT D Er:        0 |-------------------|      __ __ __ __ __ __    __ __ __ __
 Bytes Wr:        0 | Wr/Ref: BPM       | xfer: DMARW    a.s.:OFF   __ __ __ __
 Bytes Rd:        0 |             0000  | s.l.ON  arb.HDW sel.SMA   __ __ __ __
 Bytes Cp:        0 | Rd Buf: BPM       | b.p.ON  b.w.OFF           __ __ __ __
 Cmp Ers.:        0 |             0000  | ha: 0   iid: 7   tid: 4   __ __ __ __
--------------SDS-1 MENU  (Jun 12 1986  FC=4)  SETUP FUNCTIONS------------------
Z:execute_all(1);                  I:iea("LOGC");             5:line_mode("S");
X:xfermode("DMARW  ",0x4000);      E:eea("LOGC");
7:ioto(    30);                    V:fixed(1);
F:arbmode("HDW ");                 N:autosense(0);
J:selmode("SMART");                W:busywait(0);
Y:parity(1);                       G:bus_logen(1);
Q:tid(   4);                       2:ackdelay(  0000);
D:iid(  00,   07);                 C:bcu(1);
L:lun(  0);                        3:statsen(1);
1:cntlbyte(  00);                  4:stats_window("G");
A:stats_reset("A ");               P:fillbyte(0x5A,  0000,0xFFFF);

    SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT
Select Menu:(,),U,B,...  Edit Parms:^E  Function Exec:Z,X,7,F,... Help:ALT-H
```

While in the edit mode, reference information or help on any of these functions can be accessed by moving the cursor to a parameter in the desired function and holding down the **ALT** key and pressing **H** (will be written as **ALT-H**). All Test/Documentation Library functions can be accessed directly through the edit mode (as described) or through the Table of Contents in the Help System. There are some functions in the Menu Interface that are not part of the Test/Documentation Library. These functions are internal Menu Interface functions and they will be described throughout this chapter in their respective places.

~MENU.2.1 EXECUTE ALL FUNCTION

In the SETUP menu, there is only one internal function called execute_all. This function will execute all of the SETUP functions listed in the SETUP menu when enabled (parameter set to 1).

In the menu initialization process, the execute_all function is checked. If it is enabled, all of the SETUP functions will be executed as part of the initialization. Otherwise, none of the SETUP functions are executed.

The user may also change the existing environment by editing the SETUP parameters and executing those functions individually or performing execute_all(1).

~MENU.3 BUFFER MENU

The BUFFER menu contains buffer related functions such as the various fill functions that can create several different types of data patterns in the selected buffer. There are other functions that allow the user to display, load and save buffers, and also have the ability to reset or set the buffer pointers. A typical BUFFER menu screen is shown below.

FIGURE ~MENU-F4. BUFFER MENU SCREEN

```
----------------------------- I/O DRIVER STATUS --------------------------------
 I/O  Ops:          0 | uc0:            |  I/O Command Parameters   stat: __
 TGT Chks:          0 | uc1:            | CDB: __ __ __ __ __ __     sense:
 INT D Er:          0 |-----------------|      __ __ __ __ __ __     __ __ __ __
 Bytes Wr:          0 | Wr/Ref: BPM     | xfer: DMARW    a.s.:OFF   __ __ __ __
 Bytes Rd:          0 |           0000  | s.l.ON  arb.HDW sel.SMA    __ __ __ __
 Bytes Cp:          0 | Rd Buf: BPM     | b.p.ON  b.w.OFF            __ __ __ __
 Cmp Ers.:          0 |           0000  | ha: 0   iid: 7   tid: 4    __ __ __ __
------------SDS-1 MENU  (Jun 12 1986  FC=4)  BUFFER FUNCTIONS-------------------
G:dmarst("R");                           W:fillbcw( 0000,0x0200,  0000,0xFFFF);
Y:dmarst("W");                           Q:overbcb( 00,0x0200,  0000,0xFFFF);
I:dmaset("R",  0000);                    A:overbcw( 0000,0x0200,  0000,0xFFFF);
2:dmaset_va("R",  0000000);              D:loadbuf("         ",  0000,  0000);
3:dmaset_vblk("R");                      V:savebuf("         ",  0000,  0000);
P:fillbyte(0x5A,  0000,0xFFFF);          E:setbuf("                ",  0000);
L:filld(  00,  0000,0xFFFF);             F:dispbuf("L  ",  0000,0x0010);
J:filli(  00,  0000,0xFFFF);             Z:dispbuf("R  ",  0000,0x0100);
N:fillpr(  0000,  0000,0xFFFF);          T:reset();
C:fillbcb(  00,0x0200,  0000,0xFFFF);X:xfermode("DMARW  ",0x4000);
1:fillk("00,00,00,00,00,00,00,00",  0000,0x0008); 4:setfill_buf("W");

   SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT
Select Menu:(,),U,B,...  Edit Parms:^E  Function Exec:G,Y,I,2,...  Help:ALT-H
```

While in the parameter edit mode, the functions listed below cannot be accessed directly through the Help System (via **ALT-H** keys) since there are no parameters associated with them, but they are listed in the SDS-1 Reference Manual in the Function Library Description (Appendix A) under their respective names:

dmarst	reset DMA pointer
reset	reset SCSI Bus/I/O Driver.

~MENU.4 RANDOM MENU

The RANDOM menu contains functions related to the random access devices. The following figure is a typical Random Menu.

FIGURE ~MENU-F5. RANDOM MENU SCREEN

```
────────────────────────────── I/O DRIVER STATUS ──────────────────────────────
 I/O  Ops:          0 | uc0:           |  I/O Command Parameters   stat: __
 TGT Chks:          0 | uc1:           | CDB: __ __ __ __ __ __    sense:
 INT D Er:          0 |────────────────|      __ __ __ __ __ __    __ __ __ __
 Bytes Wr:          0 | Wr/Ref: BPM    | xfer: DMARW   a.s.:OFF   __ __ __ __
 Bytes Rd:          0 |           0000 | s.l.ON  arb.HDW sel.SMA  __ __ __ __
 Bytes Cp:          0 | Rd Buf: BPM    | b.p.ON  b.w.OFF          __ __ __ __
 Cmp Ers.:          0 |           0000 | ha: 0   iid: 7   tid: 4  __ __ __ __
────SDS-1 MENU  (Jun 12 1986  FC=4)  RANDOM ACCESS DEVICE FUNCTIONS────────────
   4:blk_size(  0000);                   T:reset();          Z:rezero();
   A:format(0,0,  0,  0000);             X:seek10(  00000000);
   I:inc_blk(  0000);                    F:seekl(  00000000);
   D:ccs_modsel(  00,0);                 N:sense(  00);
   C:ccs_modsen(  00,0,  00);            2:set_blk(  00000000);
   1:random_blk(  00000000,  00000000); 3:set_len(  0000);
   L:random_len(  0000,  0000);          V:verify10(0,0,  00000000,  0000);
   J:readr_blk();                        G:writer_blk();
   Q:readr10(0,  00000000,  0000);       5:writer10(0,  00000000,  0000);
   P:readr10_blk();                      Y:writer10_blk();
   E:readrl(  00000000,  00); LC=  0001 W:writerl( 00000000,  00); LC= 0001

    SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT
 Select Menu:<,>,U,B,...  Edit Parms:^E  Function Exec:4,A,I,D,... Help:ALT-H
```

While in the parameter edit mode, the functions listed below cannot be accessed directly through the Help System (via **ALT-H** keys) since there are no parameters associated with them, but they are listed in the SDS-1 Reference Manual in the Function Library Description (Appendix A) under their respective names:

readr_blk	6-byte read command using predefined block and length
readr1Ø_blk	1Ø-byte read command using predefined block and length
reset	reset SCSI Bus/I/O Driver
rezero	rezero unit command
writer_blk	6-byte write command using predefined block and length
writer1Ø_blk	1Ø-byte write command using predefined block and length.

The **LC=** (shown after the **readrl()** and **writerl()** functions) is the loop-count parameter. This controls the number of times the function is to be executed. If the loop count is zero, the function will execute indefinitely until it is halted by the user through the **ESC** key. Otherwise, the function will execute the number of times defined. The largest finite loop count is ØxFFFF (or 65,535 decimal).

~MENU.5 SEQUENTIAL MENU

The SEQUENTIAL menu contains functions related to sequential access devices. A screen sample is shown below.

FIGURE ~MENU-F6. SEQUENTIAL MENU SCREEN

```
------------------------------ I/O DRIVER STATUS ---------------------------------
| I/O  Ops:           0 | uc0:                 |  I/O Command Parameters   stat: __ |
| TGT Chks:           0 | uc1:                 | CDB: __ __ __ __ __ __    sense:   |
| INT D Er:           0 |----------------------|      __ __ __ __ __ __    __ __ __ __|
| Bytes Wr:           0 | Wr/Ref: BPM          | xfer: DMARW   a.s.:OFF   __ __ __ __|
| Bytes Rd:           0 |               0000   | s.I.ON  arb.HDW sel.SMA  __ __ __ __|
| Bytes Cp:           0 | Rd Buf: BPM          | b.p.ON  b.w.OFF          __ __ __ __|
| Cmp Ers.:           0 |               0000   | ha: 0   iid: 7   tid: 4  __ __ __ __|
|----SDS-1 MENU  (Jun 12 1986  FC=4)  SEQUENTIAL ACCESS DEVICE FUNCTIONS-----------|
|   G:ldunlds(0,0,0);                          T:reset();                          |
|   D:modsels(  00);                           X:rewind(0);                        |
|   J:modsens(  00);                           N:sense(  00);                      |
|   I:prevmeds(0);                             P:space(0,   0000);                 |
|   A:readsl(  00000000); LC=   0001           V:verifys(0,   0000);               |
|   C:recbufds(  0000);                        W:writesl(  00000000); LC=  0001    |
|   L:releases(0,0);                           F:wrtfilm(  0000);                  |
|   E:reserves(0,0);                                                               |
|                                                                                  |
|    SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT           |
|Select Menu:(,),U,B,...  Edit Parms:^E  Function Exec:G,D,J,I,...  Help:ALT-H     |
-----------------------------------------------------------------------------------
```

While in the parameter edit mode, the function listed below cannot be accessed directly through the Help System (via **ALT-H** keys) since there are no parameters associated with them, but they are listed in the SDS-1 Reference Manual in the Function Library Description (Appendix A) under its name:

reset   reset SCSI Bus/I/O Driver.

The **LC=** (shown after the **readsl()** and **writesl()** functions) is the loop-count parameter. This controls the number of times the function is to be executed. If the loop count is zero, the function will execute indefinitely until it is halted by the user through the **ESC** key. Otherwise, the function will execute the number of times defined. The largest finite loop count is ØxFFFF (or 65,535 decimal).

~MENU.6 OTHER I/O DRIVER MENU

The I/O DRIVER menu contains other I/O Driver and miscellaneous functions. A screen of the other I/O DRIVER menu is shown below. The **io6()**, **io1Ø()** and **io12()** functions provide the user with the flexibility to create any vendor-unique SCSI commands.

FIGURE ~MENU-F7. I/O DRIVER MENU SCREEN

```
----------------------------------- I/O DRIVER STATUS -------------------------------------
| I/O  Ops:        0 | uc0:            |  I/O Command Parameters   stat: __               |
| TGT Chks:        0 | uc1:            | CDB: __ __ __ __ __ __    sense:                 |
| INT D Er:        0 |-----------------|      __ __ __ __ __ __    __ __ __ __            |
| Bytes Wr:        0 | Wr/Ref: BPM     | xfer: DMARW    a.s.:OFF   __ __ __ __            |
| Bytes Rd:        0 |            0000 | s.l.ON  arb.HDW sel.SMA    __ __ __ __            |
| Bytes Cp:        0 | Rd Buf: BPM     | b.p.ON  b.w.OFF            __ __ __ __            |
| Cmp Ers.:        0 |            0000 | ha: 0   iid: 7   tid: 4    __ __ __ __            |
|---------SDS-1 MENU  (Jun 12 1986  FC=4)  OTHER I/O DRIVER FUNCTIONS--------------------|
|         C:copy(  000000);                       N:sense(  00);                          |
|         Q:inquiry(  00);                        W:testur();                             |
|         T:reset();                              X:ucname(0,"            ");             |
|         V:recvdiag(  0000);                     Y:ucinc(0,  0000);                      |
|         D:senddiag(0,0,0,   0000);              Z:ucrst(0);                             |
|                                                                                         |
|1:io6( 00, 00, 00, 00, 00, 00); LC=  0001                                               |
|2:io10( 00, 00, 00, 00, 00, 00, 00, 00, 00, 00); LC=  0001                              |
|3:io12( 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00); LC=  0001                      |
|                                                                                         |
|         A:rptstats(1);                                                                  |
|                                                                                         |
|    SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT                  |
|Select Menu:<,>,U,B,...  Edit Parms:^E  Function Exec:C,Q,T,V,...  Help:ALT-H           |
-------------------------------------------------------------------------------------------
```

When in the parameter edit mode, the functions listed below cannot be accessed directly through the Help System (via **ALT-H** keys) since there are no parameters associated with them, but they are listed in the SDS-1 Reference Manual in the Function Library Description (Appendix A) under their respective names:

reset	reset SCSI Bus/I/O Driver
testur	test unit ready command.

The **LC=** (shown after the **io6()**, **io1Ø()** and **io12()** functions) is the loop-count parameter. This controls the number of times the function is to be executed. If the loop count is zero, the function will execute indefinitely until it is halted by the user through the **ESC** key. Otherwise, the function will execute the number of times defined. The largest finite loop count is ØxFFFF (or 65,535 decimal).

~MENU.7 MP MENU

The MP menu contains microprogramming functions. A typical screen is shown below. After every execution of a microprogramming function, the SCSI bus display is shown, unless the trace function is disabled.

FIGURE ~MENU-F8. MICROPROGRAMMING MENU SCREEN

```
------------------------------- I/O DRIVER STATUS ------------------------------------
 I/O  Ops:        0 | uc0:          |  I/O Command Parameters    stat: __
 TGT Chks:        0 | uc1:          | CDB: __ __ __ __ __ __      sense:
 INT D Er:        0 |---------------|      __ __ __ __ __ __      __ __ __ __
 Bytes Wr:        0 | Wr/Ref: BPM   | xfer: DMARW    a.s.:OFF    __ __ __ __
 Bytes Rd:        0 |          0000 | s.l.ON  arb.HDW sel.SMA     __ __ __ __
 Bytes Cp:        0 | Rd Buf: BPM   | b.p.ON  b.w.OFF             __ __ __ __
 Cmp Ers.:        0 |          0000 | ha: 0   iid: 7   tid: 4     __ __ __ __
---------SDS-1 MENU  (Jun 12 1986  FC=4)  MICROPROGRAMMING FUNCTIONS-----------------
A:arb1(  00);                N:datain1(  00000000,0);        I:msgin(  00);
J:arb2(  00);                4:datain4(  00000000,0);        G:msgout(  00);
L:arblose(  00);             5:datain5(  00000000,0);        X:resel();
W:awin_res(  00);            O:dataout0(  00000000,0);       6:sel1(  00);
Q:bfreearm();                V:dataout1(  00000000,0);       7:sel2(  00,  00);
H:bfreeck();                 Y:forcbusy();                   8:sel3(  00);
E:busrel();                  F:forceattn(0);                 9:sel4(  00,  00);
D:datain0(  00000000,0); C:forcperr(  00);                   Z:statin(  00);
1:cdb61( 00, 00, 00, 00, 00, 00);                            T:ureset();
2:cdb101( 00, 00, 00, 00, 00, 00, 00, 00, 00, 00);           P:disp_scsi_bus;
3:cdb121( 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00);

    SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT
Select Menu:<,>,U,B,...  Edit Parms:^E  Function Exec:A,J,L,W,... Help:ALT-H
```

While in the parameter edit mode, the functions listed below cannot be accessed directly through the Help System (via **ALT-H** keys) since there are no parameters associated with them, but they are listed in the SDS-1 Reference Manual in the Function Library Description (Appendix A) under their respective names:

bfreearm	bus free detection logic arm
bfreeck	bus free detection check
busrel	release bus
forcbusy	force test adapter BUSY on bus
resel	verify reselection by disconnecting TARGET
ureset	generate a SCSI reset pulse for more than 25 usec.

MENU.7.1 DISPLAY SCSI BUS FUNCTION

The internal Menu function, disp_scsi_bus, shows the state of the SCSI bus at the time of request. The highlighted values indicate the asserted signals. Below is a sample display:

BSY SEL data: *0000 0000 (00)* **REQ** ACK c/D i/O MSG **ATTN** RES

~MENU.8 FKEY MENU

The FKEY menu contains user-customized FKEY sequences. This provides the user with the ability to create short custom sequences which execute at a single keystroke. A typical screen is shown below.

FIGURE ~MENU-F9. FKEY MENU SCREEN

```
------------------------------ I/O DRIVER STATUS --------------------------------
 I/O  Ops:          0 | uc0:           |  I/O Command Parameters   stat: __
 TGT Chks:          0 | uc1:           |  CDB: __ __ __ __ __ __    sense:
 INT D Er:          0 |----------------|       __ __ __ __ __ __    __ __ __ __
 Bytes Wr:          0 | Wr/Ref: BPM    |  xfer: DMARW    a.s.:OFF  __ __ __ __
 Bytes Rd:          0 |          0000  |  s.l.ON  arb.HDW sel.SMA  __ __ __ __
 Bytes Cp:          0 | Rd Buf: BPM    |  b.p.ON  b.w.OFF          __ __ __ __
 Cmp Ers.:          0 |          0000  |  ha: 0   iid: 7   tid: 4  __ __ __ __
--------SDS-1 MENU  (Jun 12 1986  FC=4)  FUNCTION KEY SEQUENCE FUNCTIONS---------
 F1:test("                  "); LC=  0001 C:key select(F1 );
 F2:test("                  "); LC=  0001      D:display/edit/append;
 F3:test("                  "); LC=  0001      E:erase;
 F4:test("                  "); LC=  0001      Y:save_fkey("              ");
 F5:test("                  "); LC=  0001      L:load_fkey("              ");
 F6:test("                  "); LC=  0001 V:save_all_fkeys("              ");
 F7:test("                  "); LC=  0001 A:load_all_fkeys("              ");
 F8:test("                  "); LC=  0001 G:debugger("R");
 F9:test("                  "); LC=  0001 F:set_er_limits(  0000);
F10:test("                  "); LC=  0001

    SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT
Select Menu:<,>,U,B,...  Edit Parms:^E  Function Exec:F1,F2,F3,.. Help:ALT-H
```

Ten function keys (F1 through F1Ø) can be assigned to FKEY sequences. The maximum number of functions per FKEY sequence is also ten. The **LC=** is the loop-count parameter; it is explained in Section MENU.8.4.

~MENU.8.1 FUNCTION KEY SELECTION

Some FKEY menu functions use the current function key selected for execution. These functions are indented below the FKEY select line on the FKEY menu screen. To modify or select the function key, the function code associated with FKEY select (**C** key) is toggled from (F1 through F1Ø) for FKEY selection. Any function execution codes shown in inverse video are toggles.

The erase function will erase the entire sequence for the current FKEY selected.

The save_fkey function will save the FKEY selected to the specified file name.

The load_fkey function will load a previously saved function key sequence to the selected FKEY, erasing the previous contents of the selected FKEY.

The key select, erase, save_fkey and load_fkey functions are internal Menu functions.

~MENU.8.1.1 DISPLAY/EDIT/APPEND MODES

MENU.8.1.1.1 DISPLAY MODE

The display/edit/append function will display the selected FKEY sequence. From this screen, the user may enter the append or edit mode, or return to the FKEY menu. Below is an example of this screen.

FIGURE ~MENU-F10. FKEY SEQUENCE DISPLAY

```
------------------------------ I/O DRIVER STATUS -------------------------------------
 I/O  Ops:         0 | uc0:           |  I/O Command Parameters    stat: __
 TGT Chks:         0 | uc1:           |  CDB: __ __ __ __ __ __     sense:
 INT D Er:         0 |----------------|       __ __ __ __ __ __     __ __ __ __
 Bytes Wr:         0 | Wr/Ref: BPM    |  xfer: DMARW    a.s.:OFF   __ __ __ __
 Bytes Rd:         0 |           0000 |  s.1.ON  arb.HDW sel.SMA   __ __ __ __
 Bytes Cp:         0 | Rd Buf: BPM    |  b.p.ON  b.w.OFF            __ __ __ __
 Cmp Ers.:         0 |           0000 |  ha: 0   iid: 7   tid: 4   __ __ __ __
------SDS-1 MENU  (Jun 25 1986  FC=4)  FUNCTION KEY SEQUENCE FUNCTIONS---------
 1  test("Wr/R/Cmp 256blks");
 2  xfermode("HSHCV   ",0x4000);
 3  fillpr(  0000,   0000,0xFFFF);
 4  dmarst("W");
 5  writer10(0,   00000000,0x4C00);
 6  writer10(0,0x4C00     ,0x70   );
 7  dmarst("W");
 8  readr10(0,   00000000,0x4C00);
 9  readr10(0,0x4C00     ,0x70   );
 10 loop back to line 04;    0001 times



 Create/Edit Functions:^A  Edit Parameters:^E  Exit to FKEY:ESC  Help:ALT-H
```

MENU.8.1.1.2 APPEND MODE

To enter the append mode, press **CTRL-A (^A)**; use the same keys to exit this mode. Once in append mode, an inverse video right arrow "cursor" will appear to the right of the line numbers. This indicates where the next function is to be added. It also indicates where the next insertion or deletion will occur. The user may move this "cursor" by using the **Up** or **Down Arrow** keys. This "cursor" will stay within its sequence limits.

To build the FKEY sequence, enter the append mode and choose one of the menus displayed on the menu line by its menu code. Once the menu has been picked, the screen will display the chosen menu. At this point, a function can be picked by entering its execution code and the screen will display the current sequence. If the wrong menu was picked, the user may skip picking a function or exit out of the append mode. Basically, the append process is selecting the menu and selecting the function. These steps may be repeated until the user exits the append mode or the maximum number of functions for sequences has been reached.

If a function is to be inserted, move the "cursor" to where the function is to be inserted and pick the menu and function as in the append process. The following functions in the sequence are moved down to make room for the inserted function.

To delete functions from the FKEY sequence, move the "cursor" to the function to be deleted and press the **^D** keys. If there are any functions following, they are moved up, so that the functions in the sequence are contiguous.

There are other append functions. One of them is a Loopback instruction to the function sequence. This allows a function to go back to a line for the specified number of times. Nested loopbacks are also possible, but be careful of overlapping loops since MENU does not detect them.

The Goto instruction will allow one FKEY sequence to transfer to another FKEY sequence. This will allow chaining of FKEY sequences.

After the append mode is terminated, the display/edit/append screen will appear, displaying the current sequence.

MENU.8.1.1.3 EDIT MODE

To edit the parameters of a function sequence, enter the edit mode with the **CTRL-E** (**^E**) keys. To exit this mode, enter the same keys (**^E**). The edit mode is the same as the edit menu display except that an FKEY sequence is being edited and not the menu display. Both of these edit modes operate in the same fashion. When the edit mode is terminated, the display/edit/append screen will appear with the current sequence.

MENU.8.1.1.4 RETURN TO FKEY MENU

The **ESC** key will return from the display/edit/append mode to the FKEY menu.

~MENU.8.2 SAVE/LOAD FKEY SET FUNCTIONS

The save and load fkey set functions are internal Menu functions.

MENU.8.2.1 SAVE FKEY SET FUNCTION

In addition to saving one FKEY sequence, all ten FKEYs can be saved to disk by specifying the file name (or path name, if desired) and executing the appropriate function (save_all_fkeys).

MENU.8.2.2 LOAD FKEY SET FUNCTION

To load the FKEY sequence set, enter the file name or path name containing the set and execute the load_all_fkeys function. The contents of the function sequence will be replaced with the loaded function key set.

~MENU.8.3 DEBUGGER STATE

FKEY sequence execution can be executed under the SDS-1 Debugger in the single step or run mode, by setting the debugger value to S or R in the FKEY menu.

~MENU.8.4 FKEY EXECUTION LOOP COUNT

Each FKEY sequence can be executed in a loop by setting the loop count (LC) to the number of loops to perform. If the loop count is Ø, the FKEY sequence will execute indefinitely until it is halted by the user through the **ESC** key. Otherwise, the function key sequence will execute the specified number of times in the loop-count field. The largest finite execution loop count is ØxFFFF (65,535 decimal).

~MENU.8.5 STOPPING FKEY SEQUENCE EXECUTION

Use the **ESC** key to halt function execution and enter the Debugger TRACE state. To return from the TRACE state to menu, hit the **ESC** key a second time.

~MENU.9 OTHER/EXIT MENU

The OTHER/EXIT menu contains functions that are Menu Interface related. All of these functions are internal Menu functions. A typical screen is shown on the next page.

FIGURE ~MENU-F11. OTHER/EXIT MENU SCREEN

```
┌──────────────────────────── I/O DRIVER STATUS ───────────────────────────────┐
│ I/O  Ops:          0 │ uc0:              │  I/O Command Parameters   stat: __ │
│ TGT Chks:          0 │ uc1:              │ CDB: __ __ __ __ __ __    sense:   │
│ INT D Er:          0 │───────────────────│      __ __ __ __ __ __             │
│ Bytes Wr:          0 │ Wr/Ref: BPM       │ xfer: DMARW    a.s.:OFF  __ __ __ __│
│ Bytes Rd:          0 │             0000  │ s.l.ON  arb.HDW sel.SMA  __ __ __ __│
│ Bytes Cp:          0 │ Rd Buf: BPM       │ b.p.ON  b.w.OFF          __ __ __ __│
│ Cmp Ers.:          0 │             0000  │ ha: 0   iid: 7   tid: 4  __ __ __ __│
├──────────SDS-1 MENU  (Jun 12 1986  FC=4)  OTHER/EXIT FUNCTIONS───────────────┤
│                       P:save_pars("                 ");                      │
│                       L:load_pars("                 ");                      │
│                       V:save_exit("                 ");                      │
│                       1:exit();                                              │
│                                                                              │
│                       W:screen_swap(ON);                                     │
│                       N:init_menu(RANDOM     );                              │
│                       A:trace(ON);                                           │
│                                                                              │
│   SETUP  BUFFER  RANDOM  SEQUENTIAL  I/O DRIVER  MP  FKEY  OTHER/EXIT        │
│Select Menu:<,>,U,B,...  Edit Parms:^E  Function Exec:P,L,V,1,...  Help:ALT-H │
└──────────────────────────────────────────────────────────────────────────────┘
```

~MENU.9.1 SAVE PARAMETERS FUNCTION

To save all the menu parameters (including all ten of the FKEY sequences), use the save parameter function (save_pars) with a specified disk file name.

To invoke the Menu Interface with these same parameters, enter the saved file name on the command line following the MENU command or use the load_pars function described in the next section.

~MENU.9.2 LOAD PARAMETERS FUNCTION

In the load_pars function, a file saved from a save_pars function can be loaded to the Menu Interface.

~MENU.9.3 SAVE AND EXIT FUNCTION

The save_exit function will save all parameters and FKEY sequences to the specified file name and terminate the Menu Interface session.

~MENU.9.4 EXIT FUNCTION

The exit function will terminate the Menu Interface session and returns to DOS.

~MENU.9.5 SCREEN SWAP FUNCTION

The screen_swap function will enable or disable screen swapping between the Debugger (Trace Display) and the Menu Display. When screen swap is enabled, the screen will swap to the menu display while the function is executing and will swap back to the menu display after execution. When screen swap is disabled, the screen will not return to the menu after execution, but will continue to show the Trace Display. To indicate that the function has finished, the current menu name and cursor will appear at the top left corner of the window.

~MENU.9.6 INITIAL MENU SCREEN DISPLAY FUNCTION

The init_menu function sets the initial menu screen display. To set the initial menu screen, toggle to the menu screen parameter until the new default screen name appears and save the parameters to a file. Then on the next Menu invocation, load this saved file. The new default screen should appear after Menu initialization.

~MENU.9.7 TRACE FUNCTION

To enable or disable the Trace Display during all menu executions, use the trace function. This feature provides an increase in execution speed. Disabling the trace will also inhibit the SCSI bus display on microprogramming functions.

~MENU.10 MENU INTERFACE ERRORS

~MENU.10.1 NO SPACE FOR PARAMETERS

There is not enough space in the structure to enter parameters of the function. The function and its parameters are not entered into the sequence. The user can delete other functions or FKEY sequence(s) to free up space.

~MENU.10.2 NO SPACE FOR FUNCTION

There is no more space in the structure to enter another function. The user can delete other functions or FKEY sequence(s) to free up space.

~MENU.10.3 FILE I/O ERROR

Error occurred on file I/O. Below are possible causes:

- file name was not specified
- incorrect spelling of file name
- path name incorrect

~MENU.1Ø.4 VERSION MISMATCH

The load file contains a version that cannot be converted. The user can rebuild the save file with current menu version for compatibility.

~MENU.1Ø.5 MAXIMUM NUMBER OF FUNCTIONS

The maximum number of functions has been reached for a function sequence. No more functions can be added to this sequence. The user can use the Goto instruction to continue the FKEY sequence to another FKEY.

~MENU.1Ø.6 INCOMPATIBLE FILE TYPES

File types and menu version must be the same in order for loading to be successful. This error indicates that the file type to load is not the correct type requested in the load. There are three different file types: single FKEY sequence (FF), all FKEY sequences (AF), and all Menu parameters and FKEY sequences (PF).

~MENU.1Ø.7 FILE DOES NOT EXIST

File name specified for initial loading of parameters does not exist; the cause may be due to incorrect spelling of the file name. The initialization process of Menu will continue with default values. Once this process is done, try loading the correct file name using the load_pars function in the OTHER/EXIT menu.

~MENU.1Ø.8 ERROR IN CONVERTING FILE

When loading a file with a lower version number, MENU will automatically convert the saved file by renaming it with a .BAK extension, and then convert it to the current version with its original name. Once the conversion is done, there will be two files: the old version with the .BAK extension and the current version with the original file name.

If an error occurs during this process, the user may recover the older version of the saved file and try again or run the saved file with an older version of MENU that matches its version and re-saving it. Below is a list of where this error occurs:

- unsuccessful deletion of a previous .BAK file before renaming the current saved file
- unsuccessful renaming of the saved file

~MENU.1Ø.9 PC MOUSE NOT INSTALLED

If the mouse is to be used, the mouse driver, MSMOUSE.COM, must be executed before using MENU. MSMOUSE should be part of the AUTOEXEC.BAT file, check to be sure that the mouse driver is included in this file.

~MENU.10.10 TEMPORARY FILES HAVE NOT BEEN DELETED

Before the Menu Interface can run properly, all of the .TMP files in the current directory must be deleted. The error occurred while deleting those files. This is a warning to let the user know that .TMP files do exist. The user should exit MENU and delete those files through DOS commands (DEL or ERASE) and then enter MENU again.

~MENU.10.11 FILE NAME ERROR

If the file name has an extension of .TMP or .BAK, MENU will sooner or later delete it or change its contents. The user should rename the file with a different extension.

~MENU.10.12 INVALID STRING POINTER; MEMORY NOT FREED

An error occurred during deletion of FKEY functions, memory was not freed.

~MENU.11 MOUSE OPERATIONS WITH THE MENU INTERFACE

The mouse may be used with MENU to access areas on the menu screens. It may be used to change Menu screens or getting around and/or moving the cursor in the edit or append mode. The following table defines the mouse movements and buttons in the different Menu Interface states.

TABLE ~MENU-T1. MOUSE MOVEMENT AND BUTTON DEFINITIONS

MENU INTERFACE STATES	MOUSE MOVEMENTS	BUTTON DEFINITIONS LEFT=^E MIDDLE=<- RIGHT=->
At Menu Screens	left & right movements = left & right arrows for menu selection	^E = enter/exit edit mode <- & -> = menu selection
At FKEY sequence display screen	mouse movements ignored	^E = enter/exit edit mode
In Edit Mode	left & right movements = UP & DOWN ARROWS for moving to previous or next parameter up & down movements = PAGE UP & PAGE DOWN for previous & next line	^E = exit edit mode <- & -> = movement within fields and also for toggling parameters
In Append Mode	up & down movements = "cursor" movements	buttons ignored

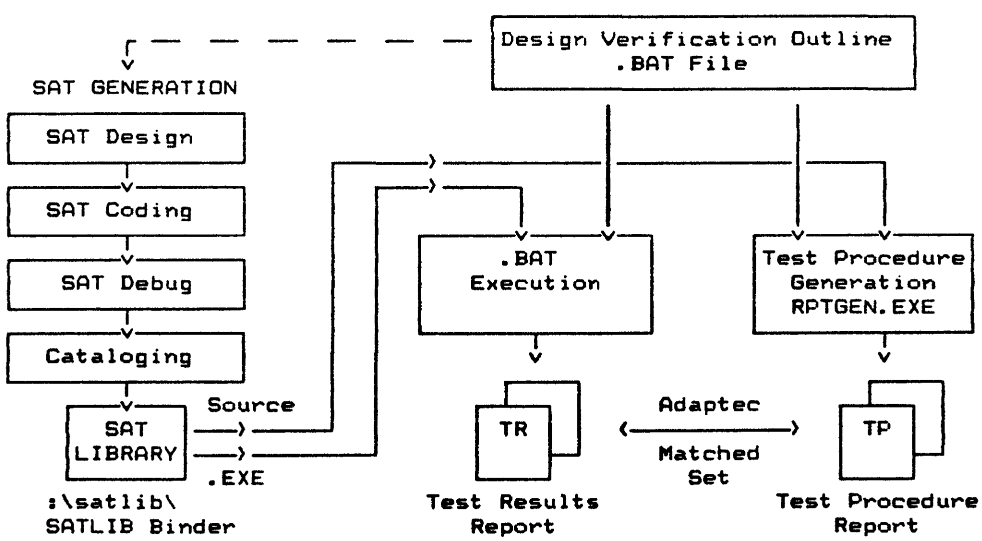
If mouse is not setup, refer to Section RPTG.2.1.2.5.1 for mouse setup procedures or the PC PAINT PLUS reference manual.

~SAT.Ø STAND-ALONE TEST (SAT) GENERATION PROCESS

~SAT.1 INTRODUCTION

As with any large task, a Design Verification Test must be broken down into smaller manageable pieces. The SDS-1 System uses the Stand-Alone Test (SAT) as its basic Design Verification Building Block. As the name implies, the SAT will execute by itself providing a predefined pass/fail result. The Test and Documentation Function Library contains initialization (setup), execution, analysis and documentation functions necessary to accomplish the test at hand. Figure SAT-F1 shows a flow diagram of the SDS-1 Development Process. This is a structured approach to debugging, performance testing, and design verification/device qualification of SCSI peripheral devices. This section concentrates on the SAT Generation Portion of the SDS-1 Development Process.

FIGURE ~SAT-F1. SAT COMPONENT OF DESIGN VERIFICATION PROCESS



The SDS-1 System provides two types of "execution" in the "test experiment." The I/O Driver execution environment provides a high-level interface with SCSI commands. It also provides system environment and multihost emulation. The microprogramming execution environment provides a low-level interface with precise control over SCSI commands. It also provides a way to test response to forced error conditions. Figures SAT-F2 and SAT-F3 are pictures of the execution interfaces with examples of test function names. Both of these environments use functions from the Test and Documentation Function Library.

FIGURE ~SAT-F2. I/O DRIVER EXECUTION INTERFACE

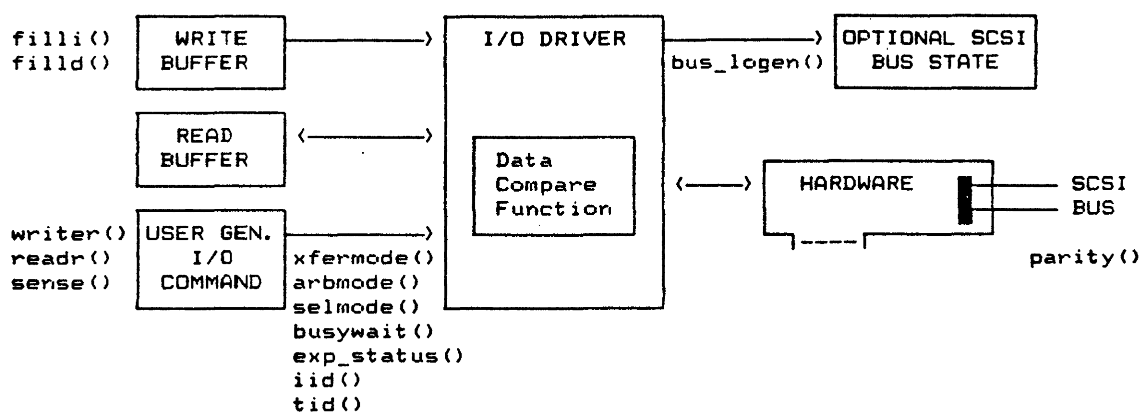
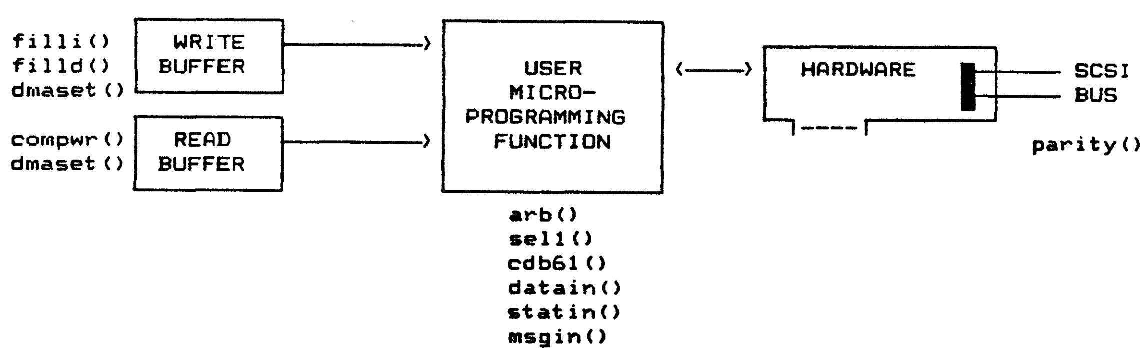


FIGURE ~SAT-F3. MICROPROGRAMMING EXECUTION INTERFACE



(THIS PAGE INTENTIONALLY LEFT BLANK)

SAT.2 SAT DESIGN AND CODING

This section will walk-through a creation of a SAT program. Before the walk-through, examples will be shown to familiarize the user with the concept of SAT. The following figure displays the contents of the SAT:

```
/* REVISION HISTORY
   TEST PROCEDURE DEFINITION
   -- GROUP (SECTION)
      -- PARAGRAPH
*/
   SETUP                  |
   EXECUTION              |
   ANALYSIS               |
      .                   |              Test Code
      .                   |--   (Library Function Calls)
      .                   |
   SETUP                  |
   EXECUTION              |
   ANALYSIS              _|
```

Below is a very simple SAT, which only uses a documentation and execution call:

```
user_test() {
    test("This is a very simple SAT");
    reset();
}
```

The test function performs library initializations and provides the test title for the Test Results report. And the reset function will reset the SCSI bus and initialize the I/O Driver.

The SAT may include report generator operators (**-GT=** and **-PT=**) and documentation functions (**group()** and **paragph()**) for report purposes (to be discussed in the next section and RPTG section). An expanded example of the simple SAT is shown below:

```
user_test() {
    test("This SAT uses the -GT= and -PT= operators");

/* -DOC
   -GT="Perform RTFL Function"
   -DOC
*/
    group("Perform RTFL Function");

/* -DOC
   -PT="Reset Function"
   -DOC
*/
    paragph("Reset Function");
    reset();
}
```

SAT.2.1 USER TEMPLATE FOR SAT

Before coding, look at the Blank SAT Template shown in Figure SAT-F4. This template is a guide to help create the SAT. Notice that the test code and procedure appears in the SAT. The report generator operators (**-DOC**, **-REV**, and **-COD**) control the format of the Test Procedures report. These operators should occur as a pair (start and stop operators):

```
-DOC
    documentation line(s) are inserted
    between the -DOC operators
-DOC

-COD
    code line(s) are inserted
    between the -COD operators
-COD

-REV
    revision log line(s) are inserted
    between the -REV operators and
    this operator must appear within
    the -DOC operators
-REV
```

There are **-GT**= and **-PT**= operators to define the group and paragraph titles.

All of these operators are embedded within the comment lines. Comments are the characters between the "/*" and "*/" that the "C" compiler ignores. Comment lines are usually used for documentation purposes. When reports are generated, the Report Generator will scan through the comment lines for its operators. These operators are described in more detail in Section RPTG.2 .

This Blank SAT Template contains two templates: the Group Documentation Template and the Paragraph Documentation Template. The Group Template sets up for a group test with its first paragraph (or test). The Paragraph Template is used to add additional tests to the group(s).

FIGURE ~SAT-F4. BLANK STAND-ALONE TEST TEMPLATE (BLANKSAT.C)

```
                                        /* Blank Stand Alone Test Template */
/* -DOC
-REV
           Created:
 Initial Release:
          Revision:
-REV
Introduction to Stand-Alone Test:
     Definition / Purpose of Test
          text
           :
          text
-DOC */
                                        /* -COD */
                                        /* Start of SAT */
user_test()
{
     test("User Test Title");
          insert any one-time test initialization here
                                        /* -COD */

                                        /* Group/1st Paragraph Template */
/* -DOC
-GT="Subtest Title (Group Level)"
        insert subtest (group) description text here

-PT="1st Paragraph Test Title"
        insert 1st paragraph description text here
-DOC */
                                        /* -COD */
     group("Subtest Title");
                                        /* start of group code */
          insert group setup or initialization code here
                                        /* end of group code  */
     paragph("1st Paragraph Test Title");
                                        /* start 1st paragraph code */
          insert paragraph test code here
                                        /* end 1st paragraph code */
                                        /* -COD */

                                        /* Additional Paragraph Template */
/* -DOC
 -PT="Paragraph Test Title"
        insert paragraph description text here

-DOC */
                                        /* -COD */
     paragph("Paragraph Test Title");
          insert code here
                                        /* -COD */

}
                                        /* end of Stand-Alone Test Program */
```

SAT.2.2 SAT PROGRAM CREATION

The first step in SAT program creation is to specify or design the test. After the SAT has been specified, the next step is to find the functions in the Test Function Library that meet your specifications.

The SATs will be developed on Sidekick's editor called Notepad. The editor is one of Sidekick's features.

Invoke Sidekick by pressing the **CTRL** and **ALT** keys simultaneously. When these keys are pressed again, the Sidekick window will disappear or reappear since these keys are a toggle to enter and exit Sidekick. A list of Sidekick features are displayed in the Sidekick window. To access Notepad, press the **N** or **F2** key or move the **up** or **down arrow** keys to the Notepad line and hit the **return** key.

When in Notepad, press the F3 key for new file and type in the SAT file name. The SAT file name must have an extension of **.C**, such as **TEST.C**, so the compiler will recognize this file as a "C" source file. If this file name already exists, the user may rename this file, or if the existing file is not needed anymore, delete it.

To copy the user's template, press the CTRL and **K** keys simultaneously and then the **R** key and type in the file name **BLANKSAT.C** . If this file is not present, type in the template in Figure SAT-F4 while following the step-by-step instructions.

Sidekick's Notepad basically uses the same control keys as MicroPro's WordStar to edit files. In addition, Sidekick also uses the arrows located on the right side of the keyboard to move the cursor.

Below are the step-by-step instructions to generate a Stand-Alone Test for use with the SDS-1 System, using the template:

1. Fill in the Revision Log information found in the SAT template (do not forget to include the report generator operators if generating from scratch):

```
/* -DOC
-REV
        Created:
Initial Release:
       Revision:
-REV
```

2. Describe the Stand-Alone Test Function and any other notes or messages after the second "**-REV**" and before the ending "**-DOC */**" lines, fill in the following:

```
                                          /* Expand definition of SAT:
                                             Definition/Purpose  */
Introduction to Stand-Alone Test:
    Definition/Purpose of Test
        text
         :
        text
-DOC */
```

3. If there are no external variable declarations, **#include** or **#define** statements, the **user_test()** line must be the first noncomment line in the SAT. This will define the function as a SAT function. The brace, {, on the next line indicates the start of the SAT. There is also a closing brace, }, on the last SAT line to end the SAT. The main body of the SAT is located between these braces which contains function calls to the Test and Documentation Function Library. If variables need to be declared, they should be declared before their use. Several variable data types can be declared, refer to the "C" Reference Manual for more information. The next line(s) in the SAT following the opening brace should define any variables. Also define the test title (by **test**() function which also performs SAT initialization) for the Test Results report (remember that "C" statements and statements within braces must end with a ;):

```
                                          /* -COD */
                                          /* Start of SAT */
user_test()
{
    test("User Test Title");
        insert any one time test initialization here
                                          /* -COD */
```

4. The main body of the SAT should contain function calls to the Test and Documentation Function Library. Each function call must contain its function name and its arguments. The arguments must appear within the parentheses; if there are no arguments, the parentheses must still exist to indicate a function call. Each function call statement must end with a ;. Using "C" statements (such as, **for, if, while, ...**), will allow more flexibility in the SAT programs. There are examples of function calls, **for, if** and **while** statements in Figure SAT-F5. Some of these statements are briefly described in section SAT.2.2.1, refer to the "C" Reference Manual for more detailed information.

   Define Subtest (groups) and fill in Group/1st Paragraph Documentation Templates:

```
                                        /* Group/1st Paragraph Template */
/* -DOC
-GT="Subtest Title (Group Level)"
      insert subtest (group) description text here

-PT="1st Paragraph Test Title"
      insert 1st paragraph description text here
-DOC */
                                        /* -COD */
    group("Subtest Title");
                                        /* start of group code */
        insert group setup or initialization code here
                                        /* end of group code  */
    paragph("1st Paragraph Test Title");
                                        /* start 1st paragraph code */
        insert paragraph test code here
                                        /* end 1st paragraph code */
                                        /* -COD */
```

5. For additional paragraph tests, a Paragraph Documentation Template has been provided. Your SAT program should look similar to the OBBWRCV.C Code Listing (see Figure SAT-F5). Copies of the Group/1st Paragraph Documentation and the Paragraph Documentation Templates can be made throughout the SAT when needed. Remember to end the SAT with the closing }, since this indicates the end of the SAT program.

```
                                        /* Additional Paragraph Template */
 /* -DOC
 -PT="Paragraph Test Title"
      insert paragraph description text here

 -DOC */
                                        /* -COD */
     paragph("Paragraph Test Title");
        insert code here
                                        /* -COD */

 }
                                        /* end of Stand Alone Test Program */
```

A Stand-Alone Test program may have many groups and under each group, many paragraphs. Note that the first paragraph of each group is found in the Group Documentation Template.

When the SAT program has been entered, the SAT file should be saved by entering the following:

F2 key

or

CNTL and **K** and then **D** key.

And then to exit Sidekick, enter:

ESC key

or

CNTL and **ALT** keys.

~SAT.2.2.1 "C" NOTES

In "C," a sequence of characters enclosed by " " is a character string. Hexadecimal numbers are noted by a preceding **0**x or **0X** (zero-x or zero-X). Octal numbers are preceded by a **0** (zero). If neither exists, "C" assumes that a number is decimal.

To briefly explain the **for** statement, there are three expressions separated by semicolons and enclosed in parentheses. The first expression within the parentheses is only performed once to initialize the loop. The second expression is a condition which is checked before each iteration. As long as this condition is true, the loop will execute. The last expression is executed after each loop iteration. In multi-statement loops, the loop is started with a '{' and ends with a '}.'

In "C", another way to accomplish looping of statements is the **while** statement. A condition within the parentheses following the keyword **while** is checked. If the condition is true, the statements within the **while** statement will be executed and the condition checked after each iteration. As long as the condition is true, the execution of these statements will continue. Otherwise, if it is false, the looping will end or if false to begin with, it will skip the **while** statements.

In the **if** statement, the condition within parentheses is checked. If it is true, the rest of the **if** statement is executed. Otherwise, if it is false, the **if** statement is skipped. An **else** statement may follow the **if** statement. In this case, the **else** statement will only execute if the **if** statement was not true.

~SAT.2.3 TEST & DOCUMENTATION FUNCTION LIBRARY

The Test and Documentation Function Library contains the routines available for the SAT programs. The library contains functions for initialization, execution, analysis and documentation of the SAT programs. Each of these functions is explained in detailed in Appendix A.

SAT.2.4 COMPILATION AND LINKAGE OF SAT

Once the stand-alone test has been written, the next step is to compile and link it. The "C" compiler is used to link the Test and Documentation Function Library. The SDS-1 System contains a batch file that will build the executable SAT file and link it. To run this batch file, enter **MKSAT** along with the filename without the .C extension. For example, if the file name was **SATNAME.C**, enter:

C>**MKSAT SATNAME**

After successful completion of **MKSAT**, an executable file called **SATNAME.EXE** is generated. If errors occur during this step, refer to the Microsoft User's Guide, Appendix E.

FIGURE ~SAT-F5. OBBWRCV.C CODE LISTING

```
/*-DB=;
-DOC
; -REV
;          Created:  01-16-86
; Initial Release:  N.A.
;         Revision:  1.000
;                    06-17-86  Enable parity
; -REV
;
;Purpose:  Demostrates OBB virtual memory, _blk functions and
;          variable ack delay
;
;Procedure:  1. Use get_byte() function to determine block limits
;            2. Read/Write Testing
;                a. Fill drive via HSHCV mode with write10() func
;                b. Read entire drive using _blk functions
;                c. Read with random starting address and lengths
;                d. Time reads in sequential manner
;                e. Time reads with random starting addresses
;                f. Time loop with everything random
;
;   System #1  Host i.d. = 7;
;              Target i.d. = 4;
;
;Functions Tested:    set_blk
;                     random_blk
;                     inc_blk
;                     set_len
;                     random_len
;                     inc_len
;
-DOC */
                                   /* Constant Definitions */
     #define HOST_ID 0x07
     #define TARGET_ID 0x04

user_test()
{
                                   /* Variable Definitions */
    int i;                         /* i variable */
    unsigned long last_block_num;  /* last block number on drive */
    unsigned long f_bw, f_br, f_bc, f_ce;  /* stats variables */
    unsigned block_size;           /* drive block size */
    unsigned long new_start;       /* new starting block address */
    unsigned long down_count;      /* length of disk */
    unsigned long start_blk;       /* starting block */
    unsigned long block;           /* block */
    unsigned long get_f_stats();   /* function status */
    unsigned len,akd;              /* length & ack delay variables */
    unsigned op_type;              /* operation type */
    unsigned tv;                   /* timer value */
    char dummy[100];               /* dummy string */
```

```
test("Random Function Testing");
group("Self Configuration Example");
                                /* -DOC
                                ; -GT="Self-Configuration Example"
                                ;
                                ;Demonstrate get_byte() function
                                ;determine block limits
                                ;
                                ;  -DOC */
xfermode("DMARW",0x100);        /* DMARW mode w/0x100 buf size */
reset();                        /* reset I/O Driver and SCSI bus */
ioto(600);                      /* long time-out w/two systems
                                   competing for bus */
bcu(1);                         /* buffer/command frame update */
arbmode("HDW");                 /* hardware arbitration */
selmode("SMART");               /* select SMART mode */
parity(1);                      /* SCSI parity enabled */
bus_logen(1);                   /* state bus log enabled */
ackdelay(0x0000);               /* 0 ack delay */
statsen(1);                     /* statistics enabled */
tid(TARGET_ID);                 /* set target ID */
iid(0,HOST_ID);                 /* set initiator ID */
lun(0);                         /* logical unit number is 0 *
iea("LOGH");                    /* log and halt on error */
readcap(0,01,0);                /* read capacity */
last_block_num =  ((unsigned long)get_byte("R",0) << 24) +
                  ((unsigned long)get_byte("R",1) << 16) +
                  ((unsigned long)get_byte("R",2) <<  8) +
                  (unsigned long)get_byte("R",3);

sprintf(dummy,"Drive Parameters:  Last Block Address = 0x%lX",
    last_block_num);
logp(dummy);                    /* print last block address msg */
block_size =  ((unsigned)get_byte("R",6) <<  8) +
               (unsigned)get_byte("R",7);

sprintf(dummy,"                           Block Size = 0x%X",
    block_size);
logp(dummy);                    /* print block size msg */


group("Read/Write Testing");

paragph("Fill Drive via HSHCV");
                                 /* -DOC
                                 ; -GT="Read/Write Testing"
                                 ;
                                 ; -PT="Fill Drive via HSHCV"
                                 ;
                                 ; Fill Drive with write10() cmd
                                 ; using HSHCV transfer mode
                                 ;
```

```
                                ; -DOC */
xfermode("HSHCV",0x4000);       /* set HSHCV mode & buffer size */
fillpr(0x87,0,0x4000);          /* fill buffer */
down_count = last_block_num +1L;  /* number of blocks */
start_blk = 0L;                 /* starting address */
while (down_count > 0xFFFFL) {  /* separate write commands if
                                   greater than 0xFFFF */
    writer10(0,start_blk,0xFFFF);  /* write maximum allowed */
    start_blk = start_blk + 0xFFFFL;  /* mod starting addr */
    down_count = down_count - 0xFFFFL; /* decrement blk cnt */
}
                                /* handle last write */
writer10(0,start_blk,(unsigned)down_count);  /* filled disk */
rptstats(1);                    /* report stats with header on */

paragph("Read Entire Drive Using _blk commands");
                                /* -DOC
                                ; -PT="Read Drive w/_blk cmds"
                                ;
                                ; Read and Compare Entire Disk
                                ; using _blk command and HSHCV mode
                                ; of transfer
                                ;
                                ; -DOC */
blk_size(block_size);           /* set block size */
stats_reset("ALL");             /* reset global stats */
set_blk(0x01);                  /* start at block zero */
set_len(0XFFFF);                /* read 0xFFFF blocks at a time */
dmaset_vblk("W");               /* set the virtual starting addr */
down_count = last_block_num + 1L;  /* get number of blocks */
while (down_count > 0xFFFFL) {  /* as with the writes, separate
                                   if block number greater than
                                   0xFFFF */
    readr10_blk();              /* read blocks */
    inc_blk(0xFFFF);            /* increment by 0xFFFF */
    down_count = down_count - 0xFFFFL;  /* decrement blk cnt */
}
set_len((unsigned)down_count);  /* handle last read */
readr10_blk();                  /* read blocks */
rptstats(1);                    /* report stats with header on */

                                /* Demonstrate get_f_stats() */
f_bw = get_f_stats("BW");       /* get bytes written */
f_br = get_f_stats("BR");       /* get bytes read */
f_bc = get_f_stats("BC");       /* get bytes compared */
f_ce = get_f_stats("CE");       /* get compare errors */

                                /* print stats to log device */
sprintf(dummy,"Last Read Command Statistics:");
logp(dummy);
sprintf(dummy,
    "                              Bytes Written = 0x%81X",
```

```
     f_bw);
logp(dummy);
sprintf(dummy,
     "                                Bytes Read = 0x%8lX",
     f_br);
logp(dummy);
sprintf(dummy,
     "                            Bytes Compared = 0x%8lX",
     f_bc);
logp(dummy);
sprintf(dummy,
     "                            Compare Errors = 0x%8lX",
     f_ce);
logp(dummy);

paragph("Read with Random Starting Addresses and Lengths");
                             /* -DOC
                             ; -PT="Read w/Random Addrs & Lens"
                             ;
                             ; Perform 100 read operations with
                             ; random starting addresses and
                             ; lengths
                             ;
                             ; -DOC */
stats_reset("ALL");          /* reset global statistics */
for (i =1; i <= 100; i++) {
     len = random_len(1,0x1000);  /* transfer length limit */
     block = random_blk(0L,last_block_num-(unsigned long)len+1);
     dmaset_vblk("W");        /* set memory pointer */
     readr10_blk();           /* perform read */
                              /* check for transfer length */
     f_br = get_f_stats("BR");  /* check for read failure */
     if (f_br != (unsigned long)block_size*(unsigned long)len) {
          fail();
          sprintf(dummy,
          "Number of bytes read = 0x%08lX; Should be = 0x%08lX;",
               f_br,(block_size *len));
          logp(dummy);        /* print to log device */
     }
}
rptstats(1);                 /* report global stats */

paragph("Timed Reads (three minutes) in Sequential Manner");
                             /* -DOC
                             ; -PT="Time Seq Reads (3 mins)"
                             ;
                             ; Utilizing the user timer to
                             ; determine the number of
                             ; operations and bytes read which
                             ; can be executed in three minutes
                             ;
                             ; -DOC */
```

```
stats_reset("ALL");           /* reset statistics */
tmrset(0x0);                  /* set timer to start at 0 */
tmrstart("Up");               /* start timer counting up */
rpttmr();                     /* output timer to log */
tv = tmrvalue();              /* get current time */
sprintf(dummy,"Timer Value = 0x%04X",tv);  /* display timer */
set_len(0x100);               /* 256 block transfers */
set_blk(0x0L);                /* starting block */
while ((tv = tmrvalue()) < (unsigned)( 3*60)) {  /* 3 mins */
    dmaset_vblk("W");         /* set the virtual starting addr */
    readr_blk();              /* perform read */
    new_start = inc_blk(0x100);  /* new starting block */
    if (new_start + 0x100 > last_block_num) {  /* if starting
                                 block is greater than last
                                 block number, */
        set_blk(0x01);        /* start over on drive */
    }
}
tmrstop();                    /* end of three minute loop */
sprintf(dummy,"Timer Value = 0x%04X",tv);  /* display timer */
rpttmr();                     /* report timer to log */
rptstats(1);                  /* report statistics */

paragph("Time Reads (3 mins) with Random Starting Addresses");
                              /* -DOC
                              ; -PT="Time Reads w/Random Addrs"
                              ;
                              ; Utilize random_blk() to read
                              ; randomly over entire disk (in
                              ; a 3-minute timed loop)
                              ;
                              ; -DOC */
stats_reset("ALL");           /* reset statistics */
tmrset(0x0);                  /* set timer to start at 0 */
tmrstart("Up");               /* start timer counting up */

set_len(0x100);               /* 256-block transfers */
set_blk(0x0L);                /* starting block */
while (tmrvalue() < (unsigned)( 3*60)) {  /* 3 min count */
    dmaset_vblk("W");         /* set the virtual starting addr */
    readr_blk();              /* perform read */
                              /* calculate random block */
    random_blk(0L,(last_block_num - (unsigned long)0xFF));
}
tmrstop();                    /* end of three minute loop */
rptstats(1);                  /* report statistics */


paragph("Timed Loop (10 minutes) With All Random");
                              /* -DOC
                              ; -PT="Timed Loop with All Random"
                              ;
```

```
                                  ; Randomly select the type of
                                  ; operation:
                                  ;      6-byte read,
                                  ;      6-byte write,
                                  ;      10-byte read,
                                  ;   or 10-byte write
                                  ; Likewise randomly select the
                                  ; starting block and transfer
                                  ; length, executing all in a 10
                                  ; minute timed loop
                                  ; -DOC */
    stats_reset("ALL");           /* reset statistics */
    rptstats(1);                  /* report statistics */

    for (i = 0; i < 6; i++) {     /* one-hour test */
        tmrset(0x0);              /* set timer to start at 0 */
        tmrstart("Up");           /* start timer counting up */

        ioto(1200);               /* set long for long random acks */
        while (tmrvalue() < (10*60)) {  /* count for ten minutes */
                                  /* calc trans len & start addr */
            len = random_len(1,0x1000);  /* transfer len limit */
            block =
                random_blk(01,last_block_num-(unsigned long)len+1);
            dmaset_vblk("W");     /* set the virtual starting addr */
            akd = rand();         /* get random ack delay */
            ackdelay(0x0FF & akd);  /* set fixed delay */
            op_type = 0x0003 & rand();  /* use C library random
                                     number to choose type of
                                     operation */
            if (op_type == 0) {
               readr_blk();       /* six byte read command */
            }
            else if  (op_type == 1) {
               writer_blk();      /* six byte write command */
            }
            else if (op_type == 2) {
              readr10_blk();      /* 10 byte read command */
            }
            else {
               writer10_blk();    /* 10-byte write command */
           }
        }
        tmrstop();                 /* end of 10 minute timed loop */
        rptstats(0);               /* report statistics no header */
    }
}
```

~SAT.3 SAT DEBUG

The following sections described how the SDS-1 Debugger relates to the SAT. Refer to the Debugger Section for more detailed description of the SDS-1 Debugger.

~SAT.3.1 COMMAND TAIL OPERATOR -DB=

After successful compilation and linkage of the SAT, its executable file can now be executed using the SDS-1 Debugger. There are four different levels in the Debugger. The execution speed and debug modes vary with each level, with Level Ø being the fastest to execute but with less information displayed on the screen, to Levels 2 and 3 being the slowest with more information shown. When enabled, the frames in the Status Fixed Window will be updated (the more screen updates, the slower the execution). Usually when debugging the SAT program, Level 2 or 3 is used, since these levels provide the most screen information and updates to aid in debugging.

To execute the SAT program example in Debug Level 3, enter the SAT file name with the specified debug level:

C>**SATNAME -DB=3**

where **SATNAME** is the SAT executable file name. The **-DB=** command tail operator specifies the debug level. Command tail operators are options that can be defined on the command line. If the **-DB=** operator does not exist, the default is debug level zero. A screen should appear similar to Figure SAT-F9. As described in the DEBUG section, there are several modes: **TRACE, IOINIT, IMP ER, EXP ER** and **IOABRT.** The current mode is determined by looking at the bottom left corner of the screen. When users first enter the Debugger, the **TRACE** state is usually the first mode encountered. This mode is where users will be spending most of their SAT debug time. There is another mode called **IOINIT** which appears when using the **H**alf-Step command, but only if the half-stepped function is an I/O Driver command. The other modes are error condition modes: **IMP ER** (Implicit Errors), **EXP ER** (Explicit Errors) and **IOABRT** (I/O Driver Abort).

There are several command options for each mode, to display them press the **space bar** to show the different menu lines. The commands on these menu lines may be entered at any menu line as long as the mode supports them. Return to the first **TRACE** menu line by pressing the **space bar** until:

TRACE : Flow >**G**oto; **B**reak Pt.(Ø); **R**un; **S**tep; **H**alf Step; S**k**ip; DOS Ret;

Press the **S** key several times to step through your SAT program; notice that the S is highlighted in Step on the menu line. The Step command will advance to the next function and display it on the Trace Display Window. When the function name and its parameters are pending execution, it appears in reverse video in the Trace Display Window. After the function has been executed, the trace function name and its parameters appear in full-intensity. If a function has been skipped (the K command), the function name will appear in half-intensity.

We have looked at Step and Skip commands. If the user wants to execute the rest of the program without Debugger intervention, use the **R**un command.

Another feature of the Debugger is buffer displays. Data, SCSI Bus State Log and Sense buffers can be displayed. The SCSI Bus State Log Buffer Display can assist in problem identification and the Data Buffer Display can identify data integrity errors.

~SAT.3.1.1 DEBUG LEVEL Ø

Debug Level Ø has no statistic updates, but provides the fastest execution of the four levels of debug. There are only two windows: Test Documentation Fixed Window and Test Documentation Scrolling Window (see figure below). There are two ways to invoke this level:

C>**SATNAME -DB=Ø**

or

C>**SATNAME**

The default level is Ø. Usually, this level is used after all bugs have been fixed in the SAT and execution without interference of the debugging modes is desired.

FIGURE ~SAT-F6. DEBUG LEVEL Ø

```
---------------- ADAPTEC Test Structure Library (11-30-84) ----------------
                        DOS Command Line Execution
                          01-08-86   11:45:17
                                              Printer Output Disabled:
  1.0 On Board Buffer Write/Read/Compare Testing       01-08-86 11:45:20
     1.2 Read and Compare (via DMAHC) OBB Write Data   01-08-86 11:51:46
        1.2.9 Pseudo Random DMAHC Read                 01-08-86 12:00:33

------------------------------ REPORT DISPLAY ------------------------------
1.2.6 00 FF 55 AA DMAHC Read                      01-08-86 12:00:15

1.2.7 Incrementing Pattern DMAHC Read             01-08-86 12:00:17

1.2.8 Decrementing Pattern DMAHC Write            01-08-86 12:00:21
IOABORT IMPLICIT ERROR                            01-08-86 12:00:22
Cmp Error: Ref Buf(0x0000 = 0x04); SCSI Data = 0x22;

IOABORT IMPLICIT ERROR                            01-08-86 12:00:32
I/O Time-Out (Time-Out Value = 10 seconds)


1.2.9 Pseudo Random DMAHC Read                    01-08-86 12:00:33
```

~SAT.3.1.2 DEBUG LEVEL 1

In addition to the windows provided in Level 0, the next level includes the Status Fixed Window (see figure below). This level provides information about the SAT program in progress. When enabled, the frames in this window will be updated while the SAT is executing. There are two ways to enable/disable this window:

(1) through library functions in the SAT (**bcu()** and **statsen()**), or
(2) through the Debugger command, BCU (the statistics frame cannot be enabled through the Debugger).

To invoke this level, use the same **-DB=** operator:

C>**SATNAME -DB=1**

The following levels can be called in this manner with the specified level.

FIGURE ~SAT-F7. DEBUG LEVEL 1

```
------------------ ADAPTEC Test Structure Library (11-30-84) -------------------
                         DOS Command Line Execution
                            01-08-86    11:45:17
                                                   Printer Output Disabled:
  1.0 On Board Buffer Write/Read/Compare Testing        01-08-86 11:45:20
     1.1 OBB Fill Testing                               01-08-86 11:45:20
        1.1.6 00 FF 55 AA  OBB Write                    01-08-86 11:45:49
-------------------------------- I/O DRIVER STATUS -----------------------------
 I/O  Ops:       5 | uc0:            |  I/O Command Parameters   stat: 00 __
 TGT Chks:       0 | uc1:            | CDB: 0a 00 01 00 40 00    sense: (old)
 INT D Er:       0 |-----------------|      00 00 00 00 00 00    __ __ __ __
 Bytes Wr:   28000 | Wr/Ref: OBB     | xfer: HSRW     a.s.:OFF   __ __ __ __
 Bytes Rd:       0 |           0000  | s.l.ON  arb.HDW sel.SMA   __ __ __ __
 Bytes Cp:       0 | Rd Buf: OBB     | b.p.OFF b.w.OFF           __ __ __ __
 Cmp Ers.:       0 |           0000  | ha: 0  iid: 7  tid: 4     __ __ __ __
--------------------------------- REPORT DISPLAY -------------------------------
1.1.4 Constant 55 Pattern OBB Write                 01-08-86 11:45:42

1.1.5 1233210 Pattern OBB Write                     01-08-86 11:45:46

1.1.6 00 FF 55 AA  OBB Write                        01-08-86 11:45:50

TRACE:  <ESC> Halt :
```

~SAT.3.1.3 DEBUG LEVEL 2

Debug Level 2 has the following windows: Test Documentation Fixed Window, Status Fixed Window and the Trace Display Scrolling Window.

The function and its arguments are displayed in the Trace Display Scrolling Window which provides a step-by-step execution history of the SAT program (see figure below). Only functions from the Test and Documentation Library can be traced.

FIGURE ~SAT-F8. DEBUG LEVEL 2

```
--------------- ADAPTEC Test Structure Library (11-30-84) ---------------
                        DOS Command Line Execution
                           01-08-86    11:45:17
                                                  Printer Output Disabled:
  1.0 On Board Buffer Write/Read/Compare Testing        01-08-86 11:45:20
     1.1 OBB Fill Testing                               01-08-86 11:45:20
        1.1.11 Word Block Count OBB Write               01-08-86 11:48:34

------------------------------- I/O DRIVER STATUS -------------------------
 I/O  Ops:         E | uc0:           |  I/O Command Parameters   stat: 00 __
 TGT Chks:         0 | uc1:           | CDB: 0a 00 09 00 40 00    sense: (old)
 INT D Er:         0 |----------------|      00 00 00 00 00 00    __ __ __ __
 Bytes Wr:     68000 | Wr/Ref: OBB    | xfer: HSRW      a.s.:OFF  __ __ __ __
 Bytes Rd:         0 |          0000  | s.l.ON  arb.HDW sel.SMA   __ __ __ __
 Bytes Cp:         0 | Rd Buf: OBB    | b.p.OFF b.w.OFF           __ __ __ __
 Cmp Ers.:         0 |          0000  | ha: 0   iid: 7   tid: 4   __ __ __ __
--------------------------------- TRACE DISPLAY ---------------------------
filli(7e,0000,4000) writer(0300,0) paragph() ackdelay(243)
filld(04,0000,4000) writer(0400,10) writer(0410,1f) writer(042f,11)
paragph() ackdelay(154) fillpr(008a,0000,4000) writer(0e00,10)
writer(0e10,1f) writer(0e2f,11) paragph() ackdelay(6020)
fillbcb(90,0100,0000,4000) writer(0900,40) paragph() ackdelay(2100)
fillbcw(0940,0100,0000,4000)
TRACE : Control )Debug Level(2); BCU(1); User Cntr Reset; Stats Reset;
```

~SAT.3.1.4 DEBUG LEVEL 3

This level has only two windows: Status Fixed Window and the Trace Display Scrolling Window; as shown in the figure below.

FIGURE ~SAT-F9. DEBUG LEVEL 3

```
-------------------------------- I/O DRIVER STATUS -------------------------------------
| I/O  Ops:        2F | uc0:              | I/O Command Parameters       stat: 00 __  |
| TGT Chks:         0 | uc1:              | CDB: 08 00 00 c0 40 00       sense: (old) |
| INT D Er:         0 |-------------------|      00 00 00 00 00 00       00 00 00 00  |
| Bytes Wr:    F0400  | Wr/Ref: BPM       | xfer: DMAHC     a.s.:OFF     __ __ __ __  |
| Bytes Rd:    50000  |           0000    | s.l.ON  arb.HDW sel.SMA      __ __ __ __  |
| Bytes Cp:    20000  | Rd Buf:           | b.p.OFF b.w.OFF              __ __ __ __  |
| Cmp Ers.:         0 |                   | ha: 0   iid: 7    tid: 4     __ __ __ __  |
---------------------------------- TRACE DISPLAY ---------------------------------------
writer(0580,40) overbcw(05c0,0100,0000,4000) writer(05c0,40)
overbcw(0600,0100,0000,4000) writer(0600,40) overbcw(0640,0100,0000,4000)
writer(0640,40) overbcw(0680,0100,0000,4000) writer(0680,40)
overbcw(06c0,0100,0000,4000) writer(06c0,40) paragph() ackdelay(2100)
fillpr(009f,0000,0200) savebuf(OBBIMG.TST,0000,0200) writer(0a00,2)
paragph() dmarst(R) ackdelay(0) readr(0000,0040) paragph() dmarst(R)
ackdelay(15) readr(0040,0040) paragph() dmarst(R) ackdelay(255)
readr(0080,0040) paragph() dmarst(R) readr(00C0,0040) paragph() dmarst(R)
readr(0300,001F) readr(031F,0020) readr(033F,0001) paragph() ackdelay(0)
dmarst(R) readr(0900,0001) readr(0901,0010) readr(0911,000F)
readr(0920,0020) group() xfermode(DMAHC,4000) paragph() fillk(00,0000,4000)
readr(0000,0040) paragph() fillk(F,0000,4000) readr(0040,0040) paragph()
fillk(AA,0000,4000) readr(0080,0040) paragph() fillk(5,0000,4000)
readr(00C0,0040) paragph()
TRACE : Flow )Goto; Break Pt.(0); Run; Step; Half Step; Skip; DOS Ret;
```

~SAT.3.2 COMMAND TAIL OPERATOR -PR

Another command tail operator that can be used is the **-PR** operator which will send the Test Documentation Scrolling Window to the printer. This operator may appear anywhere on the command line after the file name.

If the **-PR** operator is not performing as it should, be sure to delete all temporary (**.TMP**) files before using this operator. These temporary files were left over from an aborted batch file execution. To delete all temporary files, enter the following:

C>**ERASE *.TMP**

~SAT.4 LIBRARY CATALOGING

SATs can begin to accumulate rapidly. To keep track of each SAT, a system of cataloging the SATs is provided. It consists of a binder with log pages and a place to put diskette copies of user's SATs. Cataloging provides revision control and history via report generator operator (-REV). It is also the central point of SAT cataloging and SAT backup.

~SAT.5 ERROR HANDLING LOGIC

The goal of the SDS-1 System is a hands-off regression test which provides a pass or fail result. Under these conditions, the user does not analyze any data to make the pass/fail decision, all decisions are made in the regression test itself.

The SDS-1 System supports two types of error detection. The first type is implicit error detection. An implicit error is an illegal condition detected by the Test Function Library that the user does not have to test for explicitly. The most common example of an implicit error is a data compare error between the write/reference buffer and the read buffer. The data miscompare is an implied error in the data compare mode and the user does not need to explicitly check for the error.

The second type is explicit error. An explicit error is an error generated by an explicit test. For example, a check for extended sense key = 6 (unit attention) is an explicit test and a sense key other than 6 will result in an explicit error.

The action taken by the SDS-1 System when an implicit or an explicit error is detected is established by the Test Library Functions **iea()** and **eea()** (implicit error action and explicit error action).

User options for each type of error action for the SAT mode are:

(CONT) Ignore Error and Continue

(HALT) Stop SAT and Invoke the Debugger ERROR PROCESSOR
(no error logging)

(LOGC) Log Error and Continue
(Up to user-defined **set_er_limits()**, default is 100 errors; otherwise, invoke the Debugger ERROR PROCESSOR)

(LOGH) Log Error and Invoke the Debugger ERROR PROCESSOR.

These error actions can also be modified when the user is in the Debugger and the Debug Level is greater than 0, by using the **IEA** and **EEA** debug menu commands. When errors are detected in the default mode, LOGC, an error message is shown and execution of SAT continues. If this mode was modified to HALT or LOGH, the Debugger will halt execution on error so that one could examine the error condition in the **IMP ER** or **EXP ER** debug mode.

The meaning of IEA and EEA value changes when running in the batch mode environment, such as in the Design Verification batch file, refer to Table DEBUG-T1 for those definitions.

~SAT.6 SAT EXECUTION HALT/INTERRUPTION

In addition to setting the error action iea and eea functions (or **I**EA or **E**EA commands) to halt on error, there are other ways stop or interrupt SAT execution.

~SAT.6.1 NORMAL END OF SAT PROGRAM

To exit from the Debugger at any level, the completion of the SAT program will return back to DOS.

~SAT.6.2 ESCAPE KEY

If the Debug Level is greater than 0, the **ESC** key can be used to stop execution of the SAT program and the user can regain control in the **TRACE** state with the next function pending execution (indicated in reverse video).

~SAT.6.3 CONTROL-BREAK KEYS

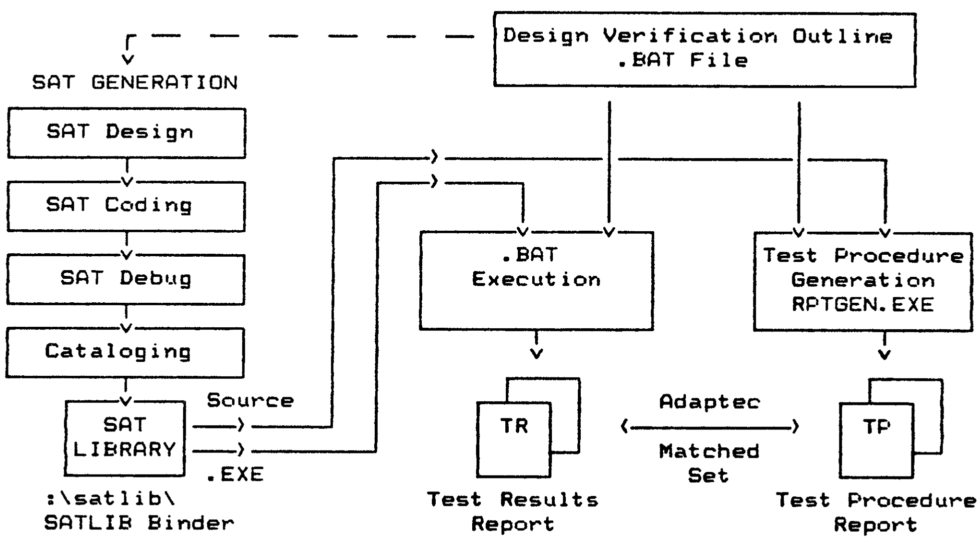
The **CTNL-BREAK** (**^Brk**) key sequence can be used to interrupt execution of SAT program, at which point the user has the option to display the SCSI bus, exit to DOS or resume execution. The **CNTL-BREAK** sequence will exit the user from a SAT execution with the exception of PC crashes.

~DV.Ø DESIGN VERIFICATION PROCESS

~DV.1 INTRODUCTION

After debugging the SATs, the next step in the SDS-1 Development Process is to generate the Test Results and Test Procedure Reports. This is the Design Verification process (see Figure DV-F1).

FIGURE ~DV-F1. DESIGN VERIFICATION PROCESS



~DV.2 DESIGN VERIFICATION RESULTS

The Adaptec Matched Sets: Test Results and Test Procedure Reports are the final products of the Design Verification Test Sequence, see Figure DV-F1. The Test Results Report is generated while executing the SAT in the Design Verification Batch File (described in the next section). The Test Procedure Report is the documentation or code report of the test procedure using the same execution batch file.

~DV.2.1 TEST RESULTS DOCUMENTATION

The Test Results documentation is generated by executing the Design Verification Batch File. Shown below are the contents that make up the Test Results Report:

TITLE PAGE
TABLE OF CONTENTS
SAT #1 RESULTS
.
.
.
SAT #N RESULTS
APPENDIX A: BATCH FILE COPY
APPENDIX B: TEST DATA SUMMARY

The Design Verification batch file executes the SATs sequentially and provides the "hands-off" test execution. The following **BLANKDV.BAT** file can be used as a template.

FIGURE ~DV-F2. BLANK DESIGN VERIFICATION FILE (BLANKDV.BAT)

```
ECHO OFF
TITLEPG %0 -TI="Design Verification Title" -CD=07-15-85 -RN=RN# -FO=%0.TR
          REM
          REM
          REM

                          REM Stand Alone Test Selection
                          REM Abort Regression Test if BLANKSAT1 fails
BLANKSAT1 -TN=
    IF ERRORLEVEL 1 GOTO BAD

BLANKSAT2 -TN=
BLANKSAT3 -TN=

    ENDTS -M1=" Pass Messages Here "  -M2="Same as M1"
GOTO END

:BAD
    ENDTS -M1=" Failure Message Here"  -M2="Same as M1"

:END

    ERASE *.TMP
    ECHO ON
```

The **ECHO OFF** and **ECHO ON** are DOS batch commands to turn off and on the screen display of the command lines in the batch file.

The **TITLEPG** command line prints the title page of the Test Results documentation. Its operators are:

- **-TI** - Title of Test Results Documentation
- **-CD** - Creation Date
- **-RN** - Reference Number or Name
- **-FO** - File Name Output

The **%0** that appears on this command line is the batch file name with the **.BAT** file extension removed. According to the above **TITLEPG** command line, if the batch file name is **DVFILE.BAT**, the test results would be located in a file named **DVFILE.TR** .

Lines that contain **REM** are the remark or comment lines which are ignored during batch execution.

The BLANKSAT1, BLANKSAT2 and BLANKSAT3 are the SAT programs to be executed. The **-TN=** operator is the test section number assigned to the SAT for documentation purposes. If **-TN=** is not assigned, the next sequential number will be used as its test section number.

If an error occurs during execution of a SAT, the ERRORLEVEL value is nonzero. The user can check the ERRORLEVEL for good SAT completion as shown in Figure DV-F2.

The **ENDTS** command line prints out a message in Appendix B of the Test Results report known as the Test Data Summary Section. **ENDTS** can define up to four 80-character messages, but they all must appear on one command line.

The **:BAD** and **:END** are labels used by the **GOTO** batch command. The label consists of a colon followed by a label name. The **GOTO** command causes execution to transfer to the next command following the label.

This batch file can be created to produce the Test Results documentation using Sidekick's Notepad and using **BLANKDV.BAT** as a template. Batch file names should always have an extension of **.BAT** . The file name of the batch file is all that is needed to execute this file. Suppose the batch file name is **BATNAME.BAT**, then to execute it, enter:

C>**BATNAME**

While executing, the screen will show the execution sequence of this batch file. After it has completed (when the DOS prompt appears), the Test Results File can be viewed or printed out to a printer:

C>**PRINT BATNAME.TR**

See Section B.5.3 for an example of the Test Results Report.

~DV.2.2 TEST PROCEDURE DOCUMENTATION

The Test Procedure Report is the documented procedure of the Test Results. This document is formatted by the report generator input file operators in the SAT code. The title page and creation date is retrieved from the batch file's **TITLEPG** command. Shown below are the typical contents that make up the Test Procedure Report:

TITLE PAGE
TABLE OF CONTENTS
SAT #1 TEST PROCEDURE
.
.
.
SAT #N TEST PROCEDURE
APPENDIX A: BATCH FILE COPY
APPENDIX B: SAT REVISION HISTORY

There are options that will include/exclude the Revision History and/or a Code Listing Title Page in Appendix B and/or C (refer to Section RPTG.3.2 for setting up the operators that control the Test Procedure Appendix).

To generate the Test Procedure documentation, the **RPTGEN** program is used. The following is a batch file called **TP.BAT** that will write the Test Procedure documentation into an input file:

FIGURE ~DV-F3. TEST PROCEDURE BATCH FILE (TP.BAT)

```
ECHO OFF
     REM Generate Test Procedure to current drive for input file
     REM Assume IBM Graphics Printer
RPTGEN %1.BAT -MD=TP -RL -RN=SDS-1TP-01 -PW=8 -FN=%1.TP
```

The above operators are:

-MD - RPTGEN mode: TP - Test Procedure Generation
CD - Code Documentation
Default mode is TP
-RL - Document Revision Log in Appendix
-RN - Reference Number or Name
-PW - Page Width Switch and Printer Control
8: 8.5" paper and IBM (Epson) Control Codes
8A: 8.5" paper and ANADEX Rapid Scribe Codes
13: 13.4" paper and no control codes
-FN - File name of Output (if -FN is not specified, report will go directly to the printer)

Refer to Section RPTG.2.3 for more detailed information on these operators.

If the above batch file does not exist, you may create it. To execute, enter **TP** and the batch file name without the .BAT extension:

C>**TP BATNAME**

This batch file uses the batch file name as input from the command line.

To print the test procedure file to printer:

C>**PRINT BATNAME.TP**

There is an option to send this document to the printer, instead of sending it to a disk file, by not using the **-FN** operator, since the output default is to the printer.

See Section B.5.2 for an example of the Test Procedure Report.

(THIS PAGE INTENTIONALLY LEFT BLANK)

~RPTG.Ø REPORT GENERATOR

~RPTG.1 INTRODUCTION

RPTGEN is a program designed to perform one of the most necessary but dreaded engineering tasks, documentation. Its primary purpose is to generate a Test Procedure Report from a Design Verification Batch File. In addition to this function, **RPTGEN** also provides a convenient means of generating a Test Results report. These two reports are referred to as the Adaptec Matched Documentation Sets.

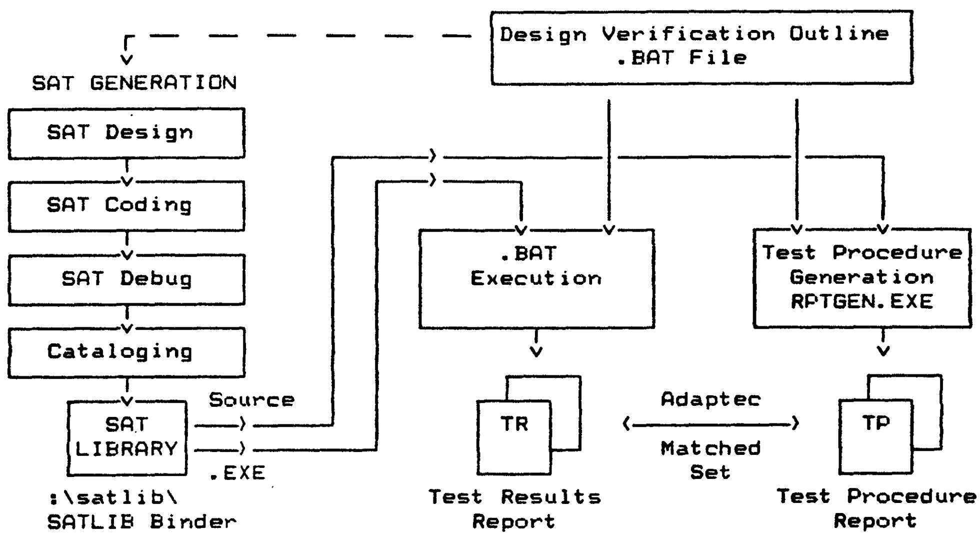
The Test Procedure report consists of the documented procedure and/or code that was used to run the test(s). In addition, a copy of the batch file is also included in Appendix A of the Test Procedure report. There is also an option for a Revision Log report and a Code Listing Title Page for Appendix B and/or C.

The Test Results report is the documented case of the execution of the batch file. Also included with this document is the Execution Batch File (Appendix A) and a Test Data Summary Report (Appendix B). Though the Test Results document does not use **RPTGEN**, there is a relationship that exists between these two documents.

~RPTG.1.1 ARCHITECTURE

Figure RPTG-F1 shows the basic "documentation" architecture in which **RPTGEN** operates. The program was designed around a batch (.BAT) file and a group of related "library" files, such as SATs. These files can be a group of individual SCSI test files or a group of program modules which are compiled (or assembled) and linked together to generate a specific program. **RPTGEN** serves as the "Documentation Linker" in combining these individual modules (files) into a single well-structured document.

FIGURE~RPTG-F1. REPORT GENERATOR (DESIGN VERIFICATION PROCESS)



~RPTG.1.2 BASIC OPERATION

RPTG.1.2.1 TEST RESULTS REPORT

After the SAT programs have been debugged, they should be ready to run in a batch file environment. A batch file example is shown in Figure RPTG-F2. If the batch file name is TEST.BAT, then all that is needed to execute this file is to enter its file name:

C>**TEST**

While execution of the batch file is in progress, the Test Results report is being generated, producing the documented execution results. Based on the **group()** and **paragph()** functions contained in the SAT, a Table of Contents (TOC) will also be generated, refer to Section FLIB.5 for other Test Results documentation functions. When batch execution has been completed, a Test Data Summary section is generated. The title page is initialized by the **TITLEPG** command line and any messages can be defined by the **ENDTS** command.

TABLE ~RPTG-T1. RPTGEN EXECUTION ERROR MESSAGES

```
--- Command Tail Error --- No Batch File Specified

--- Command Tail Error --- Hyphen not found

--- File I/O Error --- Cannot Open Batch File

--- File I/O Error --- Cannot Create Output File

--- File I/O Error --- Cannot Write to Output File

--- File I/O Error --- Cannot Open Temporary TOC File

--- File I/O Error --- Cannot Open Temporary Revision Log File

--- File I/O Error --- Cannot Write Temporary TOC File

--- File I/O Error --- Cannot Rewind Temporary TOC File

--- File I/O Error --- Cannot Rewind Temporary Rev Log File

--- File I/O Error --- Cannot Write to Temporary Rev Log File
```

RPTG.1.2.2 TEST PROCEDURES REPORT

To generate the Test Procedures document file, **RPTGEN** is executed (notice that **RPTGEN** was not involved in the Test Results report generation). **RPTGEN** receives the name of the input batch file via its command tail, for example:

RPTGEN test.bat -MD=TP -RL -RN=TP-test-01 -PW=8 -FN=test.TP

The same batch file that was used to generate the Test Results report must be used to produce the Test Procedure report. **RPTGEN** begins by looking for the title (**-TI=**) and create date (**-CD=**) operator within this batch file in the **TITLEPG** command line. This information is used to print the document cover or title page, refer to Figure RPTG-F2 for a batch file example.

FIGURE ~RPTG-F2. BATCH FILE EXAMPLE

```
TITLEPG test -TI="DEMO REPORT" -CD=09-16-85 -RN=TR-01 -FO=test.TR
test1
REM -FN=test1.c -TN=1
test2
REM -FN=test2.c
test3 -TN=
ENDTS -M1="End of Demo"
```

RPTGEN next places a copy of the input batch file in Appendix A of the Test Procedures document file. The main documentation function begins at this point. **RPTGEN** begins a line-by-line scan of the batch file looking for File Name operators (**-FN=**). This operator specifies the file name to be used for the output document. If the Test Procedure Implied Mode is used, there is no need for this operator (see Section RPTG.2.2.2.2).

When a file name is found, **RPTGEN** opens the file and processes the input file in a line-by-line manner. During this process, **RPTGEN** is looking for the input file operators. These operators define documentation lines (**-DOC**), code lines (**-COD**), revision log lines (**-REV**), group and paragraph titles (**-GT=** and **-PT=**) and **RPTGEN** control functions (**-DB, -.PA, -AI, ...**). Each input file is completely scanned for these operators. A source file may look like Figure RPTG-F3.

After scanning the input file, **RPTGEN** returns to the control or batch file for the next operation. After the batch file has been completely scanned, the document body created, and requested appendices have been generated; the last step of **RPTGEN** is to generate a Table of Contents. Refer to Appendix B.5 for a **RPTGEN** example.

TABLE ~RPTG-T2. REPORT GENERATOR OPERATORS

INPUT FILE OPERATORS			BATCH FILE OPERATORS			RPTGEN OPERATORS	
	-DB=	(TP)		-TI=	(TP/TR)	-FN=	(TP)
global mode	-DOC	(TP)	TITLEPG ops	-CD=	(TP/TR)	-WS=	(TP)
	-COD	(TP)		-RN=	(TR)	-MD=	(TP)
				-FO=	(TR)	-RL=	(TP)
						-RN=	(TP)
-DOC mode	-REV	(TP)				-CP=	(TP)
	-GT=	(TP)	batch cmd	-FN= -TN=	(TP)	-PW=	(TP)
	-PT=	(TP)				-TE=	(TP)
	-.PA	(TP)					
	-AI=	(TP)		-M1=	(TR)		
			ENDTS ops	-M2=	(TR)		
				-M3=	(TR)		
-COD mode	-.PA	(TP)		-M4=	(TR)		
-REV mode	-.PA	(TP)					

TP = Test Procedures Report
TR = Test Results Report

FIGURE ~RPTG-F3. SOURCE FILE WITH INPUT FILE OPERATORS EXAMPLE

```
/* -DOC
Filename:  test1.c

This is an example of
using the source file operators.

-REV
         Created:  09/16/85
Initial Release:
        Revision:
-REV
-GT="Example of SAT"
-DOC */

/* -COD */
user_test()
{
test("Example of SAT";
group("Write/Read/Compare in DMA HC Transfer Mode");
xfermode("DMAHC",0x40);          /* DMAHC transfer mode
                                 with 64K buffer size */
ioto(10);                        /* 10 second timeout */
arbmode("HDW");                  /* hardware arbitration */
tid(0);                          /* target ID is 0 */
lun(0);                          /* logical unit # 0 */
/* -COD */

/* -DOC
-PT="Write in DMA HC Transfer Mode"
-DOC */

/* -COD */
paragph("Write in DMA HC Transfer Mode");
dmarst("W");                     /* reset DMA Write Buffer */
filli(0,0,0x40);                 /* fill buffer with
                                 incrementing pattern */
writer(0,0x40);                  /* write 64k bytes */
/* -COD */

/* -DOC
-PT="Read/Compare in DMA HC Transfer Mode"
-DOC */

/* -COD */
paragph("Read/Compare in DMA HC Transfer Mode");
readr(0,0x40);                   /* read 64k bytes */
}
/* -COD */
```

~RPTG.2 REPORT GENERATOR OPERATORS

RPTGEN makes decisions and obtains input information for a set of operators. These operators occur in three mutually-exclusive areas: input files, batch (control) file, and command tail. Each set of operators serves a specific function in the final report generation (refer to Table RPTG-T2).

~RPTG.2.1 INPUT FILE OPERATORS (TEST PROCEDURE REPORT)

During input file processing, **RPTGEN** examines each source file line in a sequential manner. It looks for file operators which establish **RPTGEN**'s "line processing mode." There are four basic line modes:

GLOBAL	(not in **-DOC**, **-COD** or **-REV** mode)
DOC	processing Document lines
REV	processing Revision Log lines (contained within Document line mode)
COD	processing Code lines.

NOTE: **RPTGEN** will find only the first operator on a line, then proceed to the next line (with the exception of the **-PT=** and **-RN=** pair).

RPTG.2.1.1 GLOBAL OPERATORS

Global operators can occur in any portion of the input file. They provide control information and cause line mode changes.

The **-DOC** and **-COD** operator pairs can be intermixed. **RPTGEN** will provide continuous print (filling up a page and continuing on the next page) until a **-GT=** operator is found between **-DOC** operators. The document mode **-GT=** operator causes a page eject and places the group title and reference number at the top of the next page.

RPTG.2.1.1.1 DOCUMENTATION BOUNDARY (-DB=)

-DB=c DEFAULT: no left-hand margin character

Establishes the character, c, to be used as the left-hand margin for the documentation and revision log portions of report generation process. A typical boundary character is the ";" which is the comment character for most assemblers. **RPTGEN** will ignore all characters and spaces preceding the boundary character for **-DOC** and **-REV** lines. The **-DB=** operator will disable the left-hand margin character function.

RPTG.2.1.1.2 START/STOP DOCUMENT OUTPUT OPERATOR (-DOC)

-DOC

Once a **-DOC** operator is encountered in GLOBAL mode, **RPTGEN** will remain in document line mode until the next **-DOC** operator (toggle function) is found, all other global operators and nondocument operators will be ignored.

> **NOTE:** The documentation line mode will truncate any text past column 66.

RPTG.2.1.1.3 START/STOP CODE OUTPUT OPERATOR (-COD)

-COD

When the **-COD** operator is encountered in GLOBAL mode, **RPTGEN** will remain in code line mode until the next **-COD** operator (toggle function) is found, and all other global operators and noncode operators will be ignored.

In code line mode, if the **-CP** (code print) operator is found in the command tail, **RPTGEN** adds a line number to the input file line and outputs the line to the printer. If 8.5-inch paper width is specified, the output lines are printed using compressed print. All other file operators are ignored when **RPTGEN** is in code output mode.

RPTG.2.1.2 DOCUMENTATION LINE MODE OPERATORS

Documentation operators are valid only within the limits established by the **-DOC** operator pair.

RPTG.2.1.2.1 START/STOP REVISION LOG OUTPUT (-REV)

-REV

If the **-RL** command tail operator appears on the **RPTGEN** command line, **RPTGEN** will enter the Revision Line Mode and output the document lines between the **-REV** operator pairs to a temporary file RPTGENRL.TMP which will be attached to the main document as Appendix B. **RPTGEN** will supply a title line and reference number from the current test, group, or paragraph, depending on where the **-REV** operator pair was embedded in the document area.

RPTG.2.1.2.2 GROUP TITLE OPERATOR (-GT=)

-GT=xxxxxxxxxxxxxxx	(single-word title)
-GT="xxxx xxxxx xxxx**"**	(multiple-word title)

xxx...xxx is the group title which will be used in the Table of Contents and at the top of the group. **RPTGEN** will automatically generate a group number with the following format:

#.x where:

is the string taken from the batch file operator **-RN=** or assigned by **RPTGEN** when no **-RN=** operator is found.

x is the next group number. At the start of a new input file, **RPTGEN** sets its group reference counter, x, to Ø. When a **-GT=** operator is encountered, the group reference counter is incremented and used to define the group. Each time the **-GT=** operator is encountered, the paragraph reference counter is reset to Ø.

The group title operator will cause a TOC entry and a page eject prior to printing the group title. The page eject will be held if the **-GT=** operator occurs within the first 26 lines of a new test (section).

RPTG.2.1.2.3 PARAGRAPH TITLE OPERATOR (-PT=)

-PT=xxxxxxxxxxxxxxxxx (**-RN=**sss) (single-word title)
-PT="xxxxx xxxxx xxxx" (**-RN=**sss) (multiple-word title)

This is the paragraph title operator with an optional reference number extension. The xxx...xxx is the paragraph title which will be used in the Table of Contents and at the top of the paragraph.

NOTE: The ( ) are NOT part of syntax.

RPTGEN will automatically generate a paragraph reference number with the following format:

#.x.y.sss where:

is the string taken from the batch file **-RN=** operator.

x is the current group number.

y is the paragraph number. If the **-RN=** operator is found on the same line as the **-PT=** operator, **RPTGEN** assumes that the user wishes to expand the numbering system beyond the three-deep level supported by **RPTGEN**. Therefore, the paragraph reference counter, y, will not be incremented. If only the **-PT=** operator is found, then y will be incremented and used. y is reset at each occurrence of the **-GT=** operator.

sss is the paragraph extension supplied via the **-RN=** operator.

RPTG.2.1.2.4 PAGE EJECT OPERATOR (-.PA)

-.PA

The **-.PA** operator will cause **RPTGEN** to generate a top of form. This is useful when a description is longer than a single page and the user wishes to control the page break location.

> **NOTE:** The **-.PA** operator will be ignored if a natural page break has just occurred and the printer is at the top of a new page.

RPTG.2.1.2.5 ART INSERT OPERATOR (-AI=)

-AI=xxxxxxxx.yyy
-AI="xxxxxxxx.yyy"

RPTGEN will allow the insertion of "printer image" files which are formatted for the IBM PC Graphics printer. This allows the user to include PC PAINT PLUS artwork into the Test Procedure document, but only if the document is sent directly to the printer. **RPTGEN** assumes all artwork will be 33 lines by 8Ø columns (1Ø characters/inch). **RPTGEN** will also ensure the current page contains enough room for the art insertion or a page eject is performed.

Since graphics art insertion requires output to an IBM Graphics printer, report output to the Anadex printer or to a file (CNTL-Z problem) cannot contain a printer image. For these cases, **RPTGEN** will leave a blank area of 33 lines with the art file name centered in this area. This allows the user to paste the artwork after the document has been completed.

RPTG.2.1.2.5.1 MOUSE HARDWARE SETUP

The mouse is a small pointing device with 3 buttons. It is used to move the pointer or indicator on the screen, to select tools, to draw, and to pull down menus on the screen display. In this section, the click or clicking is done with the left button. There are three parts to the mouse: mouse, mouse pad and power supply. To connect the mouse to the SDS-1, do the following:

a. plug one end of the power supply into the RS-232C connector jack (from the mouse) and the other end into a wall outlet
b. plug the RS-232C connector (from the mouse) into the COM port of the SDS-1
c. place the mouse on top of the mouse pad.

Refer to the Mouse Systems PC PAINT PLUS reference manual for more information on the mouse and its usage.

RPTG.2.1.2.5.2 MOUSE SOFTWARE SETUP

On SDS-1 boot, the mouse driver, MSMOUSE, should have already been executed. To use the mouse software, the user should change his current directory (C:\USER1) to the C:\PAINT and execute PCPAINT:

C>**CD \PAINT**
C>**PCPAINT**

The first screen to appear will indicate that PC PAINT PLUS is running. Then the screen will change to show the user's work area with a pointer. The pointer indicates where the mouse is and the current mode or option. The initial option is the pencil. When the pointer leaves the work area, the pencil changes to an arrow. Along the top of the screen are the PC PAINT PLUS menus. Along the left side and bottom of the screen are the tool and option boxes (notice that the pencil box is highlighted, since that is the current mode). Move the mouse over the mouse pad; notice that the pointer or indicator on the screen also moves in the same direction. Now move the pointer to the Mouse System logo (located at the top left corner of the screen) and click the mouse's left button--a command list should appear. Move the pointer down to the "Control Box"; when it is highlighted, click the mouse. In this command, the user can modify the current values for running PC PAINT PLUS. Some of the control box values are initially set to:

a. Display mode: 320 X 200 4-color
b. Sensitivity: Medium
c. Pic size: 8 X 11 - Low & Portrait X-240 Y-275.

Other control box values are discussed in the Mouse Systems PC PAINT PLUS reference manual. To modify the values set, move the mouse to the appropriate box and click the mouse. Some boxes have more than one option; in this case, continue to click the mouse until all possible options are shown. Other option or value types are entered via keyboard. To accept the new values, click the mouse at the Accept box. To cancel the new values and return PC PAINT PLUS to the way it was before the control box was opened, click the mouse when pointer is at the Cancel box.

RPTG.2.1.2.5.3 MOUSE DRAWING OR PAINTING

To draw figures, use the tools and/or options available by moving the mouse to the tools and options box and clicking the tool and/or option to use. Then move the mouse to the site where the drawing is to start and, depending on the tools picked, either hold down the left button and move the mouse to draw or click the mouse to paint.

RPTG.2.1.2.5.4 SAVING THE PICTURE

Before saving the picture, the filename must be specified. Move the pointer to the File menu and click. Then move pointer down to the Save command and click. The Save screen should appear.

To modify the directory path name, move the mouse pointer to the top field in the Save screen. This is the directory path box. Click the mouse and type in the new directory path name and press the **RETURN** key.

To change the filename, move the mouse pointer to the Filename box (below the directory path box) and click the mouse. Enter the name of the figure or picture and press the **RETURN** key. If saving a picture, an extension of **.PIC** will be added to the file name or if saving a clipping, **.CLP** will be added.

To save the picture, be sure the Picture box is highlighted and move the pointer to the Save box. Then hold the **CTRL** key and click the mouse. If the replace option is requested, move the pointer to the Replace box and hold the **CTRL** key again and click the mouse. This will save the picture in a format that is compatible with the SDS-1 Report Generator.

RPTG.2.1.2.5.5 EXIT PC PAINT PLUS AND RETURN

To exit from PC PAINT PLUS, move the pointer to the File menu and click. Then move the pointer down to the Quit PC PAINT command and click. This should return the user back to DOS. Then to return back to the user directory (C:\USER1):

```
C>CD \USER1
```

RPTG.2.1.2.5.6 USING THE ART INSERT OPERATOR

For art insertion into the Test Procedure Report, use the **-AI=** operator along with its file name. If the file is not in the current user directory, specify the full path name (up to 20 characters may be used). An example of the **-AI=** operator:

```
/* -DOC
     .
     .
     .
-AI=C:\PICTURES\ART.PIC
     .
     .
     .
-DOC */
```

RPTG.2.1.3 CODE LINE MODE OPERATORS

Code operators are valid only within the limits established by the **-COD** operator pairs (one starting and one ending the code area).

RPTG.2.1.3.1 PAGE EJECT OPERATOR (-.PA)

-.PA

The **-.PA** operator will cause **RPTGEN** to generate a top of form. This is useful when a code area containing sections that could be easily understood by starting at the top of a page.

> **NOTE:** The **-.PA** operator will be ignored if a natural page break has just occurred and the printer is at the top of a new page.

RPTG.2.1.4 REVISION LOG LINE MODE OPERATORS

Revision operators are valid only within the limits established by the **-REV** operator pair (one starting and one ending the Revision Log section).

RPTG.2.1.4.1 PAGE EJECT OPERATOR (-.PA)

-.PA

The **-.PA** operator will cause **RPTGEN** to generate a top of form in the Revision Log Appendix (Appendix B of documentation).

RPTG.2.2 BATCH FILE OPERATORS

RPTG.2.2.1 INITIAL SETUP

The **TITLEPG** command will initialize the title page, reference number and file name output for the Test Results Report. Below is a typical example of the **TITLEPG** command line:

TITLEPGdtest -TI="DEMO TEST" -CD=09-16-85 -RN=TR-08 -FO=dtest.TR

RPTGEN creates the title page by scanning this command line for the title and creating date operators for the Test Procedure Report.

The batch file name follows **TITLEPG** on the command line; the **TITLEPG** operators are listed. Its operators are defined in the following sections:

RPTG.2.2.1.1 DOCUMENTATION TITLE AND HEADER (-TI=)

-TI="xxx...xxx**"**

The document title and header for the Test Results and Test Procedures reports where xxx...xxx is the title or header specified.

RPTG.2.2.1.2 CREATION DATE (-CD=)

-CD=mm-dd-yy

This operator defines the creation date of the batch file which is printed on the cover or title page of the Test Results and Test Procedure reports, where mm-dd-yy is the month, day and year.

RPTG.2.2.1.3 REFERENCE NUMBER OR NAME (-RN=)

-RN=xxx.yyy-ØØ1

The reference number or name of the Test Results report which appears on the cover or title page. The definition consist of a maximum of 35 alphanumeric characters and/or symbols in a "free format" manner.

RPTG.2.2.1.4 FILENAME OUTPUT (-FO=)

-FO=xxxxxxxx.yyy

This operator specifies the name of the file where the Test Results report is to be saved on disk where xxxxxxxx is the file name and yyy is the file extension.

RPTG.2.2.2 SPECIFY FILE NAME (TEST PROCEDURES REPORT)

There are two ways to specify the file name for the Test Procedures Report: the **-FN=** operator and the Test Procedure Implied Mode.

RPTG.2.2.2.1 FILE NAME OPERATOR (-FN=)

This is the file name operator with an optional test (section) number operator. To specify the file name for the Test Procedure report, use the **-FN=** operator:

REM **-FN=**xxxxxxxx.yyy (**-TN=**(nnn))

NOTE: The ( ) are NOT part of the syntax.

Usually located in the REM or comment line, the **-FN=** operator specifies an input source file that **RPTGEN** uses in creating the Test Procedure report. **RPTGEN** keeps an internal test (section) reference counter which is set to Ø at program initialization. Each **-FN=** occurrence increments this counter. **RPTGEN** will use this counter for the test (section) number in the Test Procedure report's Table of Contents and in group and paragraph numbering if a **-TN=** operator is not found on the same line. Otherwise, **RPTGEN** will use "nnn" as the test (section) number. The test (section) reference counter will be incremented with or without the presence of the **-TN=** operator. The **-FN=** operator will generate a TOC entry and cause a page eject in the output report; a new test (section) will always start at the top of a page.

RPTG.2.2.2.2 IMPLIED MODE

When in Test Procedure mode (**-MD=TP**), and the Batch File line contains a **-TN=** operator but no **-FN=** operator, **RPTGEN** assumes that the first command on the line is an .EXE file which was generated from a file with the same name and an extension of .C (for "C" source) or .A (for assembly code). **RPTGEN** will search for these files as input files.

> **NOTE:** The **-TN=** operator can be used without a test reference number. RPTGEN will assign the next sequential test reference number to this test.

RPTG.2.2.3 MESSAGES (TEST RESULTS REPORT)

To produce messages in the Test Results report, use the **ENDTS** command line. ENDTS prints out the message(s) in Appendix B, Test Data Summary Section of the Test Results report. The **ENDTS** program can define up to 4 messages at one time but they all must appear on the same command line. Each message may contain up to 80 characters.

ENDTS -M1="Successful Execution" -M2="No Errors Detected"

RPTG.2.3 COMMAND TAIL OPERATORS (TEST PROCEDURES REPORT)

The command tail operators appear after the batch file name on the **RPTGEN** command line:

RPTGEN test.bat -MD=CD -RL -RN=test-02 -PW=8

Command tail operators are used so **RPTGEN** itself can be utilized from the DOS batch environment.

RPTG.2.3.1 OUTPUT FILE SWITCH (-FN=)

-FN=xxxxxxxx.yyy DEFAULT: output to printer

Redirects **RPTGEN** output from the printer to a disk file specified after the operator, where xxxxxxxx is the file name and yyy is the file extension. The file will contain the printer image.

RPTG.2.3.2 WORDSTAR FILE OUTPUT (-WS=)

-WS=xxxxxxxx.yyy DEFAULT: output to printer

Redirects **RPTGEN** output from the printer to a disk file specified after the operator, where xxxxxxxx is the file name and yyy is the file extension. This file will not be a printer image but will contain WordStar "dot" commands. These dot commands generate a WordStar output identical to the document produced if the output is sent directly to the printer. This mode allows the user to modify the document in WordStar before printing.

WARNING: Table of Contents page numbers are generated by **RPTGEN.** Addition or deletion of document pages will change the TOC.

RPTG.2.3.3 RPTGEN MODE (-MD=)

-MD=TP or **-MD=CD** DEFAULT: Test Procedure Generation

Sets the Test Procedure Generation (TP) or Code Documentation (CD) mode. The only function of this flag is to modify the **RPTGEN** run time status window and the output document title page.

RPTG.2.3.4 REVISION LOG SWITCH (-RL)

-RL DEFAULT: No revision log

Document the lines between **-REV** operator pair in Appendix B of the Test Procedures Report, otherwise these lines are ignored.

RPTG.2.3.5 FILE REFERENCE NUMBER OR NAME (-RN=)

-RN=xxx.yyy-001 DEFAULT: Blank Reference Number or Name

This reference number or name appears on Document Cover Sheet or Title Page of the Test Procedures Report. The definition consists of a maximum of 35 alphanumeric characters and/or symbols in a "free format" expression.

RPTG.2.3.6 CODE PRINT SWITCH (-CP=)

-CP DEFAULT: No Code Print

Includes all code lines in output document. If **-PW=8**, then the code segments will be printed in compressed print.

RPTG.2.3.7 PAGE WIDTH SWITCH AND PRINTER CONTROL (-PW=)

-PW=8
-PW=13
-PW=8A DEFAULT: 8.5 inch paper IBM Printer

The following are the codes for the **-PW=** operator:

8 = 8.5 inch paper and IBM (Epson) Control Codes
8A = 8.5 inch paper and ANADEX Rapid Scribe Control Code
13 = 13.4 inch paper and no control codes sent.

It is used with **-CP** (code print) switch to determine if printer output will be compressed during code line printing.

RPTG.2.3.8 TAB EXPANSION OPERATOR (-TE=)

-TE=n DEFAULT : -TE=5

Defines batch (control) and input file Tab expansion stations; for example, if n = 4, then tabs are set at 5, 9, 13, 17, etc.

~RPTG.3 OUTPUT REPORT FORMAT

~RPTG.3.1 TEST RESULTS REPORT

There are many options which affect the generation of the Test Results report. Section FLIB.5 describes the report generator function that provides these options. The basic Test Results report structure is:

REPORT ELEMENT	COMMENTS
TITLE PAGE	Contains: TITLE from **TITLEPG** in Batch File CREATE DATE from **TITLEPG** in Batch File BATCH FILE NAME " " LAST REVISION DATE " " LAST REVISION TIME CURRENT DATE & TIME
TABLE OF CONTENTS	Generated from **test**(), **group**(), **paragph**() and **subpar**() functions
SAT #1 RESULTS : SAT #N RESULTS	Contains: The execution listing generated from the report generator functions
APPENDIX A	Batch File listing
APPENDIX B	Test Data Summary Report and any **ENDTS** message(s).

~RPTG.3.2 TEST PROCEDURES REPORT

The **RPTGEN** command tail provides the user with a number of options which affect the Test Procedures report appendix structure. However, the body structure of the final report is consistent. This structure is as follows:

REPORT ELEMENT	COMMENTS
TITLE PAGE	Contains: TITLE from **TITLEPG** in Batch File CREATE DATE from **TITLEPG** in Batch File BATCH FILE NAME " " LAST REVISION DATE " " LAST REVISION TIME CURRENT DATE & TIME
TABLE OF CONTENTS	Generated from Batch File **-FN=** operators and input file **-GT=** and **-PT=** operators
SAT #1 PROCEDURE : SAT #N PROCEDURE	Contains: **-DOC** and **-COD** lines from input file along with titles generated by Batch File **-FN=** operator and input file **-GT=** and **-PT=** operators
APPENDIX A	Batch File listing

Depending on the values of the **-MD=** operator and whether the **-RL** operator exists, the following is a table of the appendix definition for Appendix B and C of the Test Procedure report:

APPENDIX B	APPENDIX C	COMMAND TAIL FLAGS
NONE	NONE	-MD=TP (Test Procedure Generation)
REVISION LOG	NONE	-MD=TP -RL (Test Procedure Generation with Revision Log)
CODE LISTING TITLE PAGE	NONE	-MD=CD (Code Documentation Generation)
REVISION LOG	CODE LISTING TITLE PAGE	-MD=CD -RL (Code Documentation Generation with Revision Log)

For examples of the Test Results and Test Procedure Report, see Appendix B.5.

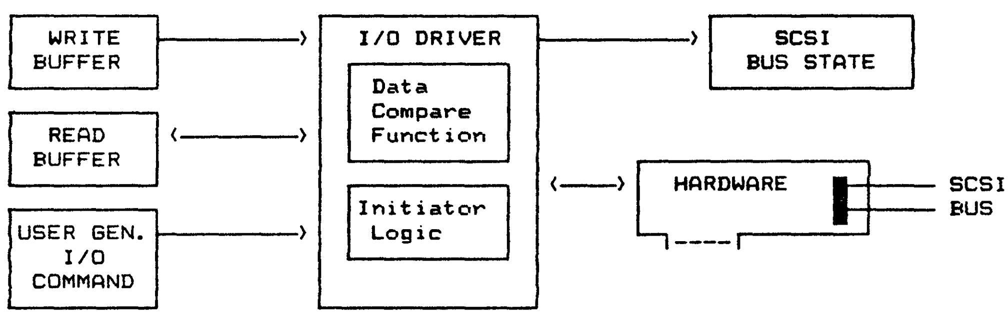
(THIS PAGE INTENTIONALLY LEFT BLANK)

~IODVR.Ø I/O DRIVER

~IODVR.1 EXECUTION ENVIRONMENT

The I/O Driver is the SDS-1's primary SCSI execution environment. It is used to execute the SCSI random and sequential functions such as **writer()** and **writes()**. These I/O Driver functions provide the user with an easy means of executing SCSI commands, with the task of SCSI bus management being performed by the I/O Driver. Figure IODVR-F1 shows the basic execution I/O Driver environment. Features and characteristics of the I/O Driver are discussed in following sections.

FIGURE ~IODVR-F1. I/O DRIVER EXECUTION ENVIRONMENT



~IODVR.2 BUFFER MANAGEMENT

An important task of the I/O Driver is memory buffer management. The SDS-1 utilizes a three-buffer architecture (see IODVR-F2). All data is written from the write buffer. Data is read from the SCSI bus into the read buffer. The third and final buffer, sense buffer, is a special case read buffer used only for SCSI sense commands (refer to section IODVR.6 for definition of its use).

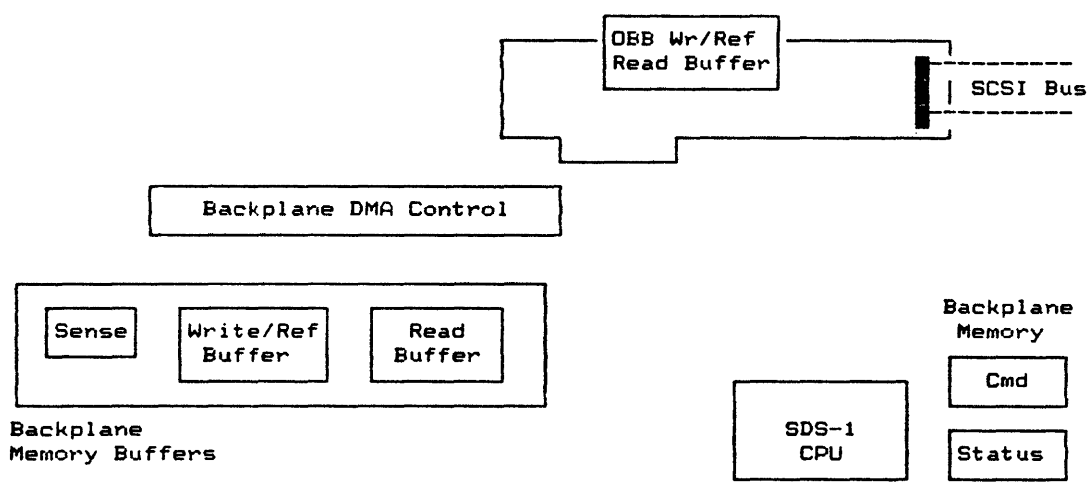
The SDS-1 utilizes two classes of buffers. Under certain conditions (see **xfermode()** in Appendix A), system main memory is used as the write and read buffers. Other modes utilize the special High-Speed On-Board Buffer located on the SDS-1 SCSI interface or test adapter board.

~IODVR.2.1 BUFFER WRAPAROUND

The I/O Driver performs buffer wraparound. In other words, an SCSI transfer that exceeds the physical buffer size will make multiple passes through the buffer. For SCSI write operations, the data pattern appearing on SCSI will repeat every buffer size. For read operations this means that after the first buffer size transfer, data will be overwritten in the SDS-1 read buffer.

When using backplane DMA transfer modes, the I/O Driver software must manage buffer wraparound (via software intervention) each time the buffer size limit is reached. The SDS-1 High-Speed On-Board Buffer (OBB) utilizes hardware wraparound, and as such, only requires software intervention every 16MB of transfer (limit of OBB transfer length counter).

FIGURE ~IODVR-F2. SDS-1 BUFFER ARCHITECTURE



~IODVR.2.2 DATA COMPARISON

A second function of the SDS-1 I/O Driver is data comparison and compare error reporting. The action taken by the I/O Driver and SDS-1 Debugger on a data compare error depends upon the implicit error (**iea()**) selected by the user and the execution environment (design verification batch file or SAT/MENU). Table IODVR-T1 defines data compare error processing. The user should also refer to the DEBUG section for further understanding of the **IOABRT** state and compare error handling.

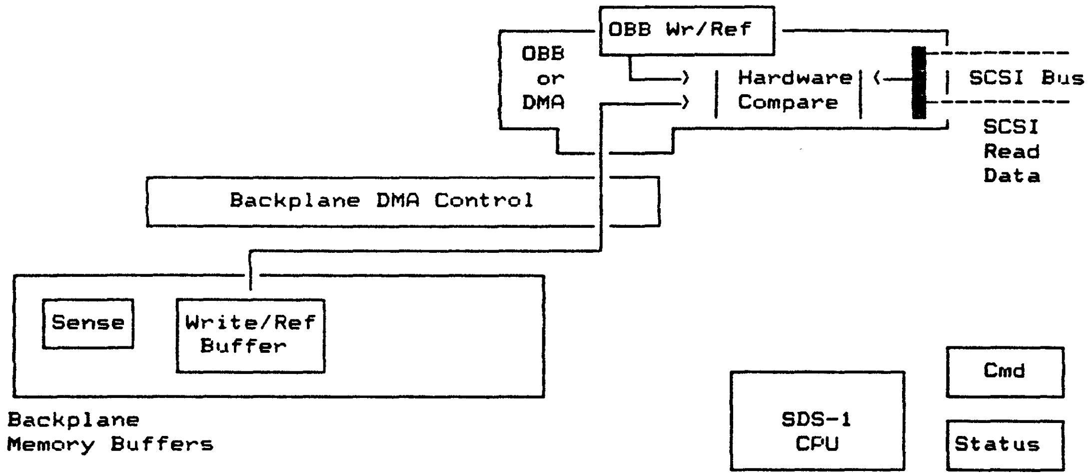
TABLE ~IODVR-T1. DATA COMPARE IMPLICIT ERROR ACTION

IMPLICIT ERROR ACTION	SAT/MENU RESPONSE (NON-BATCH MODE)	DESIGN VERIFICATION RESPONSE (BATCH MODE)
CONT (CONTINUE)	Accumulate function statistics but does not report counts or miscompare counts.	Accumulate function statistics but does not report counts or miscompare counts.
HALT (HALT)	Enter **IOABRT** state with expected and actual data displayed.	Complete I/O after compare error and return to DOS and execute the next SAT.
LOGH (LOG & HALT)	Report each compare error in log until HOE set to Ø in **IOABRT.** Accumulate function statistics and report to log the overall execution statistics at completion of I/O. Halt processing in Debugger's ERROR PROCESSOR state.	Report first compare error in log. Accumulate function statistics, report the overall execution statistics at completion of I/O. Return to DOS and execute the next SAT.
LOGC (LOG & CONTINUE)	Report each compare error in log until HOE set to Ø in **IOABRT.** Accumulate function statistics and report to log the overall execution statistics at completion of I/O. Continue execution until error limit is reached. If error limit reached, stop in Debugger's ERROR PROCESSOR state.	Report first compare error in log. Accumulate function statistics. Report the overall execution statistics at completion of I/O. Continue execution until error limit is reached. If error limit reached, return to DOS.

~IODVR.2.2.1 HARDWARE DATA COMPARE

The SDS-1 SCSI interface hardware contains a special hardware comparator which compares SCSI data in an "On-the-Fly" mode. In other words, as the data is read in from the SCSI bus it is compared against a reference buffer. There is no read buffer and the read SCSI data is not saved after the compare is completed (see Figure IODVR-F3). If a data compare error occurs, the SDS-1 freezes the SCSI REQ/ACK handshake and displays the expected data from the reference buffer and the SCSI read data.

FIGURE ~IODVR-F3. HARDWARE COMPARE ARCHITECTURE



Since data is compared "On-the-Fly," the comparison appears from a timing standpoint to look like a single read command. Buffer wraparound is managed as it would be in a simple write or read condition. The user should remember that there is no read buffer in hardware compare modes and all SCSI commands which result in a DATA IN phase (with the exception of **sense()**) will be compared against the write/ref buffer.

~IODVR.2.2.2 SOFTWARE DATA COMPARE

Software data compare is handled in one of two ways. For PIOSC (Programmed I/O Software Compare) and TRSC (Transmit/Receive Software Compare) transfer modes, each byte is compared (by the system CPU) against the write/reference buffer as it is read from the SCSI bus. This is possible because the CPU handles each and every byte of the DATA IN phase.

DMASC (DMA Software Compare) and HSSC (High-Speed Software Compare) provide a "real-time" transfer environment with an "after the transfer" data comparison by the host CPU. In other words, data is transferred into the read buffer via a DMA process and once the buffer is full, the CPU compares the write/reference buffer with read buffer. This feature allows the user to view not only the data compare error itself (as was the case in hardware compare), but also the data around the compare error. In fact the RW option in the buffer display command (**dispbuf()**) will show the read buffer and write/reference buffer side-by-side. During the software compare process, the read buffer is filled and the write/ref buffer DMA pointer is used as the reference data pointer for the software compare.

FIGURE ~IODVR-F4. SOFTWARE COMPARE OPERATION EXAMPLE

Initial Conditions

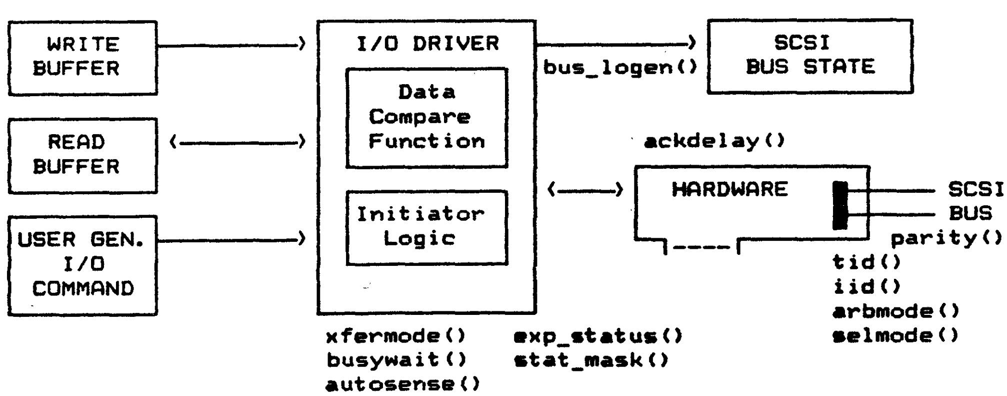
Buffer Size = 0x8000 (32K buffer)
SCSI read command will transfer -> 0xC000 bytes (48K)
Initial write/ref pointer 0x2000
Initial read pointer 0x2000

I/O Driver Operation	DMA Pointer Values Write/Ref	Read
	0x2000	0x2000
1. I/O Driver reads 0x6000 bytes up to buffer end		
	0x2000	0x0000
2. I/O Driver compares up to buffer end		
	0x0000	0x0000
3. I/O Driver completes SCSI command (0x6000 bytes)		
	0x0000	0x6000
4. I/O Driver completes compare (0x6000 bytes remaining) (0x4000 bytes left)		
	0x6000	0x6000

~IODVR.3 CONTROL FUNCTIONS

Figure IODVR-F5 shows the I/O Driver Execution Environment with the various I/O Driver control functions. These functions allow the user to simulate many different SCSI host environments.

FIGURE ~IODVR-F5. I/O DRIVER CONTROL FUNCTIONS



~IODVR.3.1 I/O TIME OUT

The **ioto()** function provides a "watch dog" timer on any I/O Driver operation. The action taken by the I/O Driver/Debugger combination is a function of implicit error action (**iea()**) selected by the user and the execution environment. Table IODVR-T2 defines this logic.

TABLE ~IODVR-T2. TIME OUT IMPLICIT ERROR ACTION

IMPLICIT ERROR ACTION	SAT/MENU RESPONSE (NON-BATCH MODE)	DESIGN VERIFICATION RESPONSE (BATCH MODE)
CONT (CONTINUE)	Abort I/O and continue with the next SAT function.	Abort I/O and continue with the next SAT function.
HALT (HALT)	Enter **IOABRT** state and and allow the user to terminate or continue I/O with secondary time-out.	Abort I/O and return to DOS and execute the next SAT.
LOGH (LOG & HALT)	Enter **IOABRT** state and allow the user to terminate or continue I/O with secondary time-out. If user terminates I/O, log as I/O time-out and halt processing in Debugger's ERROR PROCESSOR.	Abort I/O and log time-out. Return to DOS and execute next SAT.
LOGC (LOG & CONTINUE)	Abort I/O and log error. Continue execution until error limit is reached. If error limit reached, stop in Debugger's ERROR PROCESSOR state.	Abort I/O and log error. Continue execution until error limit is reached. If error limit reached, return to DOS and execute the next SAT.

NOTE: When I/O is aborted as a result of a time-out, a bus reset is performed.

~IODVR.3.2 PARITY

SCSI bus parity, both generation and checking, is controlled by the **parity()** function. The I/O Driver responds to a DATA IN parity error by asserting attention and internally setting a MESSAGE OUT of DATA PARITY ERROR. If the target requests a MESSAGE OUT in response to attention assertion, this message is sent. In addition, the Initiator Status returned by the I/O Driver will report a parity error detection. The I/O Driver handling of parity error is intentionally limited. The SDS-1 microprogramming environment is designed to provide the user with a controlled means of error generation and response checking.

~IODVR.3.3 ARBITRATION

Three modes of arbitration are supported by **arbmode()** function.

NONE: No arbitration, selection will jump on bus as with nonarbitrating SCSI devices.

HARDWARE: During hardware arbitration, the arbitration win decision is processed by hardware with no software intervention required. The hardware will continue to arbitrate after losses until it finally wins.

SOFTWARE: During software arbitration, the arbitration win decision is processed by software. (If another device asserts select, hardware will take over and remove busy from the bus.) If arbitration is lost and state logging is enabled, the loss is recorded in the state log.

~IODVR.3.4 SELECTION

The **selmode()** function provides two options: SMART and DUMB.

With SMART selection, attention is asserted during selection and an identify message (with disconnects supported) will be sent to the target. DUMB selection does not assert attention and as such will never allow disconnects.

The synergistic effects of **selmode()** and **arbmode()** are described below:

arbmode()	**selmode()**	NUMBER OF SELECT BITS	ID MESSAGE
NONE	DUMB	1	NO
NONE	SMART	2	NO
SFTW or HDW	DUMB	1	NO
SFTW or HDW	SMART	2	YES

~IODVR.3.5 SCSI PATH CONTROL

The SCSI bus path is established with the **iid()**, **tid()** and **lun()** functions. Using the **iid()** function, the user can simulate multiple hosts talking to the same SCSI target.

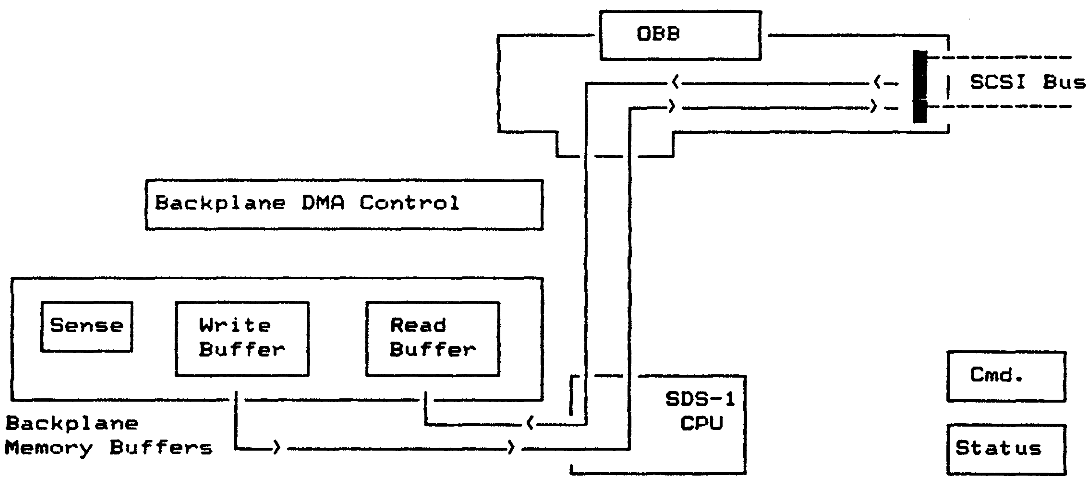
~IODVR.3.6 TRANSFER MODES

One of the SCS-1's major features is its ability to emulate various SCSI hosts. The data transfer portion of this emulation is controlled by the **xfermode()** function. This function allows the user to select one of 13 different data transfer/compare modes for the I/O Driver. The **xfermode()** function description in Appendix A summarizes these modes, while the following sections define each mode in detail.

IODVR.3.6.1 PIO READ/WRITE (PIORW)

Each data byte is transferred by the SDS-1 CPU using Programmed I/O acknowledge handshake. This is the slowest means of transfer.

FIGURE ~IODVR-F6. PIORW TRANSFER MODE BLOCK DIAGRAM

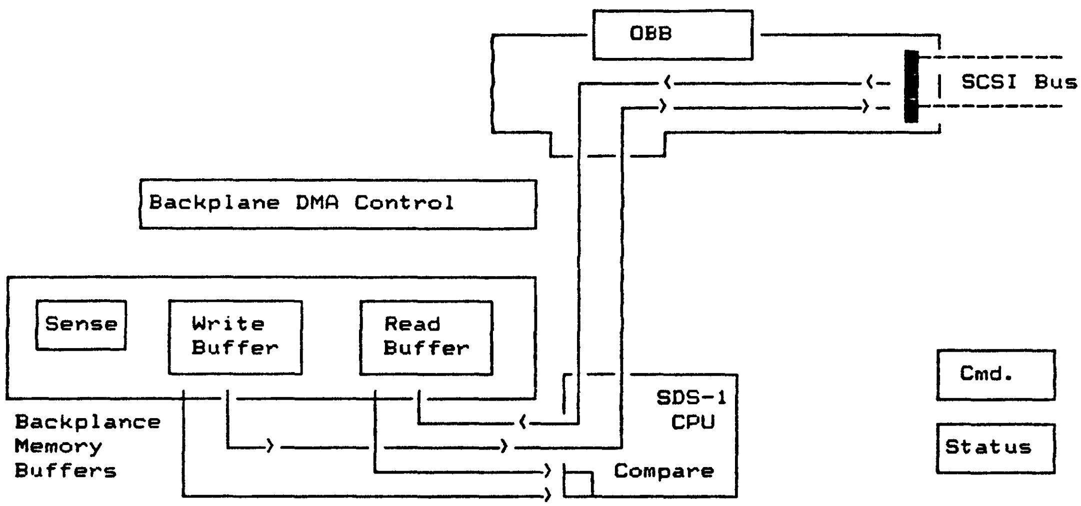


Backplane PIO Read/Write	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	CPU Data Move CPU Ack Cntl.	N.A.
Read Operation	BPM	16-> 32,768	CPU Data Move CPU Ack Cntl.	N.A.
Comments:				

IODVR.3.6.2 PIO SOFTWARE COMPARE (PIOSC)

Each data byte is transferred by the SDS-1 CPU using Programmed I/O acknowledge handshake. During read operations, each DATA IN byte is compared against the write/ref buffer.

FIGURE ~IODVR-F7. PIOSC TRANSFER MODE BLOCK DIAGRAM

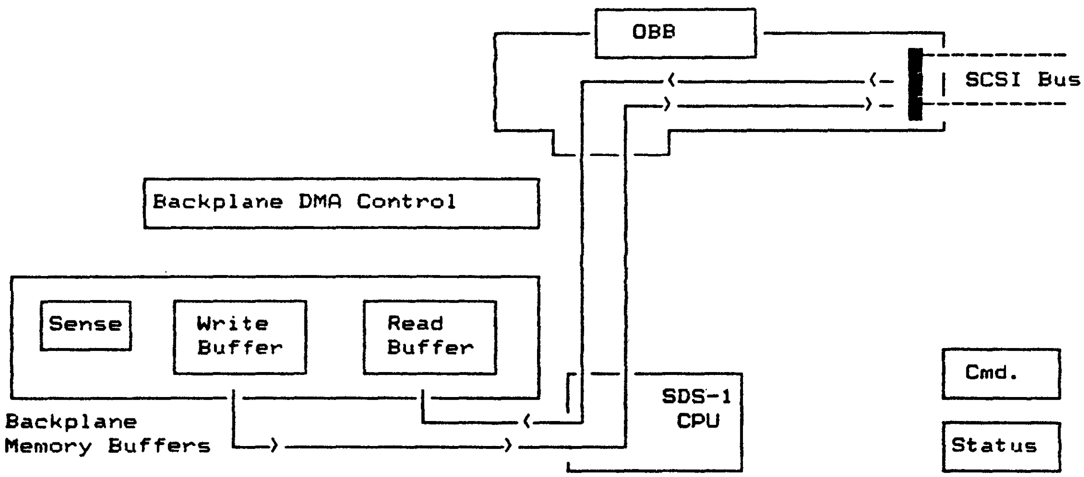


Backplane PIO Software Compare	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	CPU Data Move CPU Ack Cntl.	N.A.
Read Operation	BPM	16-> 32,768	CPU Data Move CPU Ack Cntl.	CPU Ref & Read Buffer Compare
Comments: I/O Driver will always try to fill read buffer then perform compare, until last read which may be less than a buffer in length.				

IODVR.3.6.3 TR READ/WRITE (TRRW)

Each data byte is transferred by the SDS-1 CPU using a special hardware acknowledge logic (the ACK signal is generated automatically on information transfer).

FIGURE ¯IODVR-F8. TRRW TRANSFER MODE BLOCK DIAGRAM

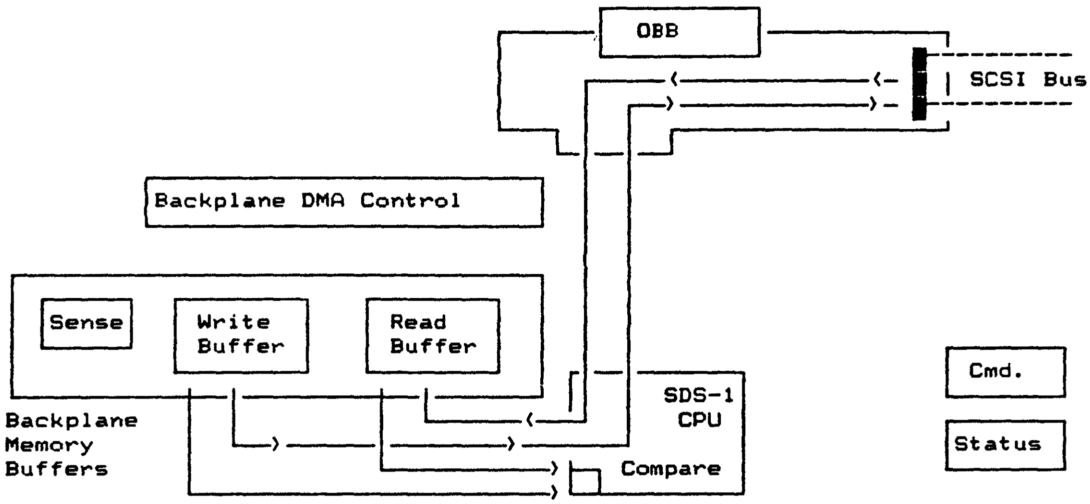


Backplane TR Read/Write	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	CPU Data Move Hardware Ack	N.A.
Read Operation	BPM	16-> 32,768	CPU Data Move Hardware Ack	N.A.
Comments:				

IODVR.3.6.4 TR SOFTWARE COMPARE (TRSC)

Each data byte is transferred by the SDS-1 CPU using a special hardware acknowledge logic. During read operations each DATA IN byte is compared against the write/ref buffer.

FIGURE ~IODVR-F9. TRSC TRANSFER MODE BLOCK DIAGRAM



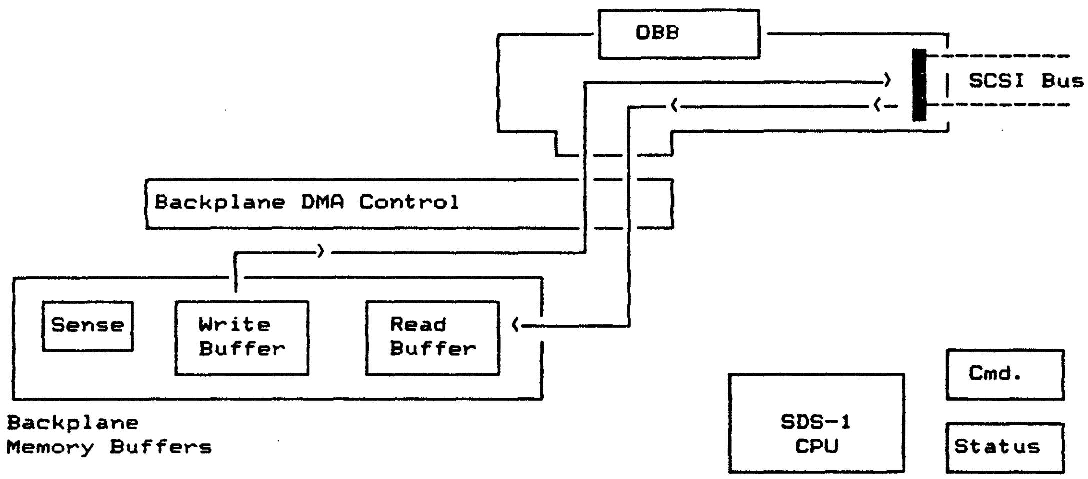
Backplane TR Software Compare	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	CPU Data Move Hardware Ack	N.A.
Read Operation	BPM	16-> 32,768	CPU Data Move Hardware Ack	CPU Ref & Read Buffer Compare

Comments: I/O Driver will always try to fill read buffer then perform compare, until last read which may be less than a buffer in length.

IODVR.3.6.5 DMA READ/WRITE (DMARW)

DMARW utilizes the backplane memory buffers and the SDS-1 host DMA controller to transfer write and read data. All handshaking is handled via the DMA logic.

FIGURE ~IODVR-F10. DMARW TRANSFER MODE BLOCK DIAGRAM

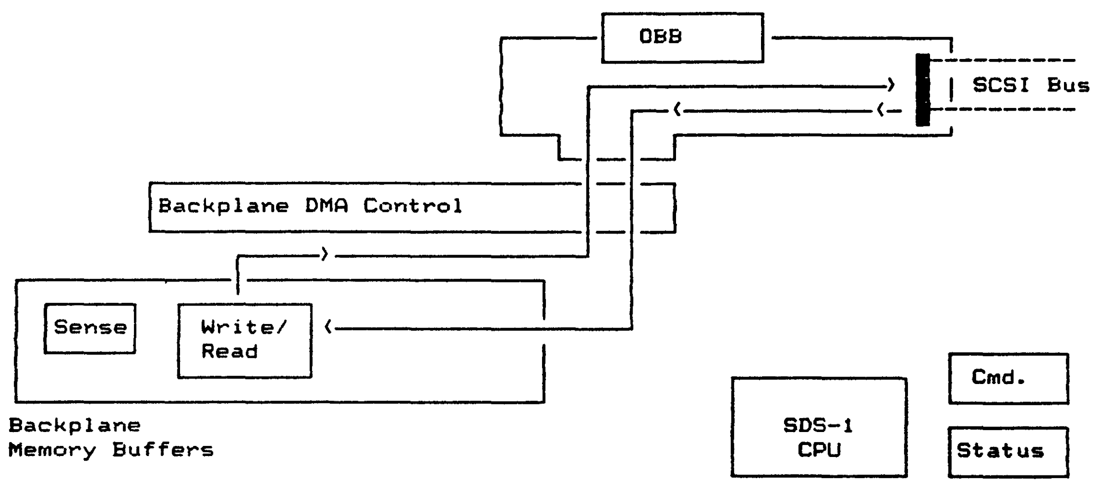


Backplane DMA Read/Write	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	Backplane DMA Controller	N.A.
Read Operation	BPM	16-> 32,768	Backplane DMA Controller	N.A.
Comments:				

IODVR.3.6.6 DMA COPY (DMACOPY)

DMACOPY is similar to DMARW with the difference that the write and read buffer are the same physical buffer. This is useful for peripheral-to-peripheral transfer.

FIGURE ~IODVR-F11. DMACOPY TRANSFER MODE BLOCK DIAGRAM

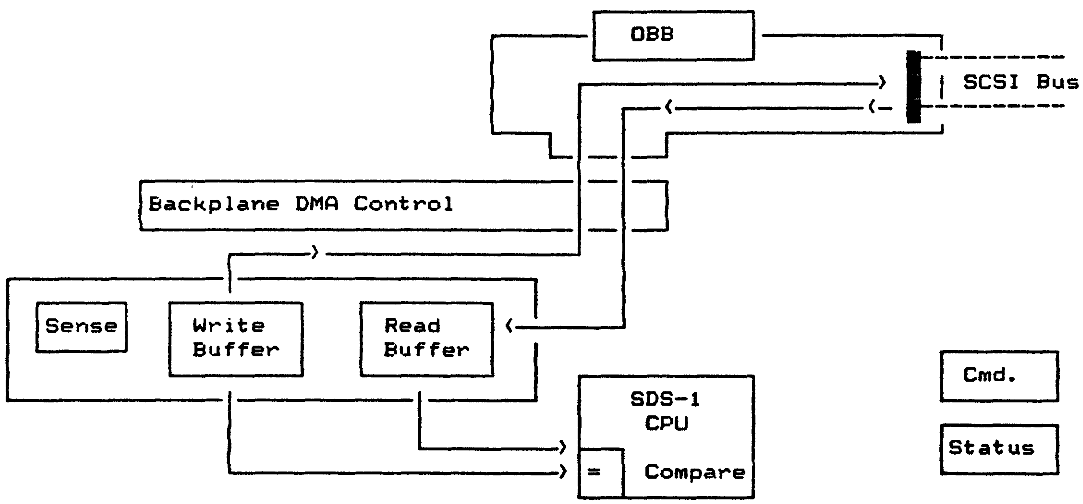


Backplane DMA Copy	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	Backplane DMA Controller	N.A.
Read Operation	BPM	16-> 32,768	Backplane DMA Controller	N.A.
Comments: Copy utilizes two separate DMA pointers one for write and one for read operations, creating a FIFO-type buffer for copy operations.				

IODVR.3.6.7 DMA SOFTWARE COMPARE (DMASC)

DMA Software Compare utilizes both a write buffer and a read buffer during operation. All handshaking is handled via the DMA logic.

FIGURE ~IODVR-F12. DMASC TRANSFER MODE BLOCK DIAGRAM

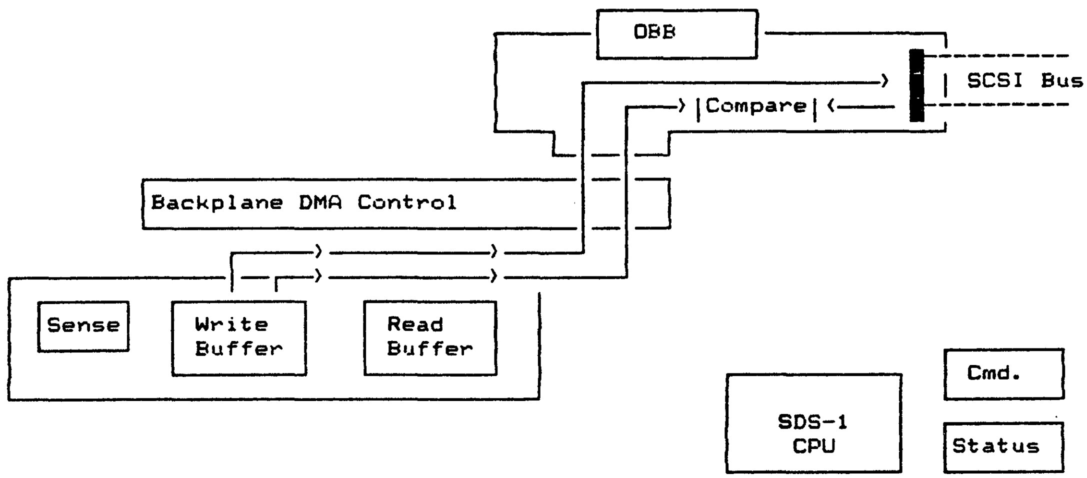


Backplane DMA Software Compare	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	Backplane DMA Controller	N.A.
Read Operation	BPM	16-> 32,768	Backplane DMA Controller	CPU Ref & Read Buffer Compare
Comments: I/O Driver will always try to fill read buffer then perform compare, until last read which may be less than a buffer in length.				

IODVR.3.6.8 DMA HARDWARE COMPARE (DMAHC)

DMA Hardware Compare utilizes the SDS-1 hardware comparator to perform "On-the-Fly" compares with the SCSI DATA IN and the write/ref buffer data.

FIGURE ~IODVR-F13. DMAHC TRANSFER MODE BLOCK DIAGRAM



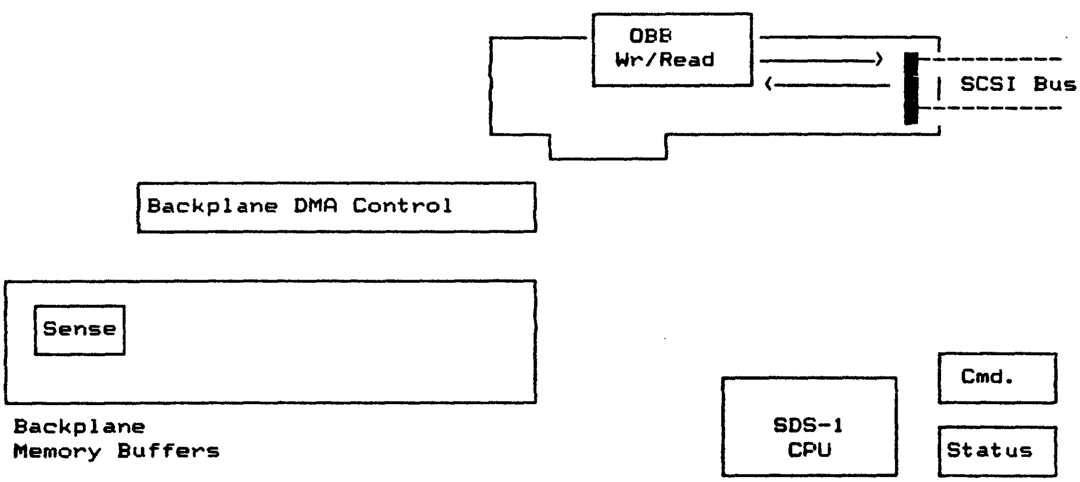
Backplane DMA Hardware Compare	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16-> 32,768	Backplane DMA Controller	N.A.
Read Operation	BPM	16-> 32,768	Backplane DMA Controller	Hardware "On-the-Fly" Compare
Comments:				

IODVR.3.6.9 HIGH-SPEED READ/WRITE COPY (HSRW/HSCOPY)

HSRW utilizes the SDS-1 High-Speed On-Board Buffer to transfer write and read data. All handshaking is handled via high-speed DMA logic.

Since the same buffer is used for both read and write operations (but with two different DMA pointers), the HSCOPY mode is identical to the HSRW mode.

FIGURE ~IODVR-F14. HSRW/HSCOPY TRANSFER MODE BLOCK DIAGRAM

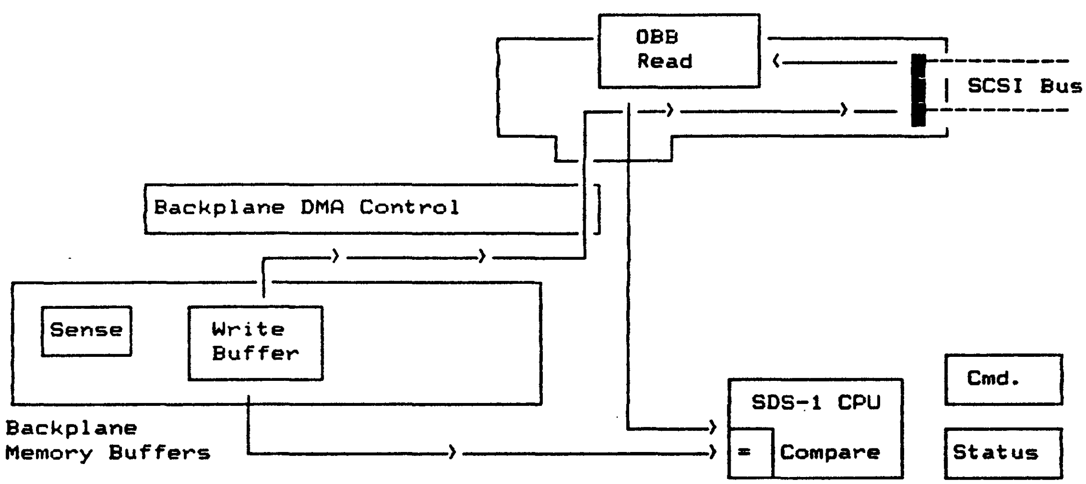


H.S. Copy and Read/Write	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	OBB	16,384	H.A. High-Speed DMA	N.A.
Read Operation	OBB	16,384	H.A. High-Speed DMA	N.A.
Comments: Both modes utilize two separate DMA pointers one for write and one for read operations.				

IODVR.3.6.10 HIGH-SPEED SOFTWARE COMPARE (HSSC)

High-Speed Software Compare is almost a contradiction in terms. The high-speed portion of the mode defines the high-speed DATA IN transfer from the SCSI bus to the on-board buffer. The software compare portion of the transfer is between the backplane memory write/ref buffer and the on-board buffer. In this mode, Write data is transferred from the backplane write/ref buffer via DMA write.

FIGURE ~IODVR-F15. HSSC TRANSFER MODE BLOCK DIAGRAM



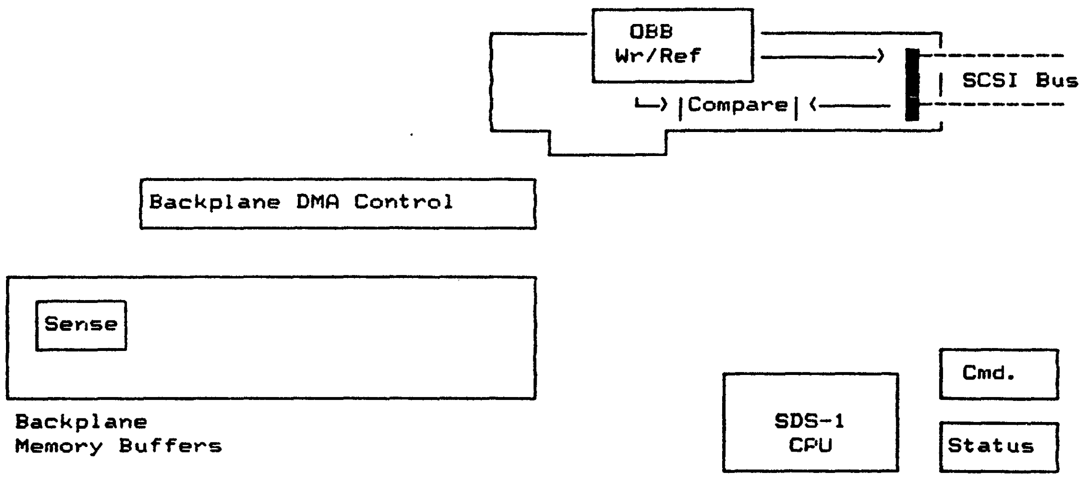
OBB High-Speed Software Compare	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	BPM	16,384	Backplane DMA Controller	N.A.
Read Operation	OBB	16,384	OBB H.S.DMA Controller	OBB Read & BPM Ref Buf. Comp.

Comments: I/O Driver will always try to fill read buffer then perform compare, until last read which may be less than a buffer in length.

IODVR.3.6.11 HIGH-SPEED HARDWARE COMPARE (HSHC)

High-Speed Hardware Compare utilizes the SDS-1 hardware comparator to perform "On-the-Fly" compares with the SCSI DATA IN and the on-board write/ref buffer data.

FIGURE ~IODVR-F16. HSHC TRANSFER MODE BLOCK DIAGRAM



OBB High-Speed Hardware Compare	Buffer	Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	OBB	16,384	H.A. High-Speed DMA	N.A.
Read Operation	OBB	16,384	H.A. High-Speed DMA	Hardware "On the Fly" Compare
Comments:				

IODVR.3.6.12 HIGH-SPEED VIRTUAL MEMORY (HSHCV)

One of the most powerful transfer modes is HSHCV. In this mode, the High-Speed On-Board Buffer is utilized in a virtual memory mode to simulate 256MB of random-access memory. This is accomplished via special hardware which double-increments the OBB address count after every 16K transfers. In other words, the buffer skips an address every wraparound. Figure IODVR-F17 shows the mapping of the 0x0 -> 0xFFFFFFF virtual address range into the physical 16K buffer. The operational details of the simulation are not important because the **dmaset_va()** and **dmaset_vblk()** functions provide access to the memory as if it were 256MB in size. (The user should utilize a **fillpr()** in order to guarantee a unique data pattern in every block over the entire 256MB range.)

FIGURE IODVR-F17. VIRTUAL/PHYSICAL BUFFER MAPPING

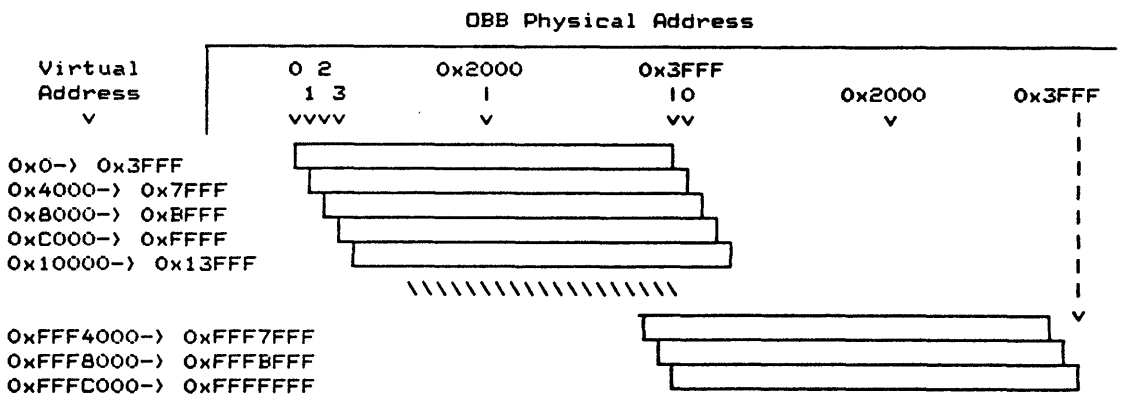
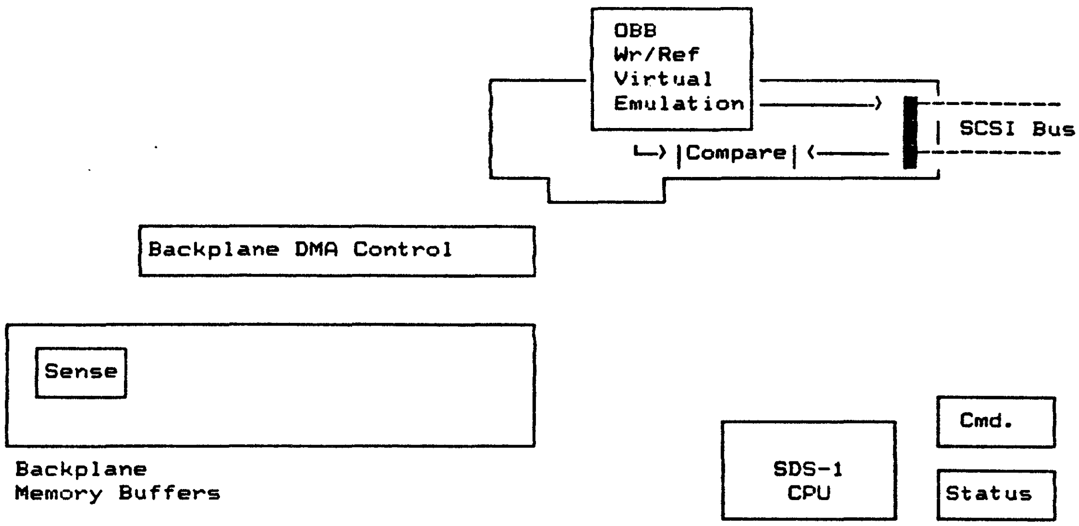


FIGURE IODVR-F18. HSHCV TRANSFER MODE BLOCK DIAGRAM



OBB High-Speed Hardware Compare	Buffer	Physical Size	Transfer Mechanism	Data Compare Mechanism
Write Operation	OBB	16,384	H.A. High-Speed DMA	N.A.
Read Operation	OBB	16,384	H.A. High-Speed DMA	Hardware "On-the-Fly" Compare

Comments: Virtual Memory Simulation provides a 2**28-1 memory space (fillpr() is required for pattern to be unique over entire range of 2**28-1).

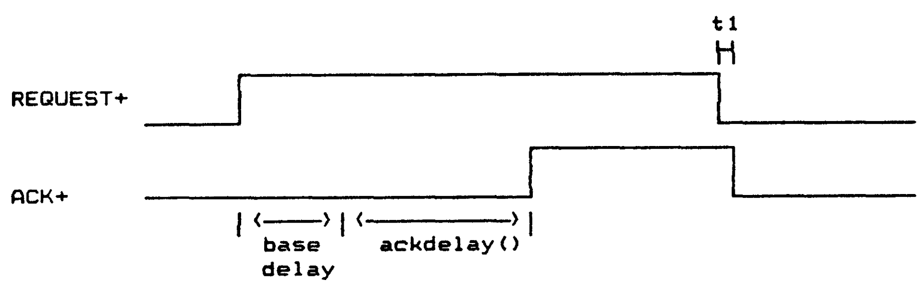
~IODVR.3.7 VARIABLE ACKNOWLEDGE DELAY

HSRW, HSSC, HSHC and HSHCV all utilize the SDS-1 on-board buffer. This buffer is equipped with special hardware which allows the user to vary the period from target REQ assertion to SDS-1 ACK assertion. **ackdelay()** adds delay in 70ns increments (for 0, 286 microseconds) to the base delay of the on-board buffer DMA logic. IODVR-T3 defines this delay for each transfer mode.

TABLE ~IODVR-T3. ACKNOWLEDGE DELAY

TRANSFER MODE	DATA IN BASE DELAY Min	DATA IN BASE DELAY Max	DATA OUT BASE DELAY Min	DATA OUT BASE DELAY Max
HSRW/HSCOPY	210ns	280ns	210ns	280ns
HSSC	210ns	280ns	NA	NA
HSHC	350ns	420ns	210ns	280ns
HSHCV	350ns	420ns	210ns	280ns

FIGURE ~IODVR-F19. REQ/ACK HANDSHAKE



t1 = REQ deassert to ACK deassertion greater than 70ns

~IODVR.3.8 BUSYWAIT

The **busywait()** function instructs the I/O Driver to retry SCSI commands which are completed with a BUSY status (SCSI Status byte = 0x08). This is particularly useful in the sequential environment where controllers return busy status during initialization. With **busywait()** enabled, the SDS-1 will continue to arbitrate and select the target until either the completion status is not busy and the command is executed or until an I/O time-out.

~IODVR.3.9 AUTOSENSE

With **autosense()** enabled, the SDS-1 will automatically perform a SENSE command anytime a check condition is reported from the target. The sense data will be reported in the error log (only the number of bytes transferred from the target will be displayed).

~IODVR.3.10 SCSI BUS STATE LOGGING

When **bus_logen()** is enabled, each I/O Driver transaction on the SCSI bus is recorded on the test adapter state log. The log entries are made at the end of each SCSI bus event. Time stamps are provided in the log. The user should be careful in the use of these time stamps (see STLOG section).

~IODVR.4 RETURN CODES

The I/O Driver is logically divided into two internal layers (see Figure IODVR-F20). Each layer has it own error handling and reporting structure. The Function Status io_stat and init_stat are the error messages from the I/O Driver and the Initiator layers, respectively.

Tables IODVR-T4 and IODVR-T5 define each of these return codes. With explicit/implicit error action **iea()** of LOGC OR LOGH these error codes will be reported in the log with verbal definition.

FIGURE ~IODVR-F20. I/O DRIVER INTERNAL PARTITION

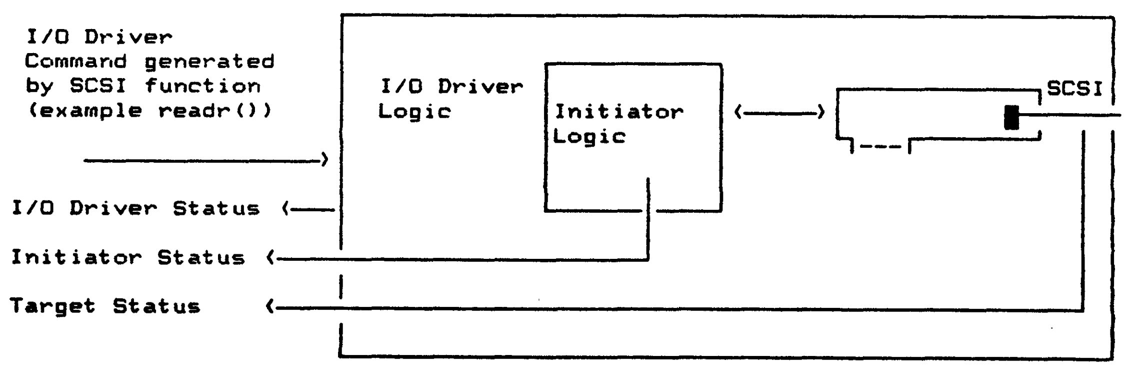


TABLE ~IODVR-T4. INITIATOR STATUS RETURN CODES

"C" DEFINE (*1)	VALUE	DEFINITION
#define GOOD	ØxØØ	good command completion
#define TIMEOUT	ØxØ5	I/O time-out
#define SELTO	ØxØ6	selection time-out
#define RESET	ØxØ9	SCSI reset detected
#define INVRSL	ØxØa	invalid reselection
#define RSLABT	ØxØb	reselection abort
#define INVPHC	ØxØc	invalid SCSI phase change
#define IVBFREE	ØxØd	invalid bus free detected
#define MCOMP	ØxØe	buffer miscompare
#define PRTYERR	ØxØf	SCSI inbound parity error
#define INTERR	Øx1Ø	internal I/O driver error

*1 DEFINE statements which can be used in "C" SAT

TABLE ~IODVR-T5. I/O DRIVER STATUS RETURN CODES

"C" DEFINE (*1)	VALUE	DEFINITION
#define GOOD	ØxØØ	good command completion
#define NOFIFO	ØxØ1	no active fifo
#define NORW	ØxØ2	no active r/w buffer
#define NOSB	ØxØ3	no active sense buffer
#define INVCMD	ØxØ4	invalid command code
#define HADERR	ØxØ8	host or test adapter detected error
#define NOHA	Øx2Ø	no physical host or test adapter
#define DUPID	Øx21	duplicate SCSI ID
#define MISCOMP	Øx8Ø	buffer miscompare
#define IOABT	ØxFF	I/O Abort from IOABRT

*1 DEFINE statements which can be used in "C" SAT

~IODVR.4.1 EXPECTED STATUS AND STATUS MASK

The **stat_mask()** and **exp_status()** functions provide the user with a means of redefining the SCSI status error state. Normally 0x00 status is considered a "passing" status. However, under certain conditions, check or busy may be the "passing" status and 0x00 is a "failing status." A 0 in the **stat_mask()** function excludes the status bit in that bit position from being compared to the **exp_status()** value. If the masked SCSI status and the expected status do not match, a fail log error entry is made along with the expected and actual status.

~IODVR.5 STATISTICS GATHERING

Each I/O Driver execution results in a function statistics generation. These statistics include:

bytes written	32-bit counter
bytes read	32-bit counter
bytes compared	32-bit counter
# of miscompares	32-bit counter

If **statsen()** is set, global statistics will be accumulated after each I/O Driver operation. These statistics include:

number of I/O Driver Operations	32-bit counter
number of Initiator-Detected Errors	32-bit counter
number of unexpected Target Errors	32-bit counter
bytes written	32-bit counter
bytes read	32-bit counter
bytes compared	32-bit counter
# of miscompares	32-bit counter

~IODVR.6 SENSE HANDLING

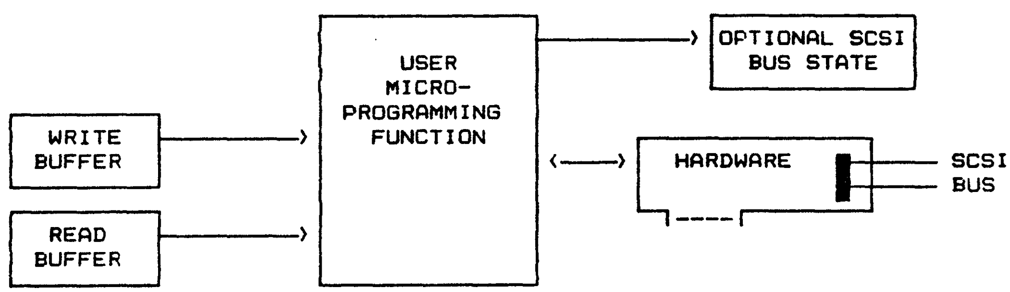
Due to its extensive usage of the SCSI, **sense()** command is handled specially. First, compare-type transfer modes would normally try to compare the sense data-in against the write/ref buffer. This is averted by changing the transfer mode for **sense()** commands to RW and pointing the read data to the start of the sense buffer. After the **sense()** is complete, the transfer mode is restored and the read pointer disappears. In other transfer mode cases, the read pointer is simply redirected to the start of the sense buffer. The sense buffer is located in backplane memory, any OBB transfer modes will be switched to DMARW in order to perform the **sense()** command.

~MP.Ø MICROPROGRAMMING

~MP.1 EXECUTION ENVIRONMENT

Microprogramming allows the user to take complete control of the SCSI bus initiator functions and generate complex bus sequences, as well as generate controlled errors on the SCSI bus. Unusual or illegal message sequences are easily created. Parity error can be forced on a given byte and true arbitration can be forced on a nonstatistical basis. Figure MP-F1 presents the Microprogramming execution environment.

FIGURE ~MP-F1. MICROPROGRAMMING EXECUTION ENVIRONMENT



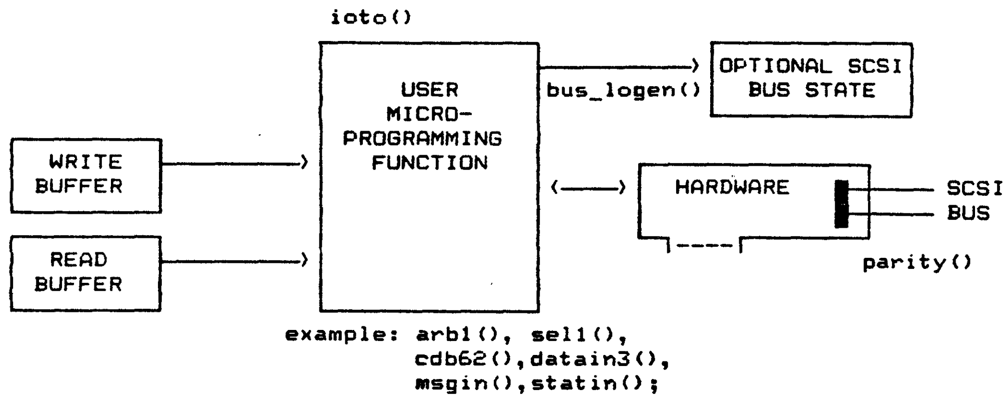
Microprogramming can be viewed as the set of internal functions necessary to create an I/O Driver. In order to maintain consistency, the microprogramming functions behave very similarly to the I/O Driver. The user should reference the IODVR section listed below for an understanding of the following topics:

BUFFER MANAGEMENT	see IODVR.2
BUFFER WRAPAROUND	see IODVR.2.1
DATA COMPARISON	see IODVR.2.2
HARDWARE DATA COMPARE	see IODVR.2.2.1

~MP.2 CONTROL FUNCTIONS

Figure MP-F2 shows the Microprogramming Execution Environment with its various control functions. Control over functions such as arbitration, selection, and message support are totally up to the user in how he utilizes the various microprogramming functions.

FIGURE ~MP-F2. MICROPROGRAMMING CONTROL FUNCTIONS



See the IODVR section for detailed information on the following functions (also refer to Appendix A):

ioto()	see	IODVR.3.1
parity()	see	IODVR.3.2
Transfer Modes	see	IODVR.3.6
datain0() (HS Read)	see	IODVR.3.6.9
datain1() (DMA Read)	see	IODVR.3.6.5
datain2() (TR Read)	see	IODVR.3.6.3
datain3() (PIO Read)	see	IODVR.3.6.1
datain4() (DMA Hardware Compare)	see	IODVR.3.6.8
datain5() (HS Hardware Compare)	see	IODVR.3.6.11
dataout0() (HS Write)	see	IODVR.3.6.9
dataout1() (DMA Write)	see	IODVR.3.6.5
dataout2() (TR Write)	see	IODVR.3.6.3
dataout3() (PIO Write)	see	IODVR.3.6.1
ackdelay()	see	IODVR.3.7
bus_logen()	see	IODVR.3.10

~MP.2.1 FUNCTION STATUS

Each Microprogramming function generates an initiator status and I/O Driver status. This is done to maintain consistency between the I/O Driver and Microprogramming. Detailed information on the function status can be found in the Function Library Definitions (Appendix A) and in IODVR.4 .

~MP.2.2 STATISTICS GATHERING

Each of the Microprogramming data transfer functions (datains and dataouts) generates function statistics. These statistics are available via **get_f_stats()**. In addition, if **statsen()** is set, these statistics will be accumulated in the global statistics. The user should reference IODVR.5 for additional information on statistics.

~MP.3 ARBITRATION TESTING

The SDS-1 utilizes dedicated hardware to truly test SCSI bus arbitration. By utilizing a third party busy (see Figure MP-F3), the SDS-1 is able to generate a head-to-head arbitration conflict which the TARGET may win or lose. Figure MP-F4 shows a sample SAT utilizing the **forcbusy()**, **arbwin()**, and **arblose()**.

FIGURE ~MP-F3. ARBITRATION TEST ENVIRONMENT

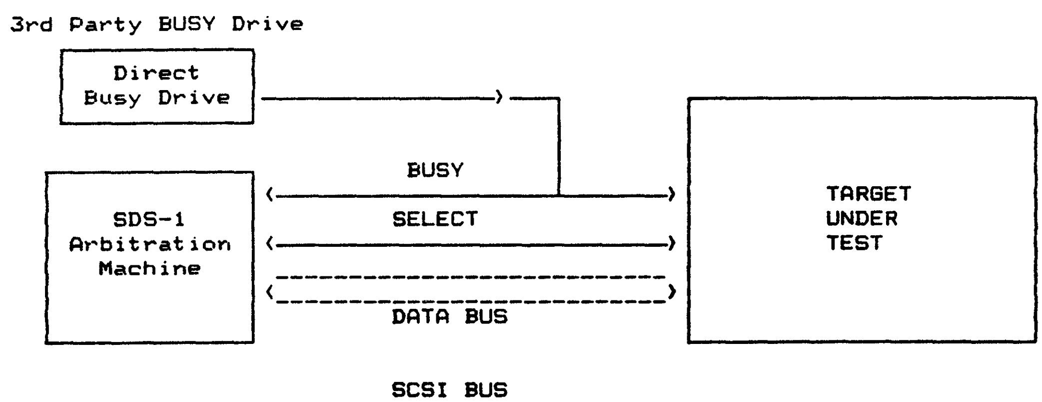


FIGURE ~MP-F4. EXAMPLE ARBITRATION SAT

```
arbtest.c                                    8-01-86  13:59:58  PAGE     1

/* -DB=;
;-DOC
;-REV
;         Created:   6/10/85
;Initial Release:   7/11/85
;        Revision:   1/10/86
;                    3/26/86  Update for Tech Ref Manual
;
;-REV
;
;Microprogramming Arbitration Example
;
;Purpose: Force Arbitration between ACB 5500 and SDS-1
;
;Setup
;   Adaptec 5500 (SCSI/ST506) Disk Controller with ST506 drive
;
;-DOC */

#define HOSTID 0x07
#define TARGETID 0x04

user_test() {
   test("Microprogramming Arbitration Test");

/*-DOC
;-GT="Arbitration Test");
;
;-AI="ARB.PIC"
;
;-PT="Disconnect Setup"
;
; Rezero Unit and then issue a Seek Command
; which will result in a disconnect
; -DOC */

   group("Arbitration Test");
   paragph("Disconnect Setup");
   tid(TARGETID);                   /* target ID */
   ureset();                        /* reset */
   parity(1);                       /* parity enabled */
   bus_logen(1);                    /* state log enabled */
   rezero();                        /* rezero unit */
   arb2(HOSTID);                    /* host arb */
   sel4(TARGETID,0xC0);             /* select target with disconnect */
   cdb62(0x0B,00,0x10,00,00,00); /* seek command */
   forcbusy();                      /* force busy */
   msgin(0x02);                     /* save data pointer message */
   msgin(0x04);                     /* disconnect message */
   delays(1);                       /* be sure target is trying
                                       to reconnect */

/*-DOC
;-PT="Verify Arbitration Loss by Target"
```

FIGURE MP-F4. EXAMPLE ARBITRATION SAT (continued)

```
arbtest.c                                    8-01-86  13:59:58  PAGE     2

        ;
        ;
        ; -DOC */

           paragph("Verify Arbitration Loss by Target");
           arblose(0x07);                /* verify target lost */
           arblose(0x06);
           arblose(0x05);

        /*-DOC
        ;-PT="Verify Arbitration Win by Target"
        ;
        ; Check win against lower I.D.
        ;
        ; -DOC */

           paragph("Verify Arbitration Win by Target");
           arbwin(0x03);                 /* target should win arbitration */
           resel();                      /* reselection */

           msgin(0x80);                  /* identify */
           msgin(0x03);                  /* restore pointers */
           statin(0x00);                 /* good completion status */

        /*-DOC
        ;-PT="Bus Free Verification"
        ;
        ;Check for good completion and bus free
        ;
        ; -DOC */

           paragph("Bus Free Verification");
           bfreearm();                   /* verify bus goes free after compl
        etion */
           msgin(0x00);                  /* command complete message */
           delays(1);                    /* delay for target to release bus
        */
           bfreeck();                    /* check the bus has gone free */

        }
```

~MP.4 PARITY ERROR GENERATION

Parity error generation on a given outbound byte (command out, data out, or message out) can be generated utilizing the **forcperr()** function. An example of this is shown in Figure MP-F5.

FIGURE ~MP-F5. PARITY ERROR TESTING EXAMPLE

```
                                    /* -DB=;
                                    ;-DOC
                                    ;-REV
                                    ;        Created:   6/8/85
                                    ;Initial Release:   7/1/85
                                    ;       Revision:  1/10/86
                                    ;
                                    ;-REV
                                    ;
                                    ;Parity Error Generation Example
                                    ;
                                    ;Purpose:  Generate Parity Errors during
                                    ;differet information out phases
                                    ;
                                    ;Setup
                                    ;   Adaptec 3530P (SCSI/QIC-36)
                                    ;   Streaming Tape Controller with
                                    ;   QIC 36 Drive
                                    ;
                                    ;-DOC */
user_test()

{
  int host=7;
  int target=0;

  test("Parity Error Generation Example");

                                    /* -DOC
                                    ;GT="Initialization"
                                    ;Define SCSI path and enable parity
                                    ; -DOC */
  group("Initialization");
  init();
  ureset();
  delays(15);
  filli(00,00,00);
  sense(0x10);
                                    /* -DOC
                                    ;GT="Parity Error on Command Out"
                                    ;Generate Parity on 5 byte of command out
                                    ; -DOC */
  group("Parity Error on Command Out");
  arb2(host);
  sel3(target);
```

FIGURE MP-F5. PARITY ERROR TESTING EXAMPLE (continued)

```
forcperr(4);
cdb62(01,00,00,00,00,00);
statin(02);
msgin(00);
sense(0x10);
sbb(04,02);
                                      /* -DOC
                                      ;GT="Parity Error First Block of Data Out"
                                      ;Generate Parity on byte 0x80 of write block
                                      ; -DOC */
group("Parity Error on First Block of Data Out");
arb2(host);
sel3(target);
cdb62(0x0a,01,00,00,0x10,00);
dataout1(0x100L,1);
forcperr(0x80);
dataout1(0x100L,2);
statin(02);
msgin(00);
sense(0x10);
sbb(04,02);
uprwd(0);
                                      /* -DOC
                                      ;GT="Parity Error on 100th block of data out"
                                      ;Generate Parity on byte 0x80 of write block
                                      ; -DOC */
group("Parity Error on 100th Block of Data Out");
arb2(host);
sel3(target);
cdb62(0x0a,01,00,01,00,00);
dataout1(0xC600L,1);
forcperr(0x80);
dataout1(0x200L,2);
statin(02);
msgin(00);
sense(0x10);
sbb(04,02);
rewind(0);
                                      /* -DOC
                                      ;GT="Verify Good Data"
                                      ;After Tape is rewound verfiy first 99
                                      ;blocks written ok
                                      ; -DOC */
group("Verfify Good Data");
reads(99);                            /* use I/O driver reads */
                                      /* -DOC
                                      ;GT="Verify 100th block did not get Written"
                                      ;Verify end of meida after 99th block
                                      ; -DOC */
group("Verify 100th block did not get Written");
arb2(host);   sel3(target);
cdb62(0x08,01,00,00,01,0); statin(2);  msgin(0);
sense(0x10);   sbb(8,02);      /* end of recorded media */
}
```

(THIS PAGE INTENTIONALLY LEFT BLANK)

~STLOG.Ø BUS STATE LOG

~STLOG.1 INTRODUCTION

The SCSI Bus State Log is a powerful debugging tool which allows the user to capture SCSI bus events and examine them in an easy-to-read SCSI hierarchical format. The state log is utilized by both the I/O Driver and Microprogramming environments. (See Figures STLOG-F1 and STLOG-F2.)

The state log is a software log of the SCSI events occurring between the SDS-1 and a SCSI Target. It is not a third-party hardware logic analyzer watching the SCSI bus. Since the logging function is performed in software, processing time will be taken away from the I/O Driver or the Microprogramming operation. The state log is designed to minimize this time, none-the-less it will effect the I/O operation. In situations where logging is not needed **bus_logen()** can be utilized to turn off the state log.

FIGURE ~STLOG-F1. I/O DRIVER EXECUTION ENVIRONMENT

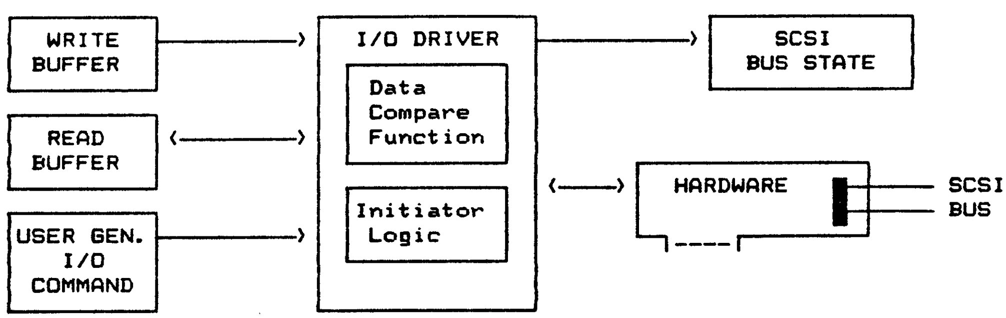
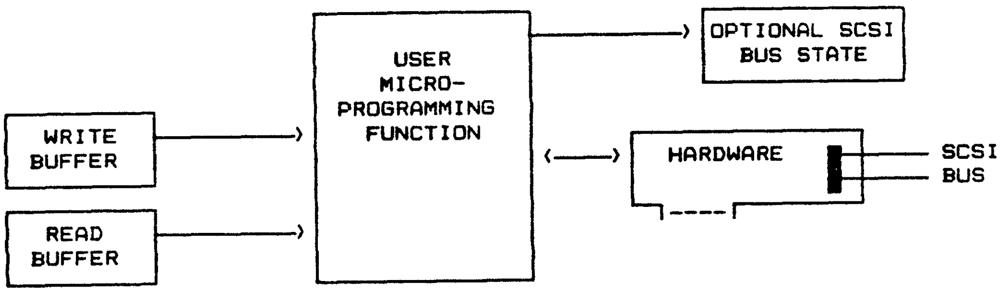


FIGURE ~STLOG-F2. MICROPROGRAMMING EXECUTION ENVIRONMENT



~STLOG.1.1 DATA ACQUISITION/DISPLAY

The state log acquisition memory is a 1024-word-deep FIFO which stores information (start time, event description and its data, end time and line number) for each state log update call. The state log display logic is responsible for translating this raw compacted data into the display format. While certain SCSI events are stored in the state log on a byte-by-byte basis (such as command out) they are better understood if displayed on one or two lines. This is the case for command out information. Refer to Figure STLOG-F3 for an actual state log display.

FIGURE ~STLOG-F3. STATE LOG DISPLAY

```
Dbuf( Buf:   W; Strt: 0000; Len: 0020; On: D  B Grouping)

rptbuf(L,    0,   10)                              07-09-86 09:17:29

 Start Time              Event Description               End Time   Line #
+----------+-----------------------------------------+----------+-------
 0083.62869          Message in 00                                 00F
 0083.62965  Bus Free Detected                                     00E
 0086.62319      Arbitration as 07                    0086.62346   00D
 0086.62441         Selection ids = (1001 0000b)      0086.62489   00C
 0086.62521             Message out C0                             00B
 0086.62600                Command out 08 00 00 00 80 00  0086.62849   00A
 0086.62886             Message in 02                              009
 0086.62987             Message in 04                              008
 0086.63077  Bus Free Detected                                     007
 0086.65187         Reselection  ids = (1001 0000)                 006
 0086.65203             Message in 80                              005
 0086.65287             Message in 03                              004
 0086.65664                   Data In     8000H byte(s)            003
 0086.89434                Status in 00                            002
 0086.89508             Message in 00                              001
 0086.89604  Bus Free Detected                                     000
```

Since the state log is a FIFO, it always records the most recent bus events pushing old information up and eventually out of the FIFO. When displaying the state log, line Ø represents the last transaction on the bus with high lines (1, 2, 3, ..., 3FFh) representing aging transactions.

~STLOG.2 STATE LOG ENTRIES

Each State Log Entry is basically comprised of five fields. These fields are

START TIME	EVENT DESCRIPTION	DATA (optional)	END TIME (optional)	LINE #

Table STLOG-T1 shows all the possible state log entries along with comments regarding each entry.

TABLE ~STLOG-T1. STATE LOG SUMMARY

EVENT FIELD	DATA FIELD	COMMENTS
--- Test Initialization ---	N.A.	Appears each time a SAT or MENU is executed
Arbitration as hh Arbitration as hh (incomplete) Arbitration as hh (lost)	Initiator ID	Arbitration (incomplete arb - a time out or bus reset occurred) (lost arb - only during software arbitration)
blank	N.A.	After SDS-1 system reset, log is reset
Bus Reset Asserted	N.A.	SDS-1 has forced a SCSI bus reset
Bus Reset Detected	N.A.	Bus reset detected
Bus Free Detected	N.A.	SCSI Bus free has been detected
Command out hh hh hh hh hh hh	Command Bytes	SCSI command bytes
Data in hhhhH bytes(s) Data out hhhhH bytes(s)	Bytes Transferred	All contiguous DATA IN/OUT log entries are displayed as one line
Message in hh	Message Value	SDS-1 inbound message
Message out hh	Message Value	SDS-1 outbound message
Reselection ids = (bbbb bbbb)	Reselect IDs	Reselect
Selection ids = (bbbb bbbb)	Select IDs	SDS-1 selection of SCSI Target
Status in hh	Status In Value	TARGET status

hh = hexidecimal
bbbb = binary

~STLOG.3 TIME STAMPING

Most of the State Log entries are time-stamped with a start and end time. These times are logged in seconds with a resolution of 5Ø usec. Some events display only the start time; in these cases, the time elapsed is either trivial (i.e., message out) or can be derived from other events (i.e., the end of a data phase is typically the start of the following status or message phase). This time stamp is read from the SDS-1 real-time clock and stored along with the event code and related data. For command transfers of greater than 6 bytes, the start time is displayed on the first line and the end time is displayed on the second line.

~STLOG.4 STATE LOG REDUCTION FUNCTIONS

RTFL provides two reduction functions for obtaining information from the state log. The **delta_time()** function gets the real time elapsed between 2 bus state log entries and the **state_data()** function gets the data associated with a particular state log entry (refer to Appendix A for more information on these functions).

~DEBUG.Ø SDS-1 DEBUGGER

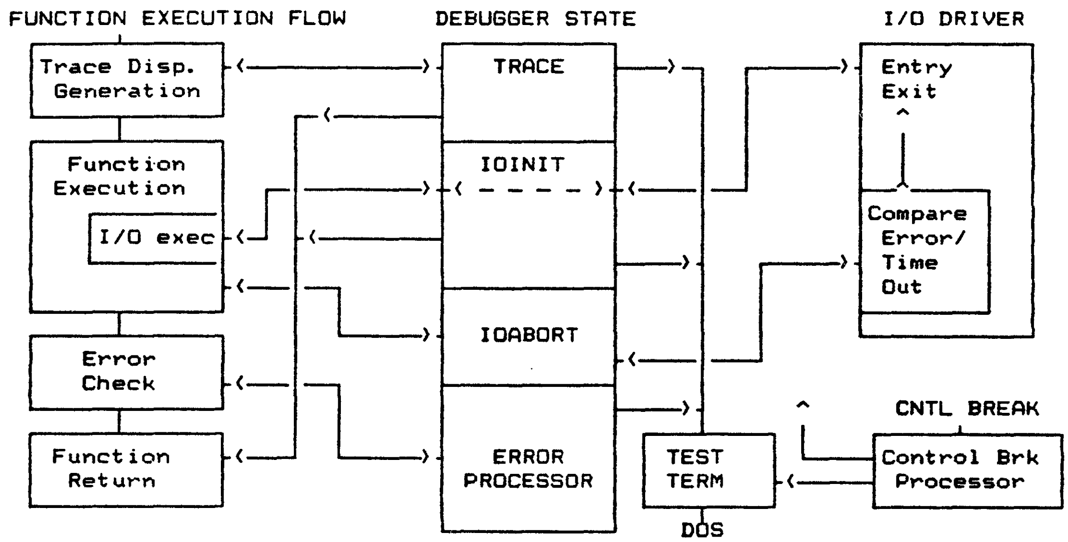
~DEBUG.1 INTRODUCTION

The SDS-1 Debugger is a programming tool that provides both debugging and statistics gathering functions. The debugger is an integrated portion of the test function and documentation library and is used in both the MENU and C compiler environments. Some features of the Debugger are listed below:

- Real Time Statistics Display
- Real Time Buffer and SCSI Command Block Display
- Test/Documentation Function Trace Display
- Read/Write Buffer Display
- Host Adapter State Transaction Log Display
- Real Time Execution Interruption <ESC>
- Break Point/Single-Step Execution Control
- Explicit/Implicit Error Action Mode Selection.

The debugger has four basic states: **TRACE, IOINIT, IOABRT** and **ERROR PROCESSOR.** The **TRACE** state allows control over the executing environment. The **IOINIT** state updates statistics and displays the Command Descriptor Block (CDB) pending execution. The **IOABRT** state provides the user with an error trap within the I/O Driver or Microprogramming functions to deal with function time-outs and data compare errors. The **ERROR PROCESSOR** is the central point for all SDS-1 Implicit and Explicit Error Handling. Figure DEBUG-F1 shows a "State Diagram" of Function Execution flow, the Debugger and I/O Driver. The interaction and flow between states will be developed in succeeding sections.

FIGURE ~DEBUG-F1. DEBUGGER STATES



Terminology Definition:

While the Debugger will be defined in the following sections, certain terms will be used that are defined below to aid in understanding the Debugger.

Debugger Level: Refers to the Debugger display format.

Debugger State: Refers to one of four Debugger states: **TRACE, IOINIT, IOABRT** or **ERROR PROCESSOR**

Debugger Command Line: Refers to the line at the bottom of the display so that the user can interact with the Debugger. Each Debugger state has its own command or menu line (set).

The SDS-1 MENU System utilizes the Debugger (in debug display level 3) as its execution environment. The user can also change the debug display level and invoke the Debugger command line from a Stand-Alone Test. This is very useful in "debugging" both the SAT and sometimes the system under test.

~DEBUG.1.1 SAT COMMAND TAIL INVOCATION

The SDS-1 Debugger Display can be invoked from the DOS command line via a **-DB**=n command tail operator where n is the debug level. There are four debug levels (Ø to 3), the default level is Ø (no debug operation). The following is an example of the command line invocation:

C>**TESTPROG -DB=2**

The above command starts the execution of test file, **TESTPROG,** with the debug level 2 display in the **TRACE** state command line. The **TRACE** state displays the pending function and its argument(s) and it also allows the user to examine buffers and/or control the debug mode (refer to Section DEBUG.3.1 for more on the **TRACE** state). The command tail **-PR** operator can be used in conjunction with the **-DB=** to direct the Documentation output to the system printer, for example:

C>**TESTPROG -DB=1 -PR**

will invoke the Debugger with a debug level 1 display and output the report scrolling window to the printer.

~DEBUG.1.2 FUNCTION INVOCATION

The Debugger can be invoked from within a SAT via the **debug(n)** function. This function sets the debug display mode and causes program execution to halt on the **debug(n)** function. At this point, the user has the option to skip or execute the debug function. If the user decides to skip, the debug level remains unchanged. Otherwise, if the **debug()** function is executed, the debug display level will change to the level selected and the **TRACE** command line will be displayed with the next program function ready to execute.

~DEBUG.1.3 ERROR ACTION INVOCATION

Many times it is desirable to invoke the Debugger only on an error condition. Table DEBUG-T1 defines the Error Processor logic and how it relates to the Debugger. When the Debugger **ERROR PROCESSOR** command line is invoked from the error processor it remains in the currently defined debug level.

TABLE ~DEBUG-T1. BATCH OR SAT ERROR ACTION

IEA or EEA ERROR ACTION	BATCH MODE	SAT MODE (non-batch mode)	MENU INTERFACE
CONT	Continue	Continue	Not defined
HALT	Exit to DOS and execute next SAT	Invoke Debugger ERROR PROCESSOR Command Line	Not defined
LOGC	Log and continue up to error limit value otherwise, exit to DOS	Log and continue up to error limit value, otherwise, invoke Debugger ERROR PROCESSOR command line	
LOGH	Log and exit to DOS	Log and invoke ERROR PROCESSOR command line	

The implicit and explicit error action defaults to LOGC. To change the error action, use the Debugger **IEA** or **EEA** command in the **TRACE** command line or use the function library **iea()** and **eea()** functions in the SAT or MENU. The error limit default is 100 errors; to change this limit, use the **set_er_limits()** function.

~DEBUG.1.4 MENU INTERFACE INVOCATION

The SDS-1 Menu system utilizes Debug Display Level 3 as its execution environment. This debug level provides the user with statistics, CDB, DMA Pointers and Function Trace Displays. Prior to FKEY execution the user may invoke the debug **TRACE** command line by setting the debugger function in the FKEY menu to S for single step. During FKEY execution, **ESC** can be used to access the **TRACE** command line.

Single-key execution cannot invoke the Debugger **TRACE** state but error action of LOGC or LOGH can invoke the error processor command line to resume or exit from the **TRACE** command line to the **MENU**.

~DEBUG.2 DEBUGGER DISPLAY

The Debugger Display consists of a Primary and a Secondary Display Screen. The Primary display shows various data or information on the executing test or design verification batch file. The Secondary display is used for buffer and state log displays.

~DEBUG.2.1 PRIMARY DISPLAY SCREEN

The Primary Display Screen is a function of the Debug Level (refer to Figures DEBUG-F2 through DEBUG-F5). The four debug levels are designed to step the user from the debug environment to the SAT environment with the end goal of executing a SAT which simply generates a pass or fail result (debug level 3 to Ø). The debugger display manager handles two fixed windows, two scrolling windows, and one fixed line. Table DEBUG-T2 summarizes the display for each debug level.

TABLE ~DEBUG-T2. DEBUGGER DISPLAY WINDOWS

DEBUGGER DISPLAY LEVEL	FIXED WINDOWS	SCROLLING WINDOW	FIXED LINE
0	Fixed Documentation	Report Display	The fixed line is utilized for Debugger message reporting and debug command line presentation and interaction. In addition, functions such as user_input() will interface with the user on this line.
1	Fixed Documentation Status	Report Display	
2	Fixed Documentation Status	Trace Display	
3	Status	Trace Display	

FIGURE ~DEBUG-F2. DEBUG LEVEL 0

```
 ---------------- ADAPTEC Test Structure Library (11-30-84) ----------------
                        DOS Command Line Execution
                          01-08-86    11:45:17
                                                  Printer Output Disabled:
   1.0 On Board Buffer Write/Read/Compare Testing       01-08-86 11:45:20
      1.2 Read and Compare (via DMAHC) OBB Write Data   01-08-86 11:51:46
         1.2.9 Pseudo Random DMAHC Read                 01-08-86 12:00:33
 ------------------------------ REPORT DISPLAY ------------------------------
1.2.6 00 FF 55 AA DMAHC Read                      01-08-86 12:00:15

1.2.7 Incrementing Pattern DMAHC Read             01-08-86 12:00:17

1.2.8 Decrementing Pattern DMAHC Write            01-08-86 12:00:21
IOABORT IMPLICIT ERROR                            01-08-86 12:00:22
Cmp Error: Ref Buf(0x0000 = 0x04); SCSI Data = 0x22;

IOABORT IMPLICIT ERROR                            01-08-86 12:00:32
I/O Time Out (Time Out Value = 10 seconds)


1.2.9 Pseudo Random DMAHC Read                    01-08-86 12:00:33
```

FIGURE ~DEBUG-F3. DEBUG LEVEL 1

```
 ---------------- ADAPTEC Test Structure Library (11-30-84) ----------------
                        DOS Command Line Execution
                          01-08-86    11:45:17
                                                  Printer Output Disabled:
   1.0 On Board Buffer Write/Read/Compare Testing       01-08-86 11:45:20
      1.1 OBB Fill Testing                              01-08-86 11:45:20
         1.1.6 00 FF 55 AA  OBB Write                   01-08-86 11:45:49
 ---------------------------- I/O DRIVER STATUS ----------------------------
 I/O  Ops:        5 | uc0:          | I/O Command Parameters   stat: 00 __
 TGT Chks:        0 | uc1:          | CDB: 0a 00 01 00 40 00   sense: (old)
 INT D Er:        0 |---------------|      00 00 00 00 00 00   __ __ __ __
 Bytes Wr:    28000 | Wr/Ref: OBB   | xfer: HSRW     a.s.:OFF  __ __ __ __
 Bytes Rd:        0 |         0000  | s.l.ON  arb.HDW sel.SMA  __ __ __ __
 Bytes Cp:        0 | Rd Buf: OBB   | b.p.OFF b.w.OFF          __ __ __ __
 Cmp Ers.:        0 |         0000  | ha: 0   iid: 7   tid: 4  __ __ __ __
 ----------------------------- REPORT DISPLAY ------------------------------
1.1.4 Constant 55 Pattern OBB Write               01-08-86 11:45:42

1.1.5 1233210 Pattern OBB Write                   01-08-86 11:45:46

1.1.6 00 FF 55 AA  OBB Write                      01-08-86 11:45:50

TRACE:  <ESC> Halt :
```

FIGURE ~DEBUG-F4. DEBUG LEVEL 2

```
------------------ ADAPTEC Test Structure Library (11-30-84) ------------------
                         DOS Command Line Execution
                           01-08-86    11:45:17
                                                      Printer Output Disabled:
  1.0 On Board Buffer Write/Read/Compare Testing         01-08-86 11:45:20
     1.1 OBB Fill Testing                                01-08-86 11:45:20
        1.1.11 Word Block Count OBB Write                01-08-86 11:48:34

------------------------------ I/O DRIVER STATUS ------------------------------
 I/O  Ops:        E | uc0:              | I/O Command Parameters    stat: 00 __
 TGT Chks:        0 | uc1:              | CDB: 0a 00 09 00 40 00    sense: (old)
 INT D Er:        0 |-------------------|      00 00 00 00 00 00    __ __ __ __
 Bytes Wr:    68000 | Wr/Ref: OBB       | xfer: HSRW    a.s.:OFF    __ __ __ __
 Bytes Rd:        0 |           0000    | s.l.ON  arb.HDW sel.SMA   __ __ __ __
 Bytes Cp:        0 | Rd Buf: OBB       | b.p.OFF b.w.OFF           __ __ __ __
 Cmp Ers.:        0 |           0000    | ha: 0   iid: 7   tid: 4   __ __ __ __
-------------------------------- TRACE DISPLAY --------------------------------
filli(7e,0000,4000) writer(0300,0) paragph() ackdelay(243)
filld(04,0000,4000) writer(0400,10) writer(0410,1f) writer(042f,11)
paragph() ackdelay(154) fillpr(008a,0000,4000) writer(0e00,10)
writer(0e10,1f) writer(0e2f,11) paragph() ackdelay(6020)
fillbcb(90,0100,0000,4000) writer(0900,40) paragph() ackdelay(2100)
fillbcw(0940,0100,0000,4000)
TRACE : Control >Debug Level(2); BCU(1); User Cntr Reset; Stats Reset;
```

FIGURE ~DEBUG-F5. DEBUG LEVEL 3

```
------------------------------ I/O DRIVER STATUS ------------------------------
 I/O  Ops:       2F | uc0:              | I/O Command Parameters    stat: 00 __
 TGT Chks:        0 | uc1:              | CDB: 08 00 00 c0 40 00    sense: (old)
 INT D Er:        0 |-------------------|      00 00 00 00 00 00    00 00 00 00
 Bytes Wr:    F0400 | Wr/Ref: BPM       | xfer: DMAHC   a.s.:OFF    __ __ __ __
 Bytes Rd:    50000 |           0000    | s.l.ON  arb.HDW sel.SMA   __ __ __ __
 Bytes Cp:    20000 | Rd Buf:           | b.p.OFF b.w.OFF           __ __ __ __
 Cmp Ers.:        0 |                   | ha: 0   iid: 7   tid: 4   __ __ __ __
-------------------------------- TRACE DISPLAY --------------------------------
writer(0580,40) overbcw(05c0,0100,0000,4000) writer(05c0,40)
overbcw(0600,0100,0000,4000) writer(0600,40) overbcw(0640,0100,0000,4000)
writer(0640,40) overbcw(0680,0100,0000,4000) writer(0680,40)
overbcw(06c0,0100,0000,4000) writer(06c0,40) paragph() ackdelay(2100)
fillpr(009f,0000,0200) savebuf(OBBIMG.TST,0000,0200) writer(0a00,2)
paragph() dmarst(R) ackdelay(0) readr(0000,0040) paragph() dmarst(R)
ackdelay(15) readr(0040,0040) paragph() dmarst(R) ackdelay(255)
readr(0080,0040) paragph() dmarst(R) readr(00C0,0040) paragph() dmarst(R)
readr(0300,001F) readr(031F,0020) readr(033F,0001) paragph() ackdelay(0)
dmarst(R) readr(0900,0001) readr(0901,0010) readr(0911,000F)
readr(0920,0020) group() xfermode(DMAHC,4000) paragph() fillk(00,0000,4000)
readr(0000,0040) paragph() fillk(F,0000,4000) readr(0040,0040) paragph()
fillk(AA,0000,4000) readr(0080,0040) paragph() fillk(5,0000,4000)
readr(00C0,0040) paragph()
TRACE : Flow >Goto; Break Pt.(0); Run; Step; Half Step; Skip; DOS Ret;
```

DEBUG.2.1.1 TEST DOCUMENTATION FIXED WINDOW

The Test Documentation Fixed Window provides the state of the Test or Design Verification in progress. Appearing on the upper half of the screen for all debug levels except level 3, this window provides a date/time-stamped indication of the BATCH FILE (if any), TEST, SUBTEST, PARAGRAPH and SUBPARAGRAPH currently being executed (refer to Figures DEBUG-F2 through DEBUG-F5).

DEBUG.2.1.2 TEST DOCUMENTATION SCROLLING WINDOW (REPORT DISPLAY)

The T.D. Scrolling Window (Report Display) provides a view of the test document which is being generated by the test execution and the Test Documentation Features of the SDS-1, in other words, it displays the execution results. This window is also used for explicit and implicit error message displays as well as an output display for any information generated by a Test. This window appears on the lower half of the screen for debug level 0 and 1 with a label of "REPORT DISPLAY" (refer to Figures DEBUG-F2 and DEBUG-F3).

DEBUG.2.1.3 STATUS FIXED WINDOW

The Status Fixed Window provides the user with four frames of real-time information. The I/O Driver Status window appears in the center of the screen in debug level 1 and 2 and on the top of the screen in level 3 (refer to Figures DEBUG-F3 to DEBUG-F5).

DEBUG.2.1.3.1 STATISTICS FRAME

Whenever Statistics Display is enabled (**statsen(1)**), the Statistics Frame will reflect the current test statistics and each I/O Driver or related Microprogramming function call will result in an update to this frame. The Statistics Frame can display either global statistics, which is the cumulative statistics from the last statistics reset operation, or individual function statistics, which are only for the current function.

GLOBAL STATISTICS:	I/O Ops:	# of I/O Operations
	TGT Chks:	# of Target Check Conditions (errors)
	INT D Er:	# of Initiator-Detected Errors
	Bytes Wr:	# of Bytes Written
	Bytes Rd:	# of Bytes Read
	Bytes Cp:	# of Bytes Compared
	Cmp Ers.:	# of Compare Errors
FUNCTION STATISTICS:	TGT Stat:	Target Status Byte
	INIT Stat:	Initiator Status Byte
	I/O Stat:	I/O Driver Status Byte
	Bytes Wr:	# of Bytes Written
	Bytes Rd:	# of Bytes Read
	Bytes Cp:	# of Bytes Compared
	Cmp Ers.:	# of Compare Errors

DEBUG.2.1.3.2 USER COUNTERS FRAME

The User-Defined Counters provided in this frame will be updated by direct function calls from the Test Function Library.

```
USER COUNTERS:    uc0:  16-bit count
                  uc1:  16-bit count
```

uc0 or uc1 is the User Counter String defining the counter as set by the **ucname()** function. The user is also capable of incrementing and resetting these counters through the SAT or MENU via Test Function Library functions.

DEBUG.2.1.3.3 BUFFER FRAME

The buffer frame defines the current read and write buffer and their current DMA address. This frame is only updated when the Buffer/Command Update Flag is set by the **bcu(1)** function.

```
Wr Buf: BPM
        0343
Rd Buf: BPM
        01ff
```

DEBUG.2.1.3.4 SCSI COMMAND FRAME

This frame shows the current SCSI command and I/O Driver parameters issued to the I/O Driver or the CDB generated by a **cdbnn1()**, **cdbnn2()** or **cdbnn3()** Microprogramming function. In addition, the returning status and **sense()** information is displayed. The Debugger BCU(1) command or **bcu(1)** function is required for auto-update of this frame.

I/O COMMAND PARAMETERS:

CDB (SCSI Command):	up to 12 bytes
Status (SCSI Status):	1 byte
(previous SCSI Status):	(1 byte)
Sense (SCSI Sense):	up to 20 bytes, if more than 20 a "+" will appear after the 20th byte
xfer (data transfer mode):	HSHCV,HSHC,HSSC,HSRW,HSCOPY, DMAHC,DMASC,DMARW,DMACOPY, TRSC, TRWR, PIORW or PIOSC
a.s. (Autosense):	ON or OFF
s.l. (Bus State Logging):	ON or OFF
arb. (Arbitration Mode):	NONE, HDW or SFTW
sel. (Select Mode):	DUMB or SMART
b.p. (Bus Parity):	ON or OFF
b.w. (Busywait):	ON or OFF
iid (Initiator ID):	0 -> 7
tid (Target ID):	0 -> 7

The **xfer** through **tid** flags are not affected by any I/O Driver or Microprogramming functions. These are updated only when the functions that set them are called (i.e., **xfermode()**, **autosense()**, **bus_logen()**, ...).

DEBUG.2.1.4 TRACE DISPLAY SCROLLING WINDOW

The Trace Scrolling Window provides the user with a step-by-step execution history of Test and Documentation Functions. It appears at the lower half of the screen; this window is displayed in debug level 2 and 3 (refer to Figures DEBUG-F4 and DEBUG-F5). The following convention is used within this window:

Reverse Video: Function pending execution

Half Intensity: Function which did not execute (skipped)

Full Intensity: Function currently or previously executed

DEBUG.2.1.5 DEBUGGER COMMAND LINE

The Debug Command Line provides the user with various Debug command options. This line is displayed when the Debugger is active if the debug display level is greater than Ø or if the **debug()** function is encountered in SAT code execution. This debug control line appears on line 24 of the screen. The functions provided by the debugger are defined in Section DEBUG.3.

DEBUG.2.2 SECONDARY DISPLAY SCREEN

The Secondary Display Screen provides a means of displaying the Read/Write, Sense, OBB or State Log buffer. The Debugger saves the Primary display screen and replaces it with the Secondary screen. The primary screen is restored after the secondary screen is no longer required by the user.

DEBUG.2.2.1 BUFFER DISPLAY

DEBUG.2.2.1.1 DATA BUFFER DISPLAY

By specifying the buffer type, starting address and length of the buffer to be displayed, the data buffer display will appear on the secondary display screen. The display may also be grouped by bytes or words. For an example of the data buffer display, refer to Figures DEBUG-F6 and DEBUG-F7 .

FIGURE ~DEBUG-F6. DATA BUFFER DISPLAY WITH BYTE GROUPING

```
Dbuf( Buf:   W; Strt: 0000; Len: 0100; On: D  B Grouping)
```

```
TRACE : Dbuf(Buf:   W; Strt: 0000; Len: 0100; )  01-08-86 11:56:57

  0000   12 33 21 01 23 32 10 12 33 21 01 23 32 10 12 33
  0010   21 01 23 32 10 12 33 21 01 23 32 10 12 33 21 01
  0020   23 32 10 12 33 21 01 23 32 10 12 33 21 01 23 32
  0030   10 12 33 21 01 23 32 10 12 33 21 01 23 32 10 12
  0040   33 21 01 23 32 10 12 33 21 01 23 32 10 12 33 21
  0050   01 23 32 10 12 33 21 01 23 32 10 12 33 21 01 23
  0060   32 10 12 33 21 01 23 32 10 12 33 21 01 23 32 10
  0070   12 33 21 01 23 32 10 12 33 21 01 23 32 10 12 33
  0080   21 01 23 32 10 12 33 21 01 23 32 10 12 33 21 01
  0090   23 32 10 12 33 21 01 23 32 10 12 33 21 01 23 32
  00A0   10 12 33 21 01 23 32 10 12 33 21 01 23 32 10 12
  00B0   33 21 01 23 32 10 12 33 21 01 23 32 10 12 33 21
  00C0   01 23 32 10 12 33 21 01 23 32 10 12 33 21 01 23
  00D0   32 10 12 33 21 01 23 32 10 12 33 21 01 23 32 10
  00E0   12 33 21 01 23 32 10 12 33 21 01 23 32 10 12 33
  00F0   21 01 23 32 10 12 33 21 01 23 32 10 12 33 21 01
```

FIGURE ~DEBUG-F7. DATA BUFFER DISPLAY WITH WORD GROUPING

```
Dbuf( Buf:   W; Strt: 0000; Len: 0100; On: D  W Grouping)
```

```
TRACE : Dbuf(Buf:   W; Strt: 0000; Len: 0100; )  01-08-86 11:58:16

  0000   1233 2101 2332 1012 3321 0123 3210 1233
  0010   2101 2332 1012 3321 0123 3210 1233 2101
  0020   2332 1012 3321 0123 3210 1233 2101 2332
  0030   1012 3321 0123 3210 1233 2101 2332 1012
  0040   3321 0123 3210 1233 2101 2332 1012 3321
  0050   0123 3210 1233 2101 2332 1012 3321 0123
  0060   3210 1233 2101 2332 1012 3321 0123 3210
  0070   1233 2101 2332 1012 3321 0123 3210 1233
  0080   2101 2332 1012 3321 0123 3210 1233 2101
  0090   2332 1012 3321 0123 3210 1233 2101 2332
  00A0   1012 3321 0123 3210 1233 2101 2332 1012
  00B0   3321 0123 3210 1233 2101 2332 1012 3321
  00C0   0123 3210 1233 2101 2332 1012 3321 0123
  00D0   3210 1233 2101 2332 1012 3321 0123 3210
  00E0   1233 2101 2332 1012 3321 0123 3210 1233
  00F0   2101 2332 1012 3321 0123 3210 1233 2101
```

DEBUG.2.2.1.2 STATE LOGGING DISPLAY

The SDS-1 State Log file provides the user with a detailed accounting of all SCSI Bus transactions performed by the SDS-1. Each entry in the log represents a bus transaction. Some of the entries have been compacted into one or two log line(s), such as the Data_in, Data_out or Cmd_out entries. A few state log sample entries are listed in Table DEBUG-T3 . The numbers at the right of the entry indicate the log entry line number. The numbers at the left are a time stamp logged in seconds with a resolution of 50 usec. Refer to Figure DEBUG-F8 for an example of an actual state logging display.

TABLE DEBUG-T3. SAMPLE STATE LOG ENTRIES IN STATE LOG BUFFER

Start Time	Event Description	End Time	Line #
0000.00000		-blank-	01B
0000.00000	----------------- Test Initialization -----------------		01A
ssss.mmmuu	Arbitration as XX (incomplete)		019
ssss.mmmuu	Arbitration as XX (lost)	ssss.mmmuu	018
ssss.mmmuu	Bus Free Detected		017
ssss.mmmuu	Arbitration as XX	ssss.mmmuu	016
ssss.mmmuu	Selection ids = (bbbb bbbb)	ssss.mmmuu	015
ssss.mmmuu	Message out mm		014
ssss.mmmuu	Cmd_out cc cc cc cc cc cc		013
ssss.mmmuu	cc cc cc cc cc cc	ssss.mmmuu	
ssss.mmmuu	Data out nnnnnnnnH byte(s)		012
ssss.mmmuu	Status in ss		011
ssss.mmmuu	Message in mm		010
ssss.mmmuu	Bus Free Detected		00F
ssss.mmmuu	Selection ids = (bbbb bbbb) *timed out*		00E
ssss.mmmuu	Bus Reset Detected		00D
ssss.mmmuu	Selection ids = (bbbb bbbb)	ssss.mmmuu	00C
ssss.mmmuu	Cmd_out cc cc cc cc cc cc	ssss.mmmuu	00B
ssss.mmmuu	Message in mm		00A
ssss.mmmuu	Message in mm		009
ssss.mmmuu	Bus Free Detected		008
ssss.mmmuu	Reselection ids = (bbbb bbbb)		007
ssss.mmmuu	Message in mm		006
ssss.mmmuu	Message in mm		005
ssss.mmmuu	Data in nnnnnnnnH byte(s)		004
ssss.mmmuu	Status in ss		003
ssss.mmmuu	Message in mm		002
ssss.mmmuu	Bus Free Detected		001
ssss.mmmuu	Bus Reset Asserted		000

Some entry lines will have a ">" at the far right to indicate that there is more information in the log than currently displayed.

The most recent log entry will have a log entry number of 000, which is the last entry in the state log buffer. When displaying the most recent log entry, use a starting address of 0. This

buffer can be displayed through the Debugger (Dbuf command) or by using the **dispbuf()** or **rptbuf()** function. The maximum number of entries is 400 hex. The state log buffer is a FIFO where the oldest entry is at the top and the most recent is at the bottom. Once the maximum number of entries has been reached, the oldest entries will be deleted at the top and the latest log entry is entered at the bottom. As entries are entered, they are moved up in the buffer.

FIGURE ~DEBUG-F8. STATE LOG DISPLAY EXAMPLE

```
Dbuf( Buf:   W; Strt: 0000; Len: 0020; On: D  B Grouping)

rptbuf(L,    0,   10)                                  07-09-86 09:17:29

 Start Time                 Event Description                 End Time    Line #
+-----------+-----------------------------------------------+-----------+-------
 0083.62869           Message in 00                                        00F
 0083.62965  Bus Free Detected                                             00E
 0086.62319     Arbitration as 07                             0086.62346   00D
 0086.62441        Selection ids = (1001 0000b)               0086.62489   00C
 0086.62521           Message out C0                                       00B
 0086.62600              Command out 08 00 00 00 80 00        0086.62849   00A
 0086.62886           Message in 02                                        009
 0086.62987           Message in 04                                        008
 0086.63077  Bus Free Detected                                             007
 0086.65187        Reselection  ids = (1001 0000)                          006
 0086.65203           Message in 80                                        005
 0086.65287           Message in 03                                        004
 0086.65664                 Data In      8000H byte(s)                     003
 0086.89434              Status in 00                                      002
 0086.89508           Message in 00                                        001
 0086.89604  Bus Free Detected                                             000
```

~DEBUG.2.3 DEBUGGER DISPLAY/EXECUTION SPEED

The **bcu()** (Buffer/Command Frame Update), **statsen()** and user counter functions update the screen information--the more updates to the screen, the slower the execution. Table DEBUG-T4 defines the overall system execution speed based on the amount of screen update required. Also refer to Figure DEBUG-F9 for a diagram of where screen updates are performed. The Debugger states **TRACE**, **IOINIT**, etc.) will be discussed in detail in later sections.

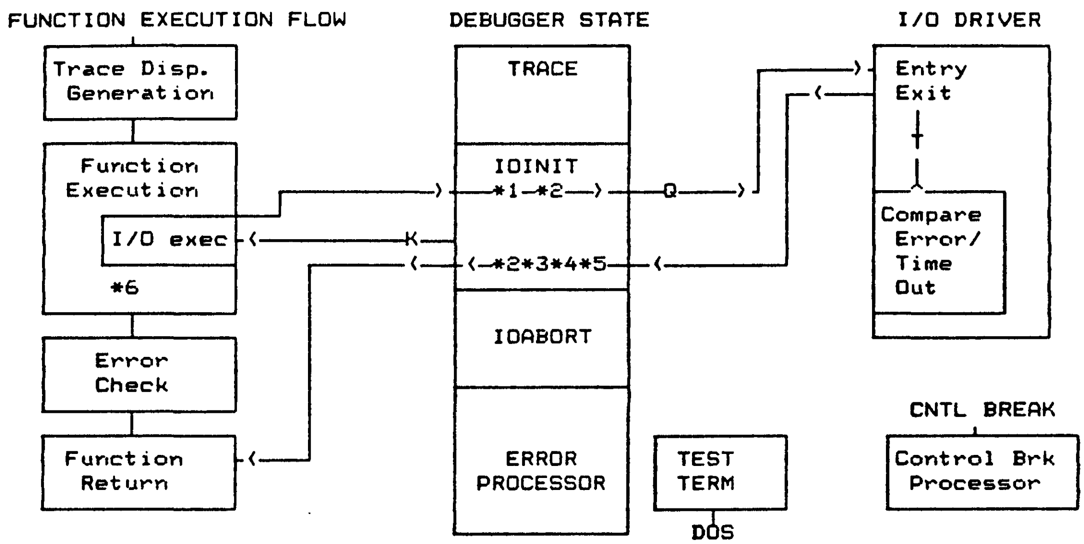
The I/O Driver control flags (data transfer mode, autosense, state logging, arbitration mode, etc.) effect the I/O Driver execution and are defined in the I/O DRIVER Section.

TABLE ~DEBUG-T4. DEBUGGER DISPLAY LEVELS (STATS GATHERING ON)

Debug Level	BCU	Display Level and Functions	Real Time I/O Driver Execution Speed Notes
Ø Debug Off	X	T.D. Fixed Window T.D. Scrolling Window	FASTEST EXECUTION SPEED: No screen updates during function or I/O Driver execution
1	Ø	T.D. Fixed Window Status Fixed Window Real Time Update: Statistics Frame T.D. Scrolling Window	SECOND FASTEST: Limited number of Fixed Window updates during function and I/O Driver execution
2	Ø	T.D. Fixed Window Status Fixed Window Real Time Update: Statistics Frame Trace Scrolling Window	SLOW: Trace Scrolling Window and limited number of Fixed Window updates during function and I/O Driver execution
3	Ø	Status Fixed Window Real Time Update: Statistics Frame Trace Scrolling Window	SLOW: Same as level 2 with BCU equal to Ø
1	1	T.D. Fixed Window Status Fixed Window Real Time Updates: Statistics Frame Buffer Frame SCSI Command Frame T.D. Scrolling Window	MEDIUM: Maximum number of Fixed Window updates during function and I/O Driver execution
2	1	T.D. Fixed Window Status Fixed Window Real Time Updates: Statistics Frame Buffer Frame SCSI Command Frame Trace Scrolling Window	SLOWEST: Trace Scrolling Window and maximum number of Fixed Window updates during and I/O Driver execution
3	1	Status Fixed Window Real Time Updates: Statistics Frame Buffer Frame SCSI Command Frame Trace Scrolling Window	SLOWEST: Same as level 2 with BCU equal Ø

T.D. = Test Documentation

FIGURE ~DEBUG-F9. SCREEN UPDATE LOGIC



*1 = Command Update
*2 = Buffer Frame Update
*3 = Status Update
*4 = Sense Update if automatic sense enabled and check condition
*5 = Statistics Update
*6 = User counter functions update

The above figure illustrates where and when the Debugger updates to the screen. Command and Buffer Frame updates occur prior to the execution of the I/O Driver Command. Buffer Frame, Status, Sense and Statistic updates occur after the execution of the I/O Driver.

The User Counter functions (**ucname()**, **ucinc()** and **ucrst()**) update the screen directly from their function.

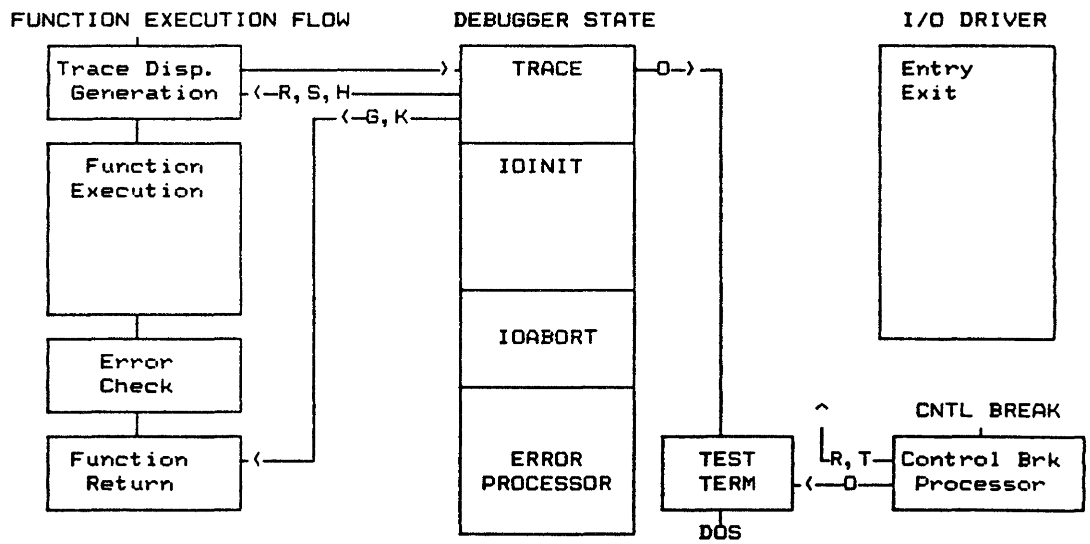
~DEBUG.3 DEBUG STATES/COMMANDS

The Debugger commands available are determined by the debug state. These current debug states are determined by the test function being executed and their execution condition (i.e. execution phase and error status).

~DEBUG.3.1 TRACE STATE

Figure DEBUG-F1Ø shows the relationship between the test function and the Debugger **TRACE** state. For debug level greater than zero, the test function enters the Debugger **TRACE** state to display the function name and it arguments (right arrow going into the left side of the **TRACE** box in Figure DEBUG-F1Ø). Within the **TRACE** state, the user has several program flow options (or commands). These commands are shown in the lines leaving the **TRACE** box. In the **TRACE** mode, the **R**un, **S**tep or **H**alf-Step commands return to execute the test function. With the Goto and S**k**ip commands, the test function is not executed. And with the DOS Return command, the SAT will exit back to DOS. When executing in the SDS-1 Menu system, the **ESC** key is used to leave the **TRACE** state and return to the Menu.

FIGURE ~DEBUG-F1Ø. TRACE STATE EXECUTION/DEBUGGER FLOW



Other **TRACE** state commands available at this point are summarized in Table DEBUG-T5. The space bar will toggle the **TRACE** command line through its command set. As long as the mode supports the command, it is not necessary to have it displayed on the command line in order to execute it.

TABLE ~DEBUG-T5. TRACE STATE COMMAND SET

Command	Function
A	statistics reset (reset a single or all global stats)
B	break point (set a program execution break point)
C	buffer/command update (update Buffer/Command frames)
D	load buffer (load current fill buffer from disk)
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)
E	eea() function (change explicit error action)
F	display buffer
0	display buffer type (Read, Write, Log, Sense, OBB)
1	display buffer starting address
2	display buffer length
3	display buffer output device
4	display buffer by bytes or words
G	goto function (skipping other functions)
H	half-step function (stop prior to SCSI execution)
I	iea() function (implicit error action)
K	skip function (do not execute this function)
L	debug display level
M	modify current fill buffer
5	modification address
N	sense
O	return to DOS
P	SCSI display (sample SCSI bus and display)
R	run mode (start execution and do not stop in TRACE)
S	step mode (execute function and stop in next TRACE)
T	SCSI bus reset
U	user counter reset
V	save buffer
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)

DEBUG.3.1.1 DETAILED DESCRIPTIONS OF TRACE COMMANDS

The following section describes each of the **TRACE** menu line options.

DEBUG.3.1.1.1 FLOW CONTROL (TRACE:Flow)

These are the flow control commands of the **TRACE** mode:

Goto (~GOTO):
Goes to the defined function skipping all functions between the current point and the GOTO function name. The GOTO function name will look for a match in the test program; for instance, if the function name **"read"** was entered, GOTO will skip functions until it finds a function with its first four characters matching **"read."** Some possible function names that GOTO will stop at are **readr, reads, readcap,** etc. With a **"writes(10)"** function name, the GOTO function will look for the exact match. Maximum function name length is 20. Count is the occurrence number, 1 indicates the first occurrence, 2 indicates the second and so on. The following prompts will appear:
Enter Function Name >
Count >

Breakpoint Set (~BREAK-POINT):
Sets Break Point function and occurrence count. After break point is set and **R**un is executed, all functions up to the B.P. function will be executed. When the B.P. function name is reached, execution is halted and user has control at this point. The B.P. Function Name matches function names exactly like the GOTO function. And Count is the occurrence number. The following prompts will appear:
B.P. Function: >
Count: >

Run (RUN MODE):
Begins continuous execution of the test program and does not halt unless a Break Point has been reached.

Step (SINGLE-STEP MODE):
Executes current command and halt in **TRACE** state on the next RTFL function.

Half-Step (HALF-STEP):
Set up the I/O Driver command but halt before I/O Driver execution. Half-Step enters the **IOINIT** state.

S**k**ip (SKIP FUNCTION):
Does not execute current command.

Exit to **D**OS (DOS RETURN):
Terminates current Test.

DEBUG.3.1.1.2 BUFFER FUNCTIONS (TRACE:Buffer)

In **TRACE**, the user can perform the following buffer-related commands:

a) display a buffer
b) save a buffer to disk
c) load a buffer from disk
d) modify a buffer

DBuf (DISPLAY BUFFER):
Before displaying the buffer, check if the following have been initialized properly:

- Ø: Buffer reference (R, W, RW, Log, Sense, OBB)
- 1: Starting Address (in hex)
- 2: Length (in hex)
- 3: Display On: (D=display, P=printer, L=log)
- 4: With Grouping of: (B=bytes, W=words)

Load or Save (LOAD OR SAVE BUFFER):
Before loading or saving buffer, check if the following values are initialized:

- 5: Starting Address of Buffer to Load or Save
- 6: Length to Load or Save
- 9: File name to Load or Save

Mod (MODIFY BUFFER):
The following should be specified before the byte can be modified:

- 5: Starting Address of Buffer to Modify

DEBUG.3.1.1.3 ERROR ACTION/RECOVERY (TRACE:EA/Rec)

These are the error action and recovery commands in the **TRACE** state:

IEA (IMPLICIT ERROR ACTION):

```
IEA(LOGC):  Log Error and Continue
   (LOGH):  Log Error and Halt and Enter Debugger
   (CONT):  Continue (ignore error)
   (HALT):  Halt on Error and Enter Debugger
```

EEA (EXPLICIT ERROR ACTION):

```
EEA(LOGC):  Log Error and Continue
   (LOGH):  Log Error and Halt and Enter Debugger
   (CONT):  Continue (ignore error)
   (HALT):  Halt on Error and Enter Debugger
```

Sense (SENSE COMMAND):
Generate a Request Sense Command to I/O Driver (does not show on trace display or modify current I/O Driver command). The target, initiator, and I/O status are also displayed at the bottom of the screen. The initiator and I/O status codes are discussed in the I/O Driver section.

SCSI **Reset** (RESET SCSI BUS/I/O DRIVER)

SCSI Display (SCSI BUS DISPLAY):
The SCSI bus display will appear as follows:

BSY SEL data: 0000 0000 (00) **REQ** ACK c/D i/O MSG RES (p=sample)

The full-intensity values indicate the asserted state of the SCSI bus at the time of request.

DEBUG.3.1.1.4 DEBUGGER CONTROL (TRACE:Control)

These are four debugger control commands available in the **TRACE** state:

Debug Level (DEBUG LEVEL):
Set Debug Display Level (Change Primary Display Format) to 0 through 3. Level 0 will disable the Debugger and commence program execution at full speed. (The Debugger cannot be reinvoked.)

BCU() (BUFFER COMMAND UPDATE FRAME FLAG):
BCU(0): Buffer Command Update Not Set
BCU(1): Buffer Command Update Set

User Cntr Reset (RESET USER COUNTERS):
Reset User Counter 0 or 1.

Stats Reset (STATISTICS RESET):
Reset all or a single global statistics counter.

DEBUG.3.2 IOINIT STATE

If the current or pending function is an I/O Driver command, a Half-Step command in the **TRACE** state will cause function execution to stop in the Debugger **IOINIT** state (note the state name change on the Debugger command line when state changes). In the **IOINIT** state, the user can view the SCSI Command Descriptor Bytes (CDB) in the SCSI Command Frame and check the buffer pointers prior to any execution on the SCSI bus. There are two flow commands that can be executed from this state, **Skip** or Exequte. The **Skip** command will not execute the CDB, while the **Exequte** command will call the I/O Driver for execution of command and return to the test function. Commands other than **Half-Step** that will cause execution of the I/O commands will bypass the **IOINIT** mode and go straight to the I/O Driver (illustrated by the broken lines through the **IOINIT** box in Figure DEBUG-F11).

The Debugger options (commands) available in the **IOINIT** state are listed in Table DEBUG-T6. The space bar will toggle the **IOINIT** command line through its command set. As long as the mode supports the command, it is not necessary to have it displayed on the command line in order to execute it.

FIGURE ~DEBUG-F11. IOINIT STATE EXECUTION/DEBUGGER FLOW

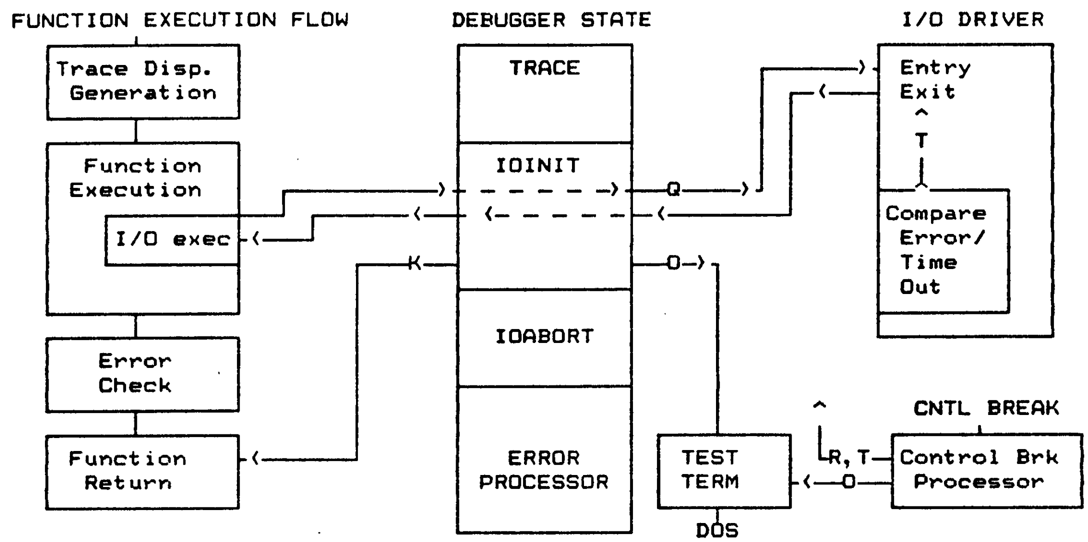


TABLE ~DEBUG-T6. IOINIT STATE COMMANDS

Command	Function
D	load buffer (load current fill buffer from disk)
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)
F	display buffer
0	display buffer type (Read, Write, Log, Sense, OBB)
1	display buffer starting address
2	display buffer length
3	display buffer output device
4	display buffer by bytes or words
K	skip function
M	modify buffer
5	address to modify
Q	execute function
V	save buffer
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)

DEBUG.3.2.1 DETAILED DESCRIPTIONS OF IOINIT COMMANDS

DEBUG.3.2.1.1 FLOW CONTROL (IOINIT:Flow)

These are the flow control commands for the **IOINIT** mode:

Skip (SKIP FUNCTION):
Does not execute current I/O command.

Exequte (EXECUTE FUNCTION):
Execute the current I/O command.

DEBUG.3.2.1.2 BUFFER FUNCTIONS (IOINIT:Buffer)

These are the buffer function commands for the **IOINIT** mode:

DBuf (DISPLAY BUFFER):
Before displaying buffer, check if the following are initialized properly:

- Ø: Buffer number or reference (R, W, RW, L, S, OBB)
- 1: Starting Address (in hex)
- 2: Length (in hex)
- 3: Display On: (D=display, P=printer, L=log)
- 4: With Grouping of: (B=bytes, W=words)

Load or Save (LOAD OR SAVE BUFFER):
Before loading or saving buffer, check if the following values are initialized:

- 5: Starting Address of Buffer to Load or Save
- 6: Length to Load or Save
- 9: File name to Load or Save

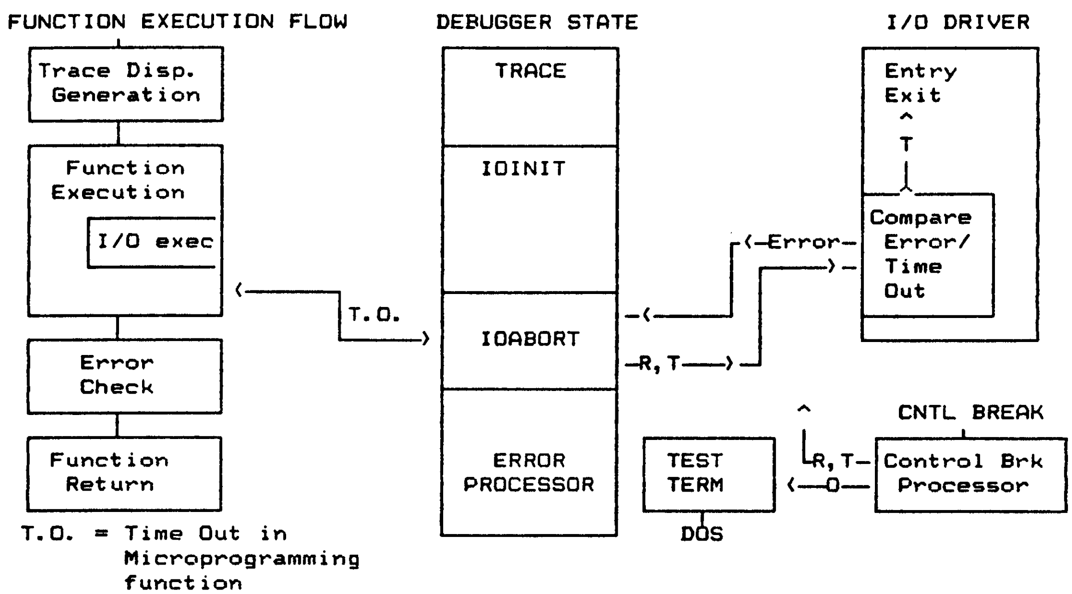
Mod (MODIFY BUFFER):
The following should be specified before the byte can be modified:

- 5: Starting Address of Buffer to Modify

DEBUG.3.3 IOABRT STATE

The **IOABRT** state is intended to provide the user with a window inside the I/O Driver or Microprogramming function (refer to Figure DEBUG-F12). This window provides the user a means to handle time-out conditions and data miscompares.

FIGURE ~DEBUG-F12. IOABRT STATE EXECUTION/DEBUGGER FLOW



The flow control options available on a compare error are resume and halt on (next) compare error (**HOCE(1)**), or resume with no further error display (**HOCE(Ø)**). For a time-out, the user can resume the operation with a secondary time value. Or for either type of condition, the user can terminate the operation. Further details on **IOABRT** commands can be found in Table DEBUG-T7.

The space bar will toggle the **IOABRT** command line through the command set. As long as the mode supports the command, it is not necessary to have it displayed on the command line in order to execute it.

TABLE ~DEBUG-T7. IOABRT STATE COMMAND SET

Command	Function
D	load buffer (load current fill buffer from disk)
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)
F *1	display buffer
0	display buffer type (Read, Write, Log, Sense, OBB)
1	display buffer starting address
2	display buffer length
3	display buffer output device
4	display buffer by bytes or words
M	modify buffer
5	address to modify
O	return to DOS
P	SCSI display
R	resume
7	secondary time out value set
8	halt on compare error flag toggle
T	I/O termination
V	save buffer
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)

*1 The Write Buffer Display Command is not allowed for any operations which utilize the On-Board Buffer (HSHCV, HSHC, HSSC or HSRW).

DEBUG.3.3.1 DETAILED DESCRIPTIONS OF IOABRT COMMANDS

DEBUG.3.3.1.1 FLOW CONTROL (IOABRT:Flow)

These are the flow control commands for the **IOABRT** mode:

Resume (RESUME I/O WITH ~SECONDARY-TIMEOUT OR HALT ON ERROR):

- 7: TO: Change secondary time-out value. After the first time-out, the default is 30 seconds.
- 8: HOCE(): Halt-On-Compare Error
  HOCE(0): Halt-On-Compare Error Not Set
  HOCE(1): Halt-On-Compare Error Set

I/O **T**ermination (RESET AND TERMINATE I/O)

DOS Ret (RETURN TO DOS)

DEBUG.3.3.1.2 BUFFER FUNCTIONS (IOABRT:Buffer)

These are the buffer function commands for the **IOABRT** mode:

DBuf (DISPLAY BUFFER):
(Not valid for OBB operations). Before displaying buffer, check if the following are initialized properly for the display:

- Ø: Buffer number or reference (R, W, RW, L, S, OBB)
- 1: Starting Address (in hex)
- 2: Length (in hex)
- 3: Display On: (D=display, P=printer, L=log)
- 4: With Grouping of: (B=bytes, W=words)

Load or Save (LOAD OR SAVE BUFFER):
Before loading or saving buffer, check if the following values are initialized:

- 5: Starting Address of Buffer to Load or Save
- 6: Length to Load or Save
- 9: File name to Load or Save

Mod (MODIFY BUFFER):
The following should be specified before the byte can be modified:

- 5: Starting Address of Buffer to Modify

DEBUG.3.3.1.3 ERROR ACTION/RECOVERY (IOABRT:EA/Rec)

There is one error action and recovery command in the **IOABRT** state:

SCSI Display (DISPLAY SCSI BUS):
The SCSI bus display will appear as below:

BSY SEL data: ØØØØ ØØØØ (ØØ) **REQ** ACK c/D i/O MSG RES (p=sample)

The full-intensity values indicate the current state of the SCSI bus at the time of request.

DEBUG.3.4 ERROR PROCESSOR STATES

The **ERROR PROCESSOR** state is called from the error checking logic contained in each SDS-1 library function. Certain types of errors are classed either as implicit errors (such as data compare errors) or explicit errors (such as an incorrect expected status). Implicit errors do not require a test on the user's part. Explicit errors, such as checking the sense data information bytes, are explicitly performed by the user. Both types of errors are processed by the **ERROR PROCESSOR** but can have different error actions. Figure DEBUG-T1 displays the execution flow.

The **ERROR PROCESSOR** is responsible for reporting both **IMP ER** (implicit errors) and **EXP ER** (explicit errors). The commands available in this state are described in Table DEBUG-T8.

FIGURE ~DEBUG-F13. ERROR PROCESSOR STATES EXECUTION/DEBUGGER FLOW

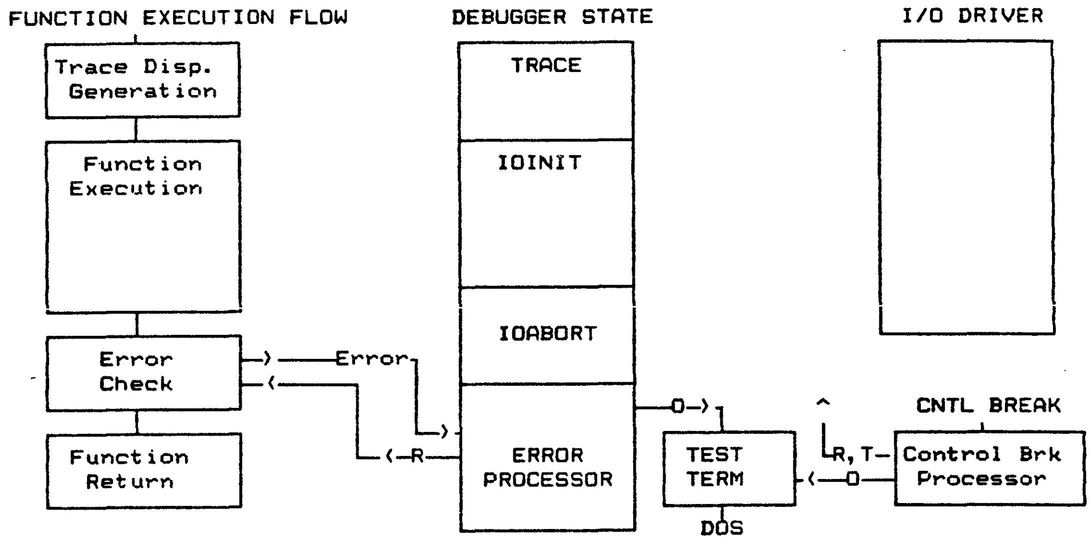


TABLE ~DEBUG-T8. ERROR PROCESSOR COMMAND SET

Command	Function
D	load buffer (load current fill buffer from disk)
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)
E	eea() function (explicit error action)
F	display buffer
0	display buffer type (Read, Write, Log, Sense, OBB)
1	display buffer starting address
2	display buffer length
3	display buffer output device
4	display buffer by bytes or words
I	iea() function (implicit error action)
M	modify buffer
5	address to modify
O	return to DOS
P	SCSI display
R	resume
T	SCSI bus reset
V	save buffer
5	buffer address (start address for load/save)
6	buffer length (length of load/save operation)
9	load file name (file name for load/save)

DEBUG.3.4.1 DETAILED DESCRIPTIONS OF ERROR PROCESSOR COMMANDS

The **ERROR PROCESSOR** state is reached after completion of the I/O command with implicit error action set to Log and Halt (**iea(LOGH)**) or Halt (**iea(HALT)**), or explicit error action is set to Log and Halt (**eea(LOGH)**) or Halt (**eea(HALT)**).

DEBUG.3.4.1.1 FLOW CONTROL

These are the flow control commands for the **ERROR PROCESSOR** mode:

Resume (CONTINUE WITH NEXT FUNCTION)

DOS Ret (RETURN TO DOS)

DEBUG.3.4.1.2 BUFFER FUNCTIONS

These are the buffer function commands for the **ERROR PROCESSOR** state:

DBu**f** (DISPLAY BUFFER):
Before displaying buffer, check if the following are initialized properly for the display:

- Ø: Buffer number or reference (R, W, RW, L, S, OBB)
- 1: Starting Address (in hex)
- 2: Length (in hex)
- 3: Display On: (D=display, P=printer, L=log)
- 4: With Grouping of: (B=bytes, W=words)

Loa**d** or Save (LOAD OR SAVE BUFFER):
Before loading or saving buffer, check if the following values are initialized:

- 5: Starting Address of Buffer to Load or Save
- 6: Length to Load or Save
- 9: File name to Load or Save

Mod (MODIFY BUFFER):
The following should be specified before the byte can be modified:

- 5: Starting Address of Buffer to Modify

DEBUG.3.4.1.3 ERROR ACTION/RECOVERY

These are the error action and recovery commands for the **ERROR PROCESSOR** mode:

IEA (IMPLICIT ERROR ACTION):
IEA(LOGC): Log Error and Continue
(LOGH): Log Error and Halt and Enter Debugger
(CONT): Continue (ignore error)
(HALT): Halt on Error and Enter Debugger

EEA (EXPLICIT ERROR ACTION):
EEA(LOGC): Log Error and Continue
(LOGH): Log Error and Halt and Enter Debugger
(CONT): Continue (ignore error)
(HALT): Halt on Error and Enter Debugger

Sense (SENSE COMMAND):
Generates a Request Sense Command to I/O Driver (does not show on trace display or modify current I/O Driver command). The target, initiator, and I/O status are also displayed at the bottom of the screen. The initiator and I/O status codes are listed in Figure IODVR-T4 and IODVR-T5.

SCSI **Reset** (RESET SCSI BUS/I/O DRIVER)

SCSI Display (SCSI BUS DISPLAY):
The SCSI bus display will appear as below:

BSY SEL data: 0000 0000 (00) **REQ** ACK c/D i/O MSG RES (p=sample)

The full-intensity values indicate the current state of the SCSI bus at the time of request.

~DEBUG.3.5 SAT EXECUTION HALT/INTERRUPTION

In addition to setting the error action, **iea()** and **eea()** functions (or **IEA** or **EEA** commands) to halt on error (as described in section DEBUG.3.4.1), there are other ways to halt or interrupt SAT execution.

DEBUG.3.5.1 NORMAL END OF SAT PROGRAM

To exit from the Debugger at any level, the completion of the SAT program will return back to DOS.

DEBUG.3.5.2 ESCAPE KEY

The **ESC** key will be used to halt Debugger execution (in display levels 1,2 or 3).

If the Debug Level is greater than 0, the **ESC** key can be used to stop execution of the SAT program. The user can stop execution in the **TRACE** state of the next library function with the next function pending execution (indicated in reverse video).

DEBUG.3.5.3 CONTROL-BREAK KEYS

See Section DEBUG.4.2.

~DEBUG.4.0 MISCELLANEOUS DEBUGGER FUNCTIONS

~DEBUG.4.1 DOS RETURN

Exits the current SAT program or Menu Interface session and returns to DOS.

~DEBUG.4.2 CONTROL-BREAK

The **Control-Break** key sequence can be used to interrupt execution of the SAT program, at which point the user has control of the execution to either resume execution or return to DOS or if a function is being timed (that is, if the **ioto()** function is valid for the currently executing function), the user can force an early time-out with the I/O **T**ermination command.

When the **Control-Break** keys are pressed during an execution of a time-out associated function, the options presented are: resume execution, force a time-out, exit to DOS or display the SCSI bus. Otherwise, the options are: resume execution, exit to DOS or display SCSI bus.

~DEBUG.4.3 BUFFER MODIFICATION

Buffers may be modified by using the **M** command in any of the Debugger states. The following should be specified before the byte can be modified:

Starting Address of Buffer to Modify (key 5)

~DEBUG.4.4 BUFFER SAVE/LOAD

Before loading or saving buffer, check if the following values are initialized:

Starting Address of Buffer to Load or Save (key 5)
Length to Load or Save (key 6)
File name to Load or Save (key 9)

~DEBUG.4.5 DISPLAY SCSI BUS

The SCSI bus display will appear as below:

BSY SEL data: 0000 0000 (00) **REQ** ACK c/D i/O MSG RES (p=sample)

The full-intensity values indicate asserted bits on the SCSI bus at the time of bus sample. **P** is used to resample the bus.

(THIS PAGE INTENTIONALLY LEFT BLANK)

~FLIB.Ø FUNCTION LIBRARY OVERVIEW

~FLIB.1 INTRODUCTION

This section is intended to give an overview of the functions listed in Appendix A: Function Library. For a complete description of each function, see Appendix A where they are listed alphabetically.

The library functions are grouped together by categories (Type, Class, and Function).

The Types are: Setup, Execution and Data Analysis/Reduction Test Functions. Within each Type are the Classes: Generic, I/O Driver and Microprogramming. Some of these Classes have been further grouped by their Functions.

There is also an additional Type which is the Report Documentation Functions. These functions are basically used for report generation of the Test Results Report.

~FLIB.2 SETUP TEST FUNCTIONS

These setup test functions are used to initialize conditions. They can either be specific, I/O Driver or Microprogramming or generic functions.

FLIB.2.1 GENERIC CLASS

FLIB.2.1.1 CONFIGURATION SETUP

Function Name	Use
ackdelay	OBB Acknowledge Delay
errdelay	Enable/Disable Five Second Error Message Delay
line_mode	Select Single-Ended or Differential SCSI mode
parity	Enable/Disable SCSI Bus Parity
reset	Reset SCSI Bus/I/O Driver
set_er_limits	Set Error Limits
xfermode	Open R/W Buffer/Set Transfer Mode

These functions are used to set up the initial conditions for either an I/O Driver or Microprogramming Execution Test.

FLIB.2.1.2 BUFFER SETUP

Function Name	Use
dmarst	Reset DMA Pointer
dmaset	Set DMA Pointer
dmaset_va	Set the Virtual DMA Address
dmaset_vblk	Set Virtual DMA Address for the Defined Block
fillbcb	Byte Block Count Fill
fillbcw	Word Block Count Fill
fillbyte	Fill Buffer with Byte
filld	Decrement Count Fill
filli	Increment Count Fill
fillk	Constant Fill
fillpr	Pseudo Random Fill
loadbuf	Load Buffer from Disk
overbcb	Overlay Block Count Byte
overbcdw	Overlay Block Count Double Word
overbcw	Overlay Block Count Word
put_byte	Put Buffer with Data Byte
savebuf	Save Fill Buffer to Disk
setbuf	Fill Buffer with ASCII String
setfill_buf	Set Current Fill Buffer

These routines are useful for buffer initializations. The **xfermode()** function will automatically open buffers and the next **xfermode()** or end of SAT will close them.

Buffers may be filled more than once during the execution of test programs. For instance, a buffer may be filled with a pseudo random pattern (**fillpr()**) and then filled with the overlay block count word (**overbcw()**) to check if the buffer was filled properly. The **savebuf()** function will allow the user to save a buffer to the SDS-1 internal disk and the **loadbuf()** allows the user to load a buffer from the SDS-1 internal disk.

FLIB.2.1.3 ERROR ACTION/RECOVERY SETUP

Function Name	Use
eea	Explicit Error Action
iea	Implicit Error Action

These routines specify the error action to be taken if an implicit or explicit error occurs. There are several actions that can be taken (refer to Appendix A function definitions). The default action is to Log the Error and Continue (LOGC) execution of the test (also see Sections SAT.5 and DEBUG.1.3).

FLIB.2.1.4 TIMER, COUNTER AND DELAY SETUP

Function Name	Use
delayms	Millisecond Delay
delays	Second Delay
stats_reset	Reset Statistics Counters
stats_window	Statistics Window Presentation (Global or Function)
tmrset	User Timer Preset
tmrstart	User Timer Start
tmrstop	User Timer Stop
ucinc	User Counter Increment/Decrement
ucname	User Counter Name
ucrst	User Counter Reset

These functions control the general-purpose timer, counter, and delay. The timer can be controlled by starting (**tmrstart()**), stopping (**tmrstop()**), or initializing (**tmrset()**) the timer. The user counter allows the user to set up his own counter. By giving the counter a name (**ucname()**), the user may be able to increment/decrement (**ucinc()**) or reset (**ucrst()**) the counter. The user can also delay execution by seconds (**delays()**) or milliseconds (**delayms()**). The statistic counters can be set to zero by the **stats_reset()** function. The presentation of the statistics can be global or within the function (**stats_window()**).

FLIB.2.1.5 MISCELLANEOUS

Function Name	Use
debug	Interrupt Test Execution and Enter Debugger
pause	Pause Test Execution

The **pause()** function will allow the user to pause during execution of the SAT program. A specified message string (passed by **pause()**) will be displayed on the screen. The SAT program will continue to execute once a **return** key has been hit.

The **debug()** function will allow the SAT program to convert over to a another debug level while executing. This function will be useful in situations where the SAT program is running under debug level Ø and one needs to examine the execution of commands or to track a certain process by inserting the **debug()** function and specifying another level. The SAT test will convert over to the level specified and the user may control the SAT execution from that level (as long as the debug level is greater than zero).

FLIB.2.2 I/O DRIVER CLASS

FLIB.2.2.1 SCSI RELATED FUNCTIONS

Function Name	Use	Default
arbmode	Set Arbitration Mode	NONE
autosense	Enable/Disable Auto Sense	OFF
busywait	Enable/Disable Busy Wait	OFF
cntlbyte	Set Execution Control Byte	00
exp_status	Expected Status After Mask	00
fixed	Sequential Access Fixed Bit	1
iid	Set ID	jumper selectable
ioto	Set I/O Time-Out Value	90 sec
lun	Set Execution LUN	0
selmode	Set Selection Mode	DUMB
stat_mask	Set Expected Status Mask	00
tid	Set Execution Target ID	0

These are the I/O Driver SCSI related functions.

FLIB.2.2.2 I/O DRIVER STATUS FUNCTIONS

Function Name	Use
bcu	Enable Buffer/Command Frame Update
statsen	Enable Statistics Gathering

To enable/disable statistics gathering, use the **statsen()** function. To enable/disable buffer/command frame update, use the **bcu()** function.

FLIB.2.2.3 blk() FUNCTIONS

Function Name	Use
blk_size	Defines Block Size to be used with **dmaset_vblk()**
inc_blk	Increment Starting Block Address for **_blk()** Functions
inc_len	Increment Transfer Length for **_blk()** Functions
random_blk	Generate Random Starting Block Address for **_blk()** Functions
random_len	Generate Random Transfer Length for **_blk()** Functions
set_blk	Set Starting Block Address for **_blk()** Functions
set_len	Set Transfer Length for **_blk()** Functions

These are the **_blk()** related functions.

FLIB.2.3 MICROPROGRAMMING CLASS

Function Name	Use
arb_or_resel	Arbitrate or Reselect
bfreearm	Arm Bus Free Detection Logic
busrel	Release Bus
forcbusy	Force SCSI Bus to Busy
forceattn	Force SCSI Bus Attention
forcperr	Force Parity Error
resel_wt	Wait for Reselection Phase
ureset	Bus Reset

These are the Microprogramming functions available for setup. These functions allow the user to exercise close control over the SCSI Initiator function. This close control permits detailed message system testing and SCSI parity error generation. In addition, Microprogramming allows faster phase transactions than the SDS-1 I/O Driver. Since the Microprogramming functions are designed for fast execution, each function has as few arguments as possible. Certain function types have many different versions to allow for flexibility in programming while maintaining execution speed.

> **NOTE:** Since Microprogramming basically takes over the Test Adapter Hardware Test Functions, calls to the I/O Driver should not be executed until the SCSI bus has gone to a bus free state with a command complete message (disconnect does not count), i.e., the test adapter Microprogramming sequence should complete a SCSI command (or issue a SCSI bus reset).

FLIB.3 EXECUTION TEST FUNCTIONS

These types of functions perform certain tasks during the execution of the test program: such as, format the disk (**format()**) or rewind the tape (**rewind()**).

FLIB.3.1 GENERIC CLASS

Function Name	Use
user_input	User Action/Response Requested

The **user_input()** function will stop SAT execution and wait for the user to enter a specific response.

FLIB.3.2 I/O DRIVER CLASS

FLIB.3.2.1 GENERAL PURPOSE SCSI FUNCTIONS (COMMANDS)

Function Name	Use
copy	Copy
inquiry	Inquiry
io6	6-Byte SCSI Command
io10	10-Byte SCSI Command
io12	12-Byte SCSI Command
recvdiag	Receive Diagnostic
senddiag	Send Diagnostic
sense	Request Sense
testur	Test Unit Ready

These are the general-purpose SCSI functions. Functions **io6()**, **io10()**, and **io12()** can be used to form commands that may not be possible for the other functions to execute. For example, the **reads()** function will accept a maximum of 64k bytes to read, but when **io6()** is setup properly, more than 64K bytes can be read.

Before the other functions (except **io6()**, **io10()** or **io12()**) are used, the **lun()** and **cntlbyte()** functions should be called to initialize their values in the command descriptor block structure, otherwise their default values will be used.

FLIB.3.2.2 RANDOM ACCESS DEVICE FUNCTIONS

Function Name	Use
ccs_modsel	CCS Mode Select
ccs_modsens	CCS Mode Sense
comp	Compare
copyver	Copy & Verify
format	Format
modesen	Mode Sense
mode_sel	Mode Select
prevmedr	Prevent/Allow Media Removal on Random Access Device
rd_buffer	CCS Read Buffer
rd_defect	CCS Read Defect Data
readcap	Read Capacity
readr	Read Random Device
readrl	Read Random 6-Byte with Long Address
readr10	Read Random Device 10-Byte
readr10_blk	Read Random using Predefined BLOCK and LENGTH
readr_blk	Read with Predefined Counts
reasgnb	Reassign Block
releaser	Release Random Device
reservr	Reserve Random Device
rezero	Rezero
searchde	Search Data Equal
searchdh	Search Data High
searchdl	Search Data Low
seek	Seek
seekl	Seek Random with Long Address
seek10	Seek Random Device 10-Byte
setlimts	Set Limits
strstop	Start/Stop
verify10	Verify 10-Byte
writer	Write Random Device
writerl	Write Random with Long Address
writer10	Write Random Device 10-Byte
writer10_blk	Write Random using Predefined BLOCK and LENGTH
writer_blk	Write with Predefined Counts
wrtvfy10	Write and Verify 10-Byte
wrt_buffer	CCS Write Buffer

These are the SCSI command functions for the random access device. Before these functions are used, the **lun()** and **cntlbyte()** functions should be called to initialize their values in the command descriptor block structure, otherwise their default values will be used.

FLIB.3.2.3 SEQUENTIAL ACCESS DEVICE FUNCTIONS

Function Name	Use
erase	Erase
ldunlds	Load/Unload
modsels	Mode Select
modsens	Mode Sense
prevmeds	Prevent/Allow Media Removal on Sequential Access Device
rdblklts	Read Block Limits
readrev	Read Reverse (64K blocks Max)
reads	Read Sequential (64K blocks Max)
readsl	Read Sequential (long count)
recbufds	Recover Buffer Data
releases	Release Unit
reserves	Reserve Unit
rewind	Rewind
space	Space (64K Max)
tksel	Track Select
verifys	Verify Sequential
writes	Write Sequential (64K blocks Max)
writesl	Write Sequential (long count)
wrtfilm	Write File Marks

These are the SCSI command functions for the sequential access devices. Before these functions are used, the **lun()**, **fixed()**, and **cntlbyte()** functions should be called to initialize their values in the command descriptor block structure, otherwise their default values will be used.

FLIB.3.3 MICROPROGRAMMING CLASS

Function Name	Use
arb1	Software Arbitration
arb2	Hardware Arbitration
cdb61	6-Byte DMA Command Out
cdb62	6-Byte T/R Machine Command Out
cdb63	6-Byte PIO Command Out
cdb1Ø1	1Ø-Byte DMA Command Out
cdb1Ø2	1Ø-Byte T/R Machine Command Out
cdb1Ø3	1Ø-Byte PIO Command Out
cdb121	12-Byte DMA Command Out
cdb122	12-Byte T/R Machine Command Out
cdb123	12-Byte PIO Command Out
datainØ	OBB Data In
datain1	DMA Data In
datain2	T/R Machine Data In
datain3	PIO Data In
datain4	OBB Hardware Compare Data In
datain5	DMA Hardware Compare Data In
dataoutØ	OBB Data Out
dataout1	DMA Data Out
dataout2	T/R Machine Data Out
dataout3	PIO Data Out
msgout	Message Out
msgout_atnf	Single Byte Message Out w/ATTN true
sel1	Nonarbitration Selection
sel2	Selection with No Message Out
sel3	Smart Arbitration Selection
sel4	Smart Selection with Message Out

These are the Microprogramming functions needed for execution of test programs that will bypass the I/O Driver. These functions will provide the programmer with a means to generate SCSI bus excitation (arbitration, selection, command out, messages, data transfer handshakes, etc.). Some of the excitation functions also contain built-in Data Reduction capability. These functions will be useful in situations where a test is needed without the I/O Driver. The SCSI bus must be in a free state when done with testing.

FLIB.4 DATA ANALYSIS/REDUCTION TEST FUNCTIONS

These functions will compare results and/or report them.

FLIB.4.1 GENERIC CLASS

Function Name	Use
bus_logen	Enable/Disable Bus Logging
chk_user_limits	Check Limits from **user_input()**
chk_user_string	Check for Match in **user_input()**
compwr	Compare Write and Read Buffers
copy_user_string	Copy String from **user_input()**
delta_time	Get Time Between Two State Log Entries
dispbuf	Display Buffer to Screen
error_ok	Decrement Error Count
eseom	Extended Sense EOM Bit Check
esfm	Extended Sense File Mark Check
esili	Extended Sense Illegal Length Indicator Check
esinfob	Extended Sense Information Bytes Check
eskey	Extended Sense Key Equal Check
eskeynot	Extended Sense Key Not Equal Check
esvalid	Extended Sense Valid Check
get_byte	Get Byte from Defined Buffer
get_user_int	Return Integer from **user_input()**
get_user_long	Return Long from **user_input()**
rbufbyte	Compare Read Buffer Byte Within Limits
rbufword	Compare Read Buffer Word Within Limits
rptbuf	Write Buffer to Report Log
rptsen	Write Sense Buffer to Report Log
rptstats	Write Statistics to Report Log
rpttmr	Write Timers to Report Log
sbb	Sense Byte Check
sbw	Sense Word Check
serclass	Std Sense Error Class
serrcd	Std Sense Error Code
sladdr	Check Std Sense Logical Block Address
state_data	Get Data Associated with a State Log Entry
svalid	Std Sense Address Valid
svu	Std Sense Vendor Unique
tmrlmt	User Timer Limit Check
tmrvalue	Return Timer Value

The functions are the data analysis functions. They compare, check and test the data values. Also included are the reduction functions.

FLIB.4.2 I/O DRIVER CLASS

Function Name	Use
bytcmp	Check Bytes Compared Limits
bytrd	Check Bytes Read Count Limits
bytwrt	Check Bytes Written Count Limits
get_f_stats	Return Function Statistics Information
get_f_status	Return Function Status Information
get_g_stats	Return Global Statistics Information
opcnt	Check Operation Count Limits

These functions compare the I/O Driver values and checks to see if they are within the range specified.

FLIB.4.3 MICROPROGRAMMING CLASS

Function Name	Use
arblose	Check for TARGET Arb Lose
arbwin	Check for TARGET Arb Win
awin_res	Check for TARGET Arb Win and Allow Reselect
bfreeck	Bus Free Check
get_infoin	Get and Acknowledge Data In
get_phase	Get Current Bus Phase
msgin	Expected Message In
resel	Reselection
statin	Expected Status In

These are the Microprogramming functions for data analysis/reduction. These functions provide a means of checking response from the TARGET to the INITIATOR excitation functions. They also look for the expected response from the TARGET and generate an implicit error if the desired response is not detected.

FLIB.5 REPORT DOCUMENTATION FUNCTIONS

These functions are the report generation functions for the Test Results report.

Function Name	Use
cmd_tail_bol	Search Command Tail for String
cmd_tail_string	Search Command Tail for String and Return the Following Parameter
fail	Print Fail line on Screen and Report
group	Print Group Line and Generate a TOC Entry
logc	Print a Log Line to Console (Log Device)
logp	Print a Log Line to Printer and Log Device
page	Page Eject in Test Results
paragph	Print a Paragraph Line and TOC Entry
pass	Print Pass Line on Screen and Report
subpar	Print Subparagraph Line and TOC Entry
summary	Print Summary Line
test	Print Test Line and TOC Entry

APPENDIX A

SDS-1 FUNCTION LIBRARY

A.0 SDS-1 FUNCTION LIBRARY

A.1 FUNCTION LISTINGS

Functions are grouped or defined in the following manner:

```
TYPE       (Setup, Execution, Analysis)
  CLASS    (Generic, I/O Driver, or Microprogramming)
    FUNCTION GROUP
```

The TYPE and CLASS groupings help the user determine the proper usage of a function. For example a GENERIC SETUP function is used to set up the initial conditions for either a I/O Driver or Microprogramming Execution Function.

A.1.1 FUNCTIONS LISTED BY TYPE, CLASS AND GROUP

	Setup Test Functions:
	Generic Class
	Configuration Setup
ackdelay(count);	OBB Acknowledge Delay
errdelay(bit);	Enable/Disable Five Second Error Message Delay
line_mode("S/D");	Select Single-Ended or Differential SCSI mode
parity(0/1);	Enable/Disable SCSI Bus Parity
reset();	Reset SCSI Bus/I/O Driver
set_er_limits(limit);	Set Error Limits
xfermode("mode",buf_size);	Open R/W Buffer/Set Transfer Mode
	Buffer Setup
dmarst("r/w");	Reset DMA Pointer
dmaset("r/w",address);	Set DMA Pointer
dmaset_va("r/w",addressL);	Set the Virtual DMA address
dmaset_vblk("r/w");	Set Virtual DMA address for the Defined Block
fillbcb(st_byt,blk_len,st_add,len);	Byte Block Count Fill
fillbcw(st_wrd,blk_len,st_add,len);	Word Block Count Fill
fillbyte(char,st_add,len);	Fill Buffer with Byte
filld(st_byt,st_add,len);	Decrement Count FIll
filli(st_byt,st_add,len);	Increment Count Fill
fillk("string",st_add,len);	Constant Fill
fillpr(seed,st_add,len);	Pseudo Random Fill
loadbuf("file",st_add,length);	Load Buffer from Disk
overbcb(st_byt,blk_len,st_add,len);	Overlay Block Count Byte
overbcdw(st_dblwrdL,blk_len,st_add,len);	Overlay Block Count Double Word
overbcw(st_wrd,blk_len,st_add,len);	Overlay Block Count Word
put_byte("r/w/s",address,byte);	Put Buffer with Data Byte
savebuf("file",st_add,length);	Save Fill Buffer to Disk
setbuf("string",st_add);	Fill Buffer with ASCII String
setfill_buf("r/w/s");	Set Current Fill Buffer
	Error Action/Recovery Setup
eea("action");	Explicit Error Action
iea("action");	Implicit Error Action
	Timer, Counter and Delay Setup
delayms(ms_delay);	Millisecond Delay
delays(sec_delay);	Second Delay
stats_reset("counter_id");	Reset Statistics Counters
stats_window("g/f");	Statistics Window Presentation (Global or Function)
tmrset(value);	User Timer Preset
tmrstart("U/D");	User Timer Start
tmrstop();	User Timer Stop
ucinc(0/1,value);	User Counter Increment/Decrement
ucname(0/1,"name");	User Counter Name
ucrst(0/1);	User Counter Reset

Miscellaneous

debug(level);	Interrupt Test Execution and Enter Debugger
pause("message");	Pause Test Execution

I/O Driver Class

SCSI Related Functions

arbmode(mode);	Set Arbitration Mode
autosense(Ø/1);	Enable/Disable Auto Sense
busywait(Ø/1);	Enable/Disable Busy Wait
cntlbyte(byte);	Set SCSI Command Control byte
exp_status(value);	Expected Status after Mask
fixed(Ø/1);	Sequential Access Fixed Bit
iid(Ø,newid);	Set ID
ioto(value);	Set I/O Time-Out Value
lun(lun);	Set Execution LUN
selmode("mode");	Set Selection Mode
stat_mask(byte);	Set Expected Status Mask
tid(newid);	Set Execution Target ID

I/O Driver Status Functions

bcu(Ø/1);	Enable Buffer/Command Frame Update
statsen(Ø/1);	Enable Statistics Gathering

blk() Functions

blk_size(size);	Defines Block Size to be used with **dmaset_vblk()**
inc_blk(increment);	Increment Starting Block Address for **_blk()** Functions
inc_len(increment);	Increment Transfer Length for **_blk()** Functions
random_blk(minL,maxL);	Generate Random Starting Block Address for **_blk()** Functions
random_len(min,max);	Generate Random Transfer Length for **_blk()** Functions
set_blk(valueL);	Set Starting Block Address for **_blk()** Functions
set_len(value);	Set Transfer Length for **_blk()** Functions

Microprogramming Class

arb_or_resel(iid);	Arbitrate or Reselect
bfreearm();	Arm Bus Free Detection Logic
busrel();	Release Bus
forcbusy();	Force SCSI Bus to Busy
forceattn(n);	Force SCSI Bus Attention
forcperr(n);	Force Parity Error
resel_wt();	Wait for Reselection Phase
ureset();	Bus Reset

Execution Test Functions:

Generic Class

```
user_input("string","type");  User Action/Response Requested
```

I/O Driver Class

General Purpose SCSI Functions

```
copy(lenL);                   Copy
inquiry(len);                 Inquiry
io6(b0,b1,...b4,b5);          6-Byte SCSI Command
io10(b0,b1,...b8,b9);         10-Byte SCSI Command
io12(b0,b1,...b10,b11);       12-Byte SCSI Command
recvdiag(len);                Receive Diagnostic
senddiag(selftst,devof,unitof,len); Send Diagnostic
sense(len);                   Request Sense
testur();                     Test Unit Ready
```

Random Access Device Functions

```
ccs_modsel(list_len,sp);      CCS Mode Select
ccs_modsens(len,pcf,pagecode);CCS Mode Sense
comp(lenL);                   Compare
copyver(bytck,lenL);          Copy & Verify
format(fd,cmpl,dflist,intrleave); Format
modesen(alloc_len);           Mode Sense
mode_sel(list_len);           Mode Select
prevmedr(prvent);             Prevent/Allow Media Removal on
                              Random Access Device
rd_buffer(length,bcv,vu2,vu3,vu4,vu5,vu6); CCS Read Buffer
rd_defect(length,p,g,format); CCS Read Defect Data
readcap(reladr,addL,pmi);     Read Capacity
readr(start,len);             Read Random Device
readrl(st_addL,len);          Read Random 6-Byte with Long
                              Address
readr10(reladr,st_addL,len);  Read Random Device 10-Byte
readr10_blk();                Read Random using Predefined BLOCK
                              and LENGTH
readr_blk();                  Read with Predefined Counts
reasgnb();                    Reassign Block
releaser(3rd,3rdid,ext,resid);Release Random Device
reservr(3rd,3rdid,ext,resid,list); Reserve Random Device
rezero();                     Rezero
searchde(inv,rcdfmt,spndat,reladr,st_addL,len); Search Data Equal
searchdh(inv,rcdfmt,spndat,reladr,st_addL,len); Search Data High
searchdl(inv,rcdfmt,spndat,reladr,st_addL,len); Search Data Low
seek(add);                    Seek
seekl(addL);                  Seek Random with Long address
seek10(addL);                 Seek Random Device 10-Byte
setlimts(rdinh,wrinh,st_addL,len); Set Limits
strstop(immed,start);         Start/Stop
verify10(bytck,reladr,st_addL,len); Verify 10-Byte
writer(start,len);            Write Random Device
writerl(startL,len);          Write Random with Long Address
writer10(reladr,st_addL,len); Write Random Device 10-Byte
writer10_blk();               Write Random using Predefined BLOCK
                              and LENGTH
```

writer_blk();	Write with Predefined Counts
wrtvfy1Ø(bytck,reladr,st_addL,len);	Write and Verify 1Ø-Byte
wrt_buffer(length,bcv,vu2,vu3,vu4,vu5,vu6);	CCS Write Buffer

<u>Sequential Access Device Functions</u>

erase(long);	Erase
ldunlds(immed,reten,load);	Load/Unload
modsels(list_len);	Mode Select
modsens(len);	Mode Sense
prevmeds(prevent);	Prevent/Allow Media Removal on Sequential Access Device
rdblklts();	Read Block Limits
readrev(len);	Read Reverse (64K blocks Max)
reads(len);	Read Sequential (64K blocks Max)
readsl(lenL);	Read Sequential (long count)
recbufds(len);	Recover Buffer Data
releases(3rd,3rdid);	Release Unit
reserves(3rd,3rdid);	Reserve Unit
rewind(immed);	Rewind
space(code,count);	Space (64K Max)
tksel(tk_val);	Track Select
verifys(bytcmp,len);	Verify Sequential
writes(len);	Write Sequential (64K blocks Max)
writesl(lenL);	Write Sequential (long count)
wrtfilm(count);	Write File Marks

<u>Microprogramming Class</u>

arb1(iid);	Software Arbitration
arb2(iid);	Hardware Arbitration
cdb61(bØ,,,,,b5);	6-Byte DMA Command Out
cdb62(bØ,,,,,b5);	6-Byte T/R Machine Command Out
cdb63(bØ,,,,,b5);	6-Byte PIO Command Out
cdb1Ø1(bØ,,,,,,,,,b9);	1Ø-Byte DMA Command Out
cdb1Ø2(bØ,,,,,,,,,b9);	1Ø-Byte T/R Machine Command Out
cdb1Ø3(bØ,,,,,,,,,b9);	1Ø-Byte PIO Command Out
cdb121(bØ,,,,,,,,,,,b11);	12-Byte DMA Command Out
cdb122(bØ,,,,,,,,,,,b11);	12-Byte T/R Machine Command Out
cdb123(bØ,,,,,,,,,,,b11);	12-Byte PIO Command Out
datainØ(countL,mode);	OBB Data In
datain1(countL,mode);	DMA Data In
datain2(countL,mode);	T/R Machine Data In
datain3(countL,mode);	PIO Data In
datain4(countL,mode);	OBB Hardware Compare Data In
datain5(countL,mode);	DMA Hardware Compare Data In
dataoutØ(countL,mode);	OBB Data Out
dataout1(countL,mode);	DMA Data Out
dataout2(countL,mode);	T/R Machine Data Out
dataout3(countL,mode);	PIO Data Out
msgout(mo);	Message Out
msgout_atnf(mo);	Single Byte Message Out w/ATTN True
sel1(tid);	Nonarbitration Selection
sel2(tid,iid);	Selection with No Message Out
sel3(tid);	Smart Arbitration Selection
sel4(tid,msgout);	Smart Selection with Message Out

Data Analysis/Reduction Functions:

Generic Class

bus_logen(0/1);	Enable/Disable Bus Logging
chk_user_limits(lo,hi);	Check Limits from **user_input()**
chk_user_string("ref_string");	Check for Match in **user_input()**
compwr(st_add,len);	Compare Write and Read Buffers
copy_user_string("tgt_string");	Copy String from **user_input()**
delta_time("state1",count1,"state2",count2);	Get Time Between Two State Log Entries
dispbuf("buffer",start_add,length);	Display Buffer to Screen
error_ok("NODSPL/DISPLAY");	Decrement Error Count
eseom(n);	Extended Sense EOM Bit Check
esfm(n);	Extended Sense File Mark Check
esili(n);	Extended Sense Illegal Length Indicator Check
esinfob(minL,maxL);	Extended Sense Information Bytes Check
eskey(value);	Extended Sense Key Equal Check
eskeynot(value);	Extended Sense Key Not Equal Check
esvalid(n);	Extended Sense Valid Check
get_byte("r/w/s",address);	Get Byte from Defined Buffer
get_user_int();	Return Integer from **user_input()**
get_user_long();	Return Long from **user_input()**
rbufbyte(address,lo,hi);	Compare Read Buffer Byte within Limits
rbufword(address,lo,hi);	Compare Read Buffer Word within Limits
rptbuf("buffer",start_add,len);	Write Buffer to Report Log
rptsen();	Write Sense Buffer to Report Log
rptstats(0/1);	Write Statistics to Report Log
rpttmr();	Write Timers to Report Log
sbb(address,min,max);	Sense Byte Check
sbw(address,min,max);	Sense Word Check
serclass(class);	Std Sense Error Class
serrcd(code);	Std Sense Error Code
sladdr(minL,maxL);	Check Std Sense Logical Block Address
state_data("state",count);	Get Data Associated with a State Log Entry
svalid(n);	Std Sense Address Valid
svu(value);	Std Sense Vendor Unique
tmrlmt(lo,hi);	User Timer Limit Check
tmrvalue();	Return Timer Value

I/O Driver Class

bytcmp(minL,maxL);	Check Bytes Compared Limits
bytrd(minL,maxL);	Check Bytes Read Count Limits
bytwrt(minL,maxL);	Check Bytes Written Count Limits
get_f_stats("counter_id");	Return Function Statistics Information
get_f_status("status_id");	Return Function Status Information

get_g_stats("counter_id");	Return Global Statistics Information
opcnt(minL,maxL);	Check Operation Count Limits
	<u>Microprogramming Class</u>
arblose(id);	Check for TARGET Arb Lose
arbwin(id);	Check for TARGET Arb Win
awin_res(iid);	Check for TARGET Arb Win and Allow Reselect
bfreeck();	Bus Free Check
get_infoin();	Get Current Inbound Information Byte
get_phase(req_wait);	Get Current Bus Phase
msgin(mi);	Expected Message In
resel();	Reselection
statin(si);	Expected Status In

<u>Report Documentation Functions:</u>

cmd_tail_bol("string");	Search Command Tail for String
cmd_tail_string("look_for","return_parameter");	Search Command Tail for String and Return the Following Parameter
fail("fail_string");	Print Fail Line on Screen and Report
group("Group Name");	Print Group Line and Generate a TOC entry
logc("string");	Print a Log Line to Console (Log Device)
logp("string");	Print a Log Line to Printer and Log Device
page();	Page Eject in Test Results
paragph("Paragraph Name");	Print a Paragraph Line and TOC Entry
pass();	Print Pass Line on Screen and Report
subpar("Sub-Paragraph Name","ref_string");	Print Subparagraph Line and TOC entry
summary("summary_string");	Print Summary Line
test("FILENAME Test Title");	Print Test Line and TOC Entry

A.1.2 FUNCTIONS LISTED ALPHABETICALLY

~A.1 A,B

ackdelay(count);	OBB Acknowledge Delay
arblose(id);	Check for TARGET Arb Lose
arbmode(mode);	Set Arbitration Mode
arbwin(id);	Check for TARGET Arb Win
arb1(iid);	Software Arbitration
arb2(iid);	Hardware Arbitration
arb_or_resel(iid);	Arbitrate or Reselect
autosense(0/1);	Enable/Disable Auto Sense
awin_res(iid);	Check for TARGET Arb Win and Allow Reselect
bcu(0/1);	Enable Buffer/Command Frame Update
bfreearm();	Arm Bus Free Detection Logic
bfreeck();	Bus Free Check
blk_size(size);	Defines Block Size to be used with **dmaset_vblk()**
busrel();	Release Bus
busywait(0/1);	Enable/Disable Busy Wait
bus_logen(0/1);	Enable/Disable Bus Logging
bytcmp(minL,maxL);	Check Bytes Compared Limits
bytrd(minL,maxL);	Check Bytes Read Count Limits
bytwrt(minL,maxL);	Check Bytes Written Count Limits

~A.2 C's

ccs_modsel(list_len,sp);	CCS Mode Select
ccs_modsens(len,pcf,pagecode);	CCS Mode Sense
cdb61(b0,,,,,b5);	6-Byte DMA Command Out
cdb62(b0,,,,,b5);	6-Byte T/R Machine Command Out
cdb63(b0,,,,,b5);	6-Byte PIO Command Out
cdb101(b0,,,,,,,,,b9);	10-Byte DMA Command Out
cdb102(b0,,,,,,,,,b9);	10-Byte T/R Machine Command Out
cdb103(b0,,,,,,,,,b9);	10-Byte PIO Command Out
cdb121(b0,,,,,,,,,,,b11);	12-Byte DMA Command Out
cdb122(b0,,,,,,,,,,,b11);	12-Byte T/R Machine Command Out
cdb123(b0,,,,,,,,,,,b11);	12-Byte PIO Command Out
chk_user_limits(lo,hi);	Check Limits from **user_input()**
chk_user_string("ref_string");	Check for Match in **user_input()**
cmd_tail_bol("string");	Search Command Tail for String
cmd_tail_string("look_for","return_parameter");	Search Command Tail for String and Return the Following Parameter
cntlbyte(byte);	Set SCSI Command Control byte
comp(lenL);	Compare
compwr(st_add,len);	Compare Write and Read Buffers
copy(lenL);	Copy
copyver(bytck,lenL);	Copy & Verify
copy_user_string("tgt_string");	Copy String from **user_input()**

~A.3 D's

datain0(countL,mode);	OBB Data In
datain1(countL,mode);	DMA Data In
datain2(countL,mode);	T/R Machine Data In
datain3(countL,mode);	PIO Data In
datain4(countL,mode);	OBB Hardware Compare Data In
datain5(countL,mode);	DMA Hardware Compare Data In

dataout0(countL,mode);	OBB Data Out
dataout1(countL,mode);	DMA Data Out
dataout2(countL,mode);	T/R Machine Data Out
dataout3(countL,mode);	PIO Data Out
debug(level);	Interrupt Test Execution and Enter Debugger
delayms(ms_delay);	Millisecond Delay
delays(sec_delay);	Second Delay
delta_time("state1",count1,"state2",count2);	Get Time Between Two State Log Entries
dispbuf("buffer",start_add,length);	Display Buffer to Screen
dmarst("r/w");	Reset DMA Pointer
dmaset("r/w",address);	Set DMA Pointer
dmaset_va("r/w",addressL);	Set the Virtual DMA address
dmaset_vblk("r/w");	Set Virtual DMA address for the Defined Block

~A.4 E,F

eea("action");	Explicit Error Action
erase(long);	Erase
errdelay(bit);	Enable/Disable Five Second Error Message Delay
error_ok("NODSPL/DISPLAY");	Decrement Error Count
eseom(n);	Extended Sense EOM Bit Check
esfm(n);	Extended Sense File Mark Check
esili(n);	Extended Sense Illegal Length Indicator Check
esinfob(minL,maxL);	Extended Sense Information Bytes Check
eskey(value);	Extended Sense Key Equal Check
eskeynot(value);	Extended Sense Key Not Equal Check
esvalid(n);	Extended Sense Valid Check
exp_status(value);	Expected Status After Mask
fail("fail_string");	Print Fail Line on Screen and Report
fillbcb(st_byt,blk_len,st_add,len);	Byte Block Count Fill
fillbcw(st_wrd,blk_len,st_add,len);	Word Block Count Fill
fillbyte(char,st_add,len);	Fill Buffer with Byte
filld(st_byt,st_add,len);	Decrement Count FIll
filli(st_byt,st_add,len);	Increment Count Fill
fillk("string",st_add,len);	Constant Fill
fillpr(seed,st_add,len);	Pseudo Random Fill
fixed(0/1);	Sequential Access Fixed Bit
forcbusy();	Force SCSI Bus to Busy
forceattn(n);	Force SCSI Bus Attention
forcperr(n);	Force Parity Error
format(fd,cmpl,dflist,intrleave);	Format

~A.5 G,H,I,J,K

get_byte("r/w/s",address);	Get Byte from Defined Buffer
get_f_stats("counter_id");	Return Function Statistics Information
get_f_status("status_id");	Return Function Status Information
get_g_stats("counter_id");	Return Global Statistics Information
get_infoin();	Get Current Inbound Information Byte

get_phase(req_wait);	Get Current Bus Phase
get_user_int();	Return Integer from **user_input()**
get_user_long();	Return Long from **user_input()**
group("Group Name");	Print Group Line and Generate a TOC entry
iea("action");	Implicit Error Action
iid(0,newid);	Set ID
inc_blk(increment);	Increment Starting Block Address for **_blk()** Functions
inc_len(increment);	Increment Transfer Length for **_blk()** Functions
inquiry(len);	Inquiry
ioto(value);	Set I/O Time-Out Value
io6(b0,b1,...b4,b5);	6-Byte SCSI Command
io10(b0,b1,...b8,b9);	10-Byte SCSI Command
io12(b0,b1,...b10,b11);	12-Byte SCSI Command

~A.6 L,M,N,O,P,Q

ldunlds(immed,reten,load);	Load/Unload
line_mode("S/D");	Select Single-Ended or Differential SCSI mode
loadbuf("file",st_add,length);	Load Buffer from Disk
logc("string");	Print a Log Line to Console (Log Device)
logp("string");	Print a Log Line to Printer and Log Device
lun(lun);	Set Execution LUN
modesen(alloc_len);	Mode Sense
mode_sel(list_len);	Mode Select
modsels(list_len);	Mode Select
modsens(len);	Mode Sense
msgin(mi);	Expected Message In
msgout(mo);	Message Out
msgout_atnf(mo);	Single Byte Message Out w/ATTN True
opcnt(minL,maxL);	Check Operation Count Limits
overbcb(st_byt,blk_len,st_add,len);	Overlay Block Count Byte
overbcdw(st_dblwrdL,blk_len,st_add,len);	Overlay Block Count Double Word
overbcw(st_wrd,blk_len,st_add,len);	Overlay Block Count Word
page();	Page Eject in Test Results
paragph("Paragraph Name");	Print a Paragraph Line and TOC Entry
parity(0/1);	Enable/Disable SCSI Bus Parity
pass();	Print Pass Line on Screen and Report
pause("message");	Pause Test Execution
prevmedr(prvent);	Prevent/Allow Media Removal on Random Access Device
prevmeds(prevent);	Prevent/Allow Media Removal on Sequential Access Device
put_byte("r/w/s",address,byte);	Put Buffer with Data Byte

~A.7 R's

random_blk(minL,maxL);	Generate Random Starting Block Address for **_blk()** Functions
random_len(min,max);	Generate Random Transfer Length for **_blk()** Functions

rbufbyte(address,lo,hi);	Compare Read Buffer Byte within Limits
rbufword(address,lo,hi);	Compare Read Buffer Word within Limits
rdblklts();	Read Block Limits
rd_buffer(length,bcv,vu2,vu3,vu4,vu5,vu6);	CCS Read Buffer
rd_defect(length,p,g,format);	CCS Read Defect Data
readcap(reladr,addL,pmi);	Read Capacity
readr(start,len);	Read Random Device
readrev(len);	Read Reverse (64K blocks Max)
readrl(st_addL,len);	Read Random 6-Byte with Long Address
readrl0(reladr,st_addL,len);	Read Random Device 10-Byte
readrl0_blk();	Read Random using Predefined BLOCK and LENGTH
readr_blk();	Read with Predefined Counts
reads(len);	Read Sequential (64K blocks Max)
readsl(lenL);	Read Sequential (long count)
reasgnb();	Reassign Block
recbufds(len);	Recover Buffer Data
recvdiag(len);	Receive Diagnostic
releaser(3rd,3rdid,ext,resid);	Release Random
releases(3rd,3rdid);	Release Unit
resel();	Reselection
resel_wt();	Wait for Reselection Phase
reserves(3rd,3rdid);	Reserve Unit
reservr(3rd,3rdid,ext,resid,list);	Reserve Random Device
reset();	Reset SCSI Bus/I/O Driver
rewind(immed);	Rewind
rezero();	Rezero
rptbuf("buffer",start_add,len);	Write Buffer to Report Log
rptsen();	Write Sense Buffer to Report Log
rptstats(0/1);	Write Statistics to Report Log
rpttmr();	Write Timers to Report Log

~A.8 S's

savebuf("file",st_add,length);	Save Fill Buffer to Disk
sbb(address,min,max);	Sense Byte Check
sbw(address,min,max);	Sense Word Check
searchde(inv,rcdfmt,spndat,reladr,st_addL,len);	Search Data Equal
searchdh(inv,rcdfmt,spndat,reladr,st_addL,len);	Search Data High
searchdl(inv,rcdfmt,spndat,reladr,st_addL,len);	Search Data Low
seek(add);	Seek
seekl(addL);	Seek Random with Long Address
seekl0(addL);	Seek Random Device 10-Byte
selmode("mode");	Set Selection Mode
sel1(tid);	Nonarbitration Selection
sel2(tid,iid);	Selection with No Message Out
sel3(tid);	Smart Arbitration Selection
sel4(tid,msgout);	Smart Selection with Message Out
senddiag(selftst,devof,unitof,len);	Send Diagnostic
sense(len);	Request Sense
serclass(class);	Std Sense Error Class
serrcd(code);	Std Sense Error Code
setbuf("string",st_add);	Fill Buffer with ASCII String
setfill_buf("r/w/s");	Set Current Fill Buffer

setlimts(rdinh,wrinh,st_addL,len);	Set Limits
set_blk(valueL);	Set Starting Block Address for **_blk()** Functions
set_er_limits(limit);	Set Error Limits
set_len(value);	Set Transfer Length for **_blk()** Functions
sladdr(minL,maxL);	Check Std Sense Logical Block Address
space(code,count);	Space (64K Max)
state_data("state",data);	Get Data Associated with a State Log Entry
statin(si);	Expected Status In
statsen(0/1);	Enable Statistics Gathering
stats_reset("counter_id");	Reset Statistics Counters
stats_window("g/f");	Statistics Window Presentation (Global or Function)
stat_mask(byte);	Set Expected Status Mask
strstop(immed,start);	Start/Stop
subpar("Sub-Paragraph Name","ref_string");	Print Subparagraph Line and TOC entry
summary("summary_string");	Print Summary Line
svalid(n);	Std Sense Address Valid
svu(value);	Std Sense Vendor Unique

~A.9 T,U

test("FILENAME Test Title");	Print Test Line and TOC Entry
testur();	Test Unit Ready
tid(newid);	Set Execution Target ID
tksel(tk_val);	Track Select
tmrlmt(lo,hi);	User Timer Limit Check
tmrset(value);	User Timer Preset
tmrstart("U/D");	User Timer Start
tmrstop();	User Timer Stop
tmrvalue();	Return Timer Value
ucinc(0/1,value);	User Counter Increment/Decrement
ucname(0/1,"name");	User Counter Name
ucrst(0/1);	User Counter Reset
ureset();	Bus Reset
user_input("string","type");	User Action/Response Requested

~A.10 V,W,X,Y,Z

verifys(bytcmp,len);	Verify Sequential
verify10(bytck,reladr,st_addL,len);	Verify 10-Byte
writer(start,len);	Write Random Device
writerl(startL,len);	Write Random with Long Address
writer10(reladr,st_addL,len);	Write Random Device 10-Byte
writer10_blk();	Write Random using Predefined BLOCK and LENGTH
writer_blk();	Write with Predefined Counts
writes(len);	Write Sequential (64K blocks Max)
writesl(lenL);	Write Sequential (long count)
wrtfilm(count);	Write File Marks
wrtvfy10(bytck,reladr,st_addL,len);	Write and Verify 10-Byte
wrt_buffer(length,bcv,vu2,vu3,vu4,vu5,vu6);	CCS Write Buffer
xfermode("mode",buf_size);	Open R/W Buffer/Set Transfer Mode

SDS-1 FUNCTION LIBRARY

DETAILED FUNCTION DEFINITIONS (LISTED ALPHABETICALLY)

NAME

ackdelay - set SCSI acknowledge delay time for OBB

SYNOPSIS

```
ackdelay(count);
unsigned count;                    /* delay count in 70ns
                                      increments */
```

DESCRIPTION

This function sets the SDS-1 OBB hardware to perform delayed acknowledge cycles for all High Speed transfer modes. The argument specifies the delay count in 70ns units. The base (minimum ack delay with n = 0) is 280ns.

Also see Section IODVR.3.7 .

DEFAULT VALUE: 0

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME
arblose - check for arbitration loss by target

SYNOPSIS
```
return =  arblose(id);
unsigned return;                 /* function return value */
BYTE id;                         /* arbitration ID used by test
                                    adapter */
```

DESCRIPTION
The **arblose()** function is used in conjunction with the **forcbusy()** function. The intent of the function is to create a situation where a disconnected TARGET will lose bus arbitration when it tries to reconnect to the INITIATOR. This is accomplished in the following manner:

1. **forcbusy()** asserts BUSY via the test adapter PIO ports while TARGET is still asserting BUSY.
2. **delayms()** creates a time delay sufficient enough for the TARGET to be ready to reconnect.
3. **arblose(id)**
   - 3a. sets up the test adapter arbitration logic to arbitrate for the bus (when PIO BUSY is released) as the SCSI ID passed in the **arblose()** argument.
   - 3b. releases PIO BUSY.
   - 3c. verifies that the test adapter arbitration logic has won the arbitration. If test adapter lost, an implicit error message is generated.
   - 3d. reasserts PIO BUSY after arbitration win.
4. **arblose(id)** is called with another SCSI ID which will still result in the TARGET losing the arbitration.

or

4. **arbwin(id)** is called with an SCSI ID which will allow the TARGET to win the arbitration and reselect the INITIATOR.

or

4. **busrel()** releases PIO BUSY asserted by **arblose()**, allowing normal SCSI bus operation.

DEFAULT VALUE: N.A.

RETURNS:
- 0 arbitration won by test adapter (assume that TARGET did not win arbitration)
- 1 arbitration lost by test adapter (assume TARGET won when it should not have)

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:
Initiator Status Byte
- 0x00 good completion
- 0x20 unexpected arbitration lost by test adapter
- 0x0D invalid bus free detected

ERROR MESSAGES:

IMP. ER> arblose(id)
Arbitration Lost By Host Adapter Date/Time Stamp

NAME

arbmode - set arbitration mode

SYNOPSIS

```
return = arbmode(mode);
int return;                    /* return code */
char *mode;                    /* "HDW" = Hardware
                                  "SFTW" = Software
                                  "NONE" = None */
```

DESCRIPTION

This function determines whether and what type of SCSI arbitration is done by the SDS-1. No arbitration (NONE) results in direct assertion of select from the bus free state. Hardware arbitration (HDW) utilizes a state machine to arbitrate and check for arbitration win or lose. Software arbitration (SFTW) utilizes a state machine to assert ID on the bus and remove them if select is detected; it uses software to determine if the SDS-1 has won arbitration resulting in a longer arbitration phase.

Also see Section IODVR.3.3 and IODVR.3.4 .

DEFAULT VALUE: HDW

RETURNS:

NULL(0) function is enabled
1 disabled or function not supported

ERROR MESSAGES:

IMP. ER> arbmode(mode)
Illegal Arbitration Mode Date/Time Stamp

NAME

arbwin - check for arbitration win by TARGET

SYNOPSIS

```
return = arbwin(id);
unsigned return;                /* function return value */
BYTE id;                        /* arbitration ID used by test
                                   adapter */
```

DESCRIPTION

The **arbwin()** function is used in conjunction with **forcbusy()** function. The intent of the function is to create a situation where a disconnected TARGET will win bus arbitration when it tries to reconnect to the INITIATOR. This is accomplished in the following manner:

1. **forcbusy()** asserts BUSY via the test adapter PIO ports while TARGET is still asserting BUSY.
2. **delayms()** creates a time delay sufficient enough for the TARGET to be ready to reconnect.
3. **arbwin(id)**
   3a. sets up the test adapter arbitration logic to arbitrate for the bus (when PIO BUSY is released) as the SCSI ID passed in the **arbwin()** argument.
   3b. releases PIO BUSY set by **forcbusy()**.
   3c. verifies that the test adapter lost arbitration and that the bus is busy (BUSY or SEL asserted).
   3d. disarms the test adapter arbitration logic and restores the correct test adapter ID.
4. **resel()** verifies a valid reselection sequence with the TARGET.

DEFAULT VALUE: N.A.

RETURNS:

Ø arbitration lost by test adapter (assume that TARGET won arbitration)

Øx21 arbitration won by test adapter (assume TARGET won when it should not have)

Øx22 arbitration lost by test adapter, but BSY and SEL false

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte

ØxØØ good completion

Øx21 unexpected win test adapter

ERROR MESSAGES:

IMP. ER> arbwin(id)

Arbitration Won By Host Adapter   Date/Time Stamp

IMP. ER> arbwin(id)
Host Adapter Lost and Bus Not Busy Date/Time Stamp

NAME

arbl - software arbitration function

SYNOPSIS

```
return = arbl(iid);
unsigned return;                /* function return value */
BYTE iid;                       /* initiator ID number */
```

DESCRIPTION

Arbitrate for the SCSI bus using a hardware state machine to assert IDs and deassert IDs if selection is detected. And using software to determine if the test adapter has won arbitration. The function does not return until arbitration has been completed.

DEFAULT VALUE: N.A.

RETURNS:

- 0x00 arbitration complete
- 0x05 function time-out
- 0x09 SCSI bus reset detected

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte

- 0x00 good completion
- 0x05 function time-out
- 0x09 SCSI bus reset detected

ERROR MESSAGES:

IMP. ER> arbl(iid)
I/O Time-out Occurred   Date/Time Stamp

IMP. ER> arbl(iid)
SCSI Reset Occurred   Date/Time Stamp

NAME
    arb2 - hardware arbitration function

SYNOPSIS
```
return = arb2(iid);
unsigned return;                 /* function return value */
BYTE iid;                        /* initiator ID number */
```

DESCRIPTION
    Arbitrate for the SCSI using a hardware state machine to determine if the test adapter has won arbitration. The function does not return until arbitration has been completed.

DEFAULT VALUE: N.A.

RETURNS:
    Øx00 arbitration complete
    Øx05 function time-out
    Øx09 SCSI bus reset detected

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:
    Initiator Status Byte
        Øx00 good completion
        Øx05 function time-out
        Øx09 SCSI bus reset detected

ERROR MESSAGES:
```
IMP. ER> arb2(Ø)
I/O Time-out Occurred                         Date/Time Stamp

IMP. ER> arb2(Ø)
SCSI Reset Occurred                           Date/Time Stamp
```

NAME

arb_or_resel - arbitrate or reselect

SYNOPSIS

```
return = arb_or_resel(iid);
unsigned return;                    /* function return */
BYTE iid;                           /* host ID used for
                                       arbitration */
```

DESCRIPTION

This function returns when one of two events occur:

1. The bus has gone free; the test adapter arbitrated as 'iid' and won.

OR

2. A reselect bus phase has been detected (BSY false, SEL true, I/O- true). This may have occurred after the host attempted to arbitrate as 'iid' and lost. In this case, the return value contains the select byte on the bus. If the user wishes to proceed with a reselect sequence, the correct 'iid' must be set up and **resel()** must be called.

This function is intended to be used in a test which is performing I/Os to more than one target, perhaps from more than one host. This function allows the test to always keep the bus as busy as possible, even when an I/O thread is disconnected.

DEFAULT VALUE: N.A.

RETURNS:

0x0000	host won arbitration
0x00bb	reselect detected; bb = data byte on the bus
0x0500	I/O time-out
0x0900	SCSI bus reset detected

EXECUTION TYPE: Microprogramming

ERROR MESSAGES:

IMP. ER> arb_or_resel(iid)
I/O Time-out Occurred Date/Time Stamp

IMP. ER> arb_or_resel(iid)
SCSI Reset Occurred Date/Time Stamp

NAME

autosense - set or reset autosense flag

SYNOPSIS

```
autosense(bit);
int bit;                        /* 0 = no autosense
                                   1 = autosense on check
                                       condition in I/O
                                       Driver */
```

DESCRIPTION

The **autosense()** function will set or reset automatic sense request flag. If enabled, each command resulting in a nonzero status function will have sense data requested for it and the results will be placed into the current sense buffer. The sense command issued by **autosense()** will execute only once and return an error if sense cannot be read.

Also see Section IODVR.3.9 .

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

awin_res - check for arbitration win by target and allow target to reselect

SYNOPSIS

```
return = awin_res(iid);
unsigned return;              /* function return */
BYTE iid;                     /* initiator ID num */
```

DESCRIPTION

awin_res() combines two functions: **arbwin()** and **resel()**. The purpose of combining these functions into one is to allow the user to step through a Stand-Alone Test without causing the controller to detect a reselect time-out between the time the **arbwin()** completes and the time the user executes the **resel()** function. Other than this timing difference, a call to **awin_res(iid)** is functionally identical to a call to **arbwin(iid)** followed by a call to **resel()**.

DEFAULT VALUE: N.A.

RETURNS:

Ø x ØØ successful - target has reselected the host
Øx Ø5 reselect time-out
ØxØ9 SCSI bus reset detected
Øx21 host won arbitration

NOTE: In this case, the host will release the bus immediately after it sees that it has won arbitration; by the time the function has returned, the target will probably have won arbitration.

EXECUTION TYPE: Microprogramming

ERROR MESSAGES:

IMP. ER> awin_res(iid) I/O Time-out Occurred	Date/Time Stamp
IMP. ER> awin_res(iid) SCSI Reset Occurred	Date/Time Stamp
IMP. ER> awin_res(iid) Arbitration won by host adapter	Date/Time Stamp
IMP. ER> awin_res(iid) Host Adapter Lost and Bus Not Busy	Date/Time Stamp
IMP. ER> awin_res(iid) Invalid Reselection Sequence	Date/Time Stamp
IMP. ER> awin_res(iid) Function Time-Out	Date/Time Stamp

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME

bcu - enable/disable buffer and command frame update

SYNOPSIS

```
bcu(bit);
int bit;                        /*  Ø = no update
                                    1 = update buffer and
                                        command frames */
```

DESCRIPTION

This function will enable or disable updates to the buffer and command frames in the I/O Driver Status Window. Listed below are the fields that are updated when this function is enabled:

```
Buffer Frame:
    Wr/Ref (write/reference buffer and address)
    Rd Buf (read buffer and address)
SCSI Command Frame:
    CDB (SCSI command bytes)
    status (SCSI current and previous status)
    sense (SCSI sense bytes)
    xfer (data transfer mode)
    a.s. (autosense)
    s.l. (state log)
    arb. (arbitration mode)
    sel. (select mode)
    b.p. (bus parity)
    b.w. (busywait)
    iid (initiator ID)
    tid (target ID)
```

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

bfreearm - bus free detection logic arm

SYNOPSIS

bfreearm();

DESCRIPTION

This function will arm the test adapter bus free detection logic such that it will detect any bus free when the TARGET releases the bus. This function should be called in advance of a known disconnect or command complete message to catch the bus free condition as soon as it occurs. **bfreeck()** works in conjunction with the **bfreearm()** to verify a bus release since the last **breearm()** execution.

> **NOTE:** Arbitration functions are not allowed between **breearm()** and **bfreeck()**.

DEFAULT VALUE: N.A.

RETURNS: N.A.

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

EXAMPLE: Check for bus free after command completion

```
arb2();
sel4();
cdb61();
dataout1();
statin();
bfreearm();          /* set up to catch bus free */
msgin();             /* command complete message */
delayms(n);          /* n msec delay to allow TARGET to
                        release bus */
bfreeck();           /* check to see if the bus has gone free
                        at any time since the bfreearm() */
```

NAME
bfreeck - bus free detection
(determines if bus has been released by TARGET)

SYNOPSIS
```
error = bfreeck();
unsigned error;                /* return status */
```

DESCRIPTION
Determines if the bus has gone free since the last **bfreeck()** (i.e., the TARGET has released the bus). **bfreeck()** requires **bfreearm()** be called prior to the bus free event. If the bus has gone free, **bfreeck()** returns Ø and if the bus has not gone free it returns a Øx22. It is possible that a delay will be required from the disconnect or command complete message **msgin()** test and **bfreeck()**. This function does not check for a current bus free condition, but for whether a bus free has been detected since the **bfreearm()** function was executed. Therefore, this function could return a bus free condition but indicating a previous bus free.

DEFAULT VALUE: N.A.

RETURNS:
ØxØØ bus free
Øx22 bus busy

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:
Initiator Status Byte
ØxØØ bus free
Øx22 bus busy

ERROR MESSAGES:
IMP. ER> bfreeck()
SCSI Bus Not Gone Free   Date/Time Stamp

NAME

blk_size - define block size of random access device transfers

SYNOPSIS

```
return = blk_size(size);
unsigned return;                    /* return size */
unsigned size;                      /* blocks size in bytes */
```

DESCRIPTION

This function sets the block size to be used by **dmaset_vblk()** to calculate a virtual memory address from a starting block number. This function is not necessary unless a pointer into the virtual buffer space needs to be generated.

DEFAULT VALUES: NONE

RETURNS: defined block size

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: N.A.

EXAMPLE:

```
blk_size(0x100);                    /* block size of 0x100 */
random_blk(0x0L,0x2000L);           /* random block length */
random_len(1,0x0x1000);             /* random transfer length */
dmaset_vblk("W");                   /* set virtual starting
                                       address */
readr10_blk();                      /* read */
```

NAME
busrel - release bus
(release the test adapter asserted BUSY)

SYNOPSIS
busrel();

DESCRIPTION
busrel() releases all assertions of BUSY by the test adapter. These include both the arbitration logic and the PIO BUSY path (usually used in conjunction with **arblose()**).

This function may be used in conjunction with the **forcbusy()** function to drop BUSY in order to allow the TARGET to reselect the HOST after arbitration is lost by the TARGET.

DEFAULT VALUE: N.A.

RETURNS: NONE

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:
Initiator Status Byte
0x00 good completion

ERROR MESSAGES: N.A.

NAME
busywait - set or reset busywait flag

SYNOPSIS
```
busywait(bit);
int bit;                          /* Ø = no busywait
                                     1 = busywait in I/O Driver
                                  */
```

DESCRIPTION
The **busywait()** function will set or reset the busywait flag. This flag is an I/O Driver option to wait for the target to become not BUSY within the time-out limits set by the **ioto()** function.

Also see Section IODVR.3.8 .

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

bus_logen - enable/disable SCSI bus state logging

SYNOPSIS

```
bus_logen(bit);
int bit;                        /* 0 = no logging
                                   1 = SCSI state logging */
```

DESCRIPTION

Enables or disables SCSI bus state logging. If enabled, each phase change that occurs on the SCSI bus (with the exception of phases in which an explicit error occurs) or bus events will be recorded into a FIFO. This information can be used to debug SCSI bus problems.

Also see Section STLOG.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

bytcmp - check the number of bytes compared

SYNOPSIS

```
return = bytcmp(minL,maxL);
int return;                    /* return code */
unsigned long minL;            /* minimum value */
unsigned long maxL;            /* maximum value */
```

DESCRIPTION

Checks the number of bytes compared to be within the 'minL' and 'maxL' limits. If the number is out of the specified range, an explicit error message is generated.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) number within range
1 number out of range

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

EXP. ER> bytcmp(121f50,122200)
Bytes Compared Out of Range, Bytes Compared = 300148

NAME

bytrd - check the number of bytes read

SYNOPSIS

```
return = bytrd(minL,maxL);
int return;                     /* return code */
unsigned long minL;             /* minimum value */
unsigned long maxL;             /* maximum value */
```

DESCRIPTION

Compares the number of bytes read with the 'minL' and 'maxL' limits. If the number is out of the specified range, an explicit error message is generated.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(0)  number within range
      1  number out of range
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> bytrd(121f50,122200)
Bytes Read Out of Range, Bytes Read = 121f00
```

NAME

bytwrt - check the number of bytes written

SYNOPSIS

```
return = bytwrt(minL,maxL);
int return;                       /* return code */
unsigned long minL;               /* minimum value */
unsigned long maxL;               /* maximum value */
```

DESCRIPTION

Compares the number of bytes written with the 'minL' and 'maxL' limits. If the number is out of the specified range, an explicit error message is generated.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(0)  number within range
      1  number out of range
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> bytwrt(121f50,122200)
Bytes Written Out of Range, Bytes Written = 122204
```

NAME

ccs_modsel - Common Command Set mode select command

SYNOPSIS

```
return = ccs_modesel(list_len,sp);
unsigned return;              /* return code */
unsigned list_len;            /* parameter list length */
unsigned sp;                  /* save parameters bit */
```

DESCRIPTION

This function will form and execute the command descriptor block for the mode select command as defined in the CCS version of SCSI. This function is the same as **mode_sel()** with the addition of the save parameters bit.

COMMAND DESCRIPTOR BLOCK FOR CCS MODE SELECT COMMAND

bit byte	7	6	5	4	3	2	1	0
0	15							
1	lun(lun);			00				SP
2	00							
3	00							
4	list_len							
5	cntlbyte(byte);							

For a complete description of the command refer to the Common Command Set (CCS) version of the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful completion
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

ccs_modsens - Common Command Set mode sense command

SYNOPSIS

```
return = modsens(len,pcf,pagecode);
unsigned return;              /* return code */
unsigned len;                 /* allocation length */
unsigned pcf;                 /* page control field bits */
unsigned pagecode;            /* page code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the mode sense command as defined in the CCS version of SCSI. This function is the same as **modesen()** with the addition of the 'pcf' and 'pagecode' fields.

COMMAND DESCRIPTOR BLOCK FOR CCS MODE SENSE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	1A							
1	lun(lun);			00				
2	PCF		PAGE CODE					
3	00							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the Common Command Set (CCS) version of the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful completion
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

cdb61 - 6-byte SCSI command transfer via DMA transfer
cdb1Ø1 - 1Ø-byte SCSI command transfer via DMA transfer
cdb121 - 12-byte SCSI command transfer via DMA transfer

SYNOPSIS

```
return = cdb61(bØ,b1,b2,b3,b4,b5);
BYTE bØ -> b5: SCSI Command Bytes

return = cdb1Ø1(bØ,b1,b2,b3,b4,b5,b6,b7,b8,b9);
BYTE bØ -> b9: SCSI Command Bytes

return = cdb121(bØ,b1,b2,b3,b4,b5,b6,b7,b8,b9,b1Ø,b11);
BYTE bØ -> b11: SCSI Command Bytes

int return;
```

DESCRIPTION

Transfers the n-byte command from the INITIATOR to the TARGET utilizing the backplane DMA. The function will return with good completion if n, and only n bytes of command are requested by the TARGET. If less than n bytes are requested, the function returns with an error code of ØxØC. As soon as the n bytes have been transferred, the function returns. After completion (good or bad), the number of command bytes transferred can be accessed as the function statistics "bytes written" field. (The global bytes written counter is not incremented by this amount.)

DEFAULT VALUE: N.A.

RETURNS:

ØxØØ if all n bytes transferred
ØxØ9 SCSI reset detected
ØxØD invalid bus free detected
ØxØ5 I/O time-out
ØxØC invalid phase change occurred

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

ØxØØ if all n bytes transferred
ØxØ9 SCSI reset detected
ØxØD invalid bus free detected
ØxØ5 I/O time-out
ØxØC invalid phase change occurred

ERROR MESSAGES:

```
IMP. ER> cdb61(0A,00,00,00,01,00)
Unexpected Phase Change
Four bytes transferred                            Date/Time Stamp

IMP. ER> cdb61(0A,00,00,00,01,00)                 Date/Time Stamp
SCSI Reset Occurred

IMP. ER> cdb61(0A,00,00,00,01,00)                 Date/Time Stamp
SCSI I/O Time-out Occurred

IMP. ER> cdb61(0A,00,00,00,01,00)                 Date/Time Stamp
SCSI I/O Invalid Bus Free Occurred
```

NAME

cdb62 - 6-byte SCSI command transfer via TR transfer
cdbl02 - 10-byte SCSI command transfer via TR transfer
cdbl22 - 12-byte SCSI command transfer via TR transfer

SYNOPSIS

```
return = cdb62(b0,b1,b2,b3,b4,b5);
BYTE b0 -> b5: SCSI Command Bytes

return = cdbl02(b0,b1,b2,b3,b4,b5,b6,b7,b8,b9);
BYTE b0 -> b9: SCSI Command Bytes

return = cdbl22(b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11);
BYTE b0 -> b11: SCSI Command Bytes

int return;
```

DESCRIPTION

Transfers the n-byte command from the INITIATOR to the TARGET utilizing the test adapter transmit/receive state machine. The function will return with good completion if n, and only n bytes of command are requested by the TARGET. If less than n bytes are requested, the function returns with an error code of 0x0C. As soon as the n bytes have been transferred, the function returns. After completion (good or bad), the number of command bytes transferred can be accessed as the function statistics "bytes written" field. (The global bytes written counter is not incremented by this amount.)

DEFAULT VALUE: N.A.

RETURNS:

0x00 if all n bytes transferred
0x09 SCSI reset detected
0x0D invalid bus free detected
0x05 I/O time-out
0x0C invalid phase change occurred

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

0x00 if all n bytes transferred
0x09 SCSI reset detected
0x0D invalid bus free detected
0x05 I/O time-out
0x0C invalid phase change occurred

ERROR MESSAGES:

```
   IMP. ER> cdb62(ØA,ØØ,ØØ,ØØ,Ø1,ØØ)
   Unexpected Phase Change
   Four Bytes Transferred                              Date/Time Stamp

   IMP. ER> cdb62(ØA,ØØ,ØØ,ØØ,Ø1,ØØ)
   Additional Command Byte Requested                   Date/Time Stamp
```

NAME

cdb63 - 6-byte SCSI command transfer via PIO Transfer
cdbl03 - 10-byte SCSI command transfer via PIO Transfer
cdbl23 - 12-byte SCSI command transfer via PIO Transfer

SYNOPSIS

```
return = cdb63(b0,b1,b2,b3,b4,b5);
BYTE b0 -> b5: SCSI Command Bytes

return = cdbl03(b0,b1,b2,b3,b4,b5,b6,b7,b8,b9);
BYTE b0 -> b9: SCSI Command Bytes

return = cdbl23(b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11);
BYTE b0 -> b11: SCSI Command Bytes

int return;
```

DESCRIPTION

Transfers the n-byte command from the INITIATOR to the TARGET utilizing the test adapter Programmed I/O. The function will return with good completion if n, and only n bytes of command are requested by the TARGET. If less than n bytes are requested, the function returns with an error code of 0x0C. As soon as the n bytes have been transferred, the function returns. After completion (good or bad), the number of command bytes transferred can be accessed as the function statistics "bytes written" field. (The global bytes written counter is not incremented by this amount.)

DEFAULT VALUE: N.A.

RETURNS:

0x00 if all n bytes transferred
0x09 SCSI reset detected
0x0D invalid bus free detected
0x05 I/O time-out
0x0C invalid phase change occurred

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

0x00 if all n bytes transferred
0x09 SCSI reset detected
0x0D invalid bus free detected
0x05 I/O time-out
0x0C invalid phase change occurred

ERROR MESSAGES:

```
   IMP. ER> cdb63(0A,00,00,00,01,00)
   Unexpected Phase Change
   Four Bytes Transferred                          Date/Time Stamp

   IMP. ER> cdb63(0A,00,00,00,01,00)
   Additional Command Byte Requested               Date/Time Stamp
```

NAME

chk_user_limits - check limits on **user_input()**

SYNOPSIS

```
return = chk_user_limits(lo,hi);
int return;                      /* return code */
int lo;                          /* low limit to check */
int hi;                          /* upper limit to check */
```

DESCRIPTION

Checks to see if the current **user_input()** integer is within limits defined by 'lo' and 'hi.' If out of range, an explicit error will be generated.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful, string matches
      1  error, string does not match
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGE

```
EXP. ER> chk_user_limits(1Ø,25)
User Value (3Ø) Out of Limits                 Date/Time Stamp
```

NAME

chk_user_string - check for match in **user_input()**

SYNOPSIS

```
return = chk_user_string("ref_string");
int return;                    /* return code */
char *ref_string;              /* string to be compared */
```

DESCRIPTION

Check to see if the current **user_input()** string matches the reference string. An explicit error is generated when there is no match.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(0)  successful, string matches
      1  error, string does not match
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> chk_user_string("string1")
User String Does Not Match
User Response: string2                         Date/Time Stamp
```

NAME

cmd_tail_bol - search the command tail for the string and return a logical 1, if string is found and a Ø, if it is not found

SYNOPSIS

```
flag_true = cmd_tail_bol("string");
int flag_true;                  /* return flag */
char *string;                   /* string to match */
```

DESCRIPTION

The command tail boolean function searches the command tail for a match with the passed string. If a match is found the function returns a 1. If a match is not found, the function returns a Ø.

DEFAULT VALUE: N.A.

RETURNS:

ØxØ if boolean not found in SAT command tail line
Øx1 if boolean found

ERROR MESSAGES: NONE

NAME

cmd_tail_string - search the command tail for the "look_for" string and return the parameter which follows the string. The parameter is defined as the word or the string bounded by " " following the "look_for" string.

SYNOPSIS

```
flag_true = cmd_tail_string("look_for","return_parameter");
int flag_true;                  /* return flag */
char *look_for;                 /* string to search */
char *return_parameter;         /* word or string following
                                   the "look_for" string */
```

DESCRIPTION

The command tail string function searches the command tail for a match with the "look_for" string. If a match is found, the function returns a 1 and returns the word or string following the "look_for" string as "return_parameter". If no match is found, the function returns as a 0 and "return_parameter" is not modified.

DEFAULT VALUE: N.A.

RETURNS:

0x0 if "look_for" string not found

0x1 if "look_for" string found and "return_parameter" string will contain the following word or string

ERROR MESSAGES: NONE

NAME

cntlbyte - set SCSI command control byte for SCSI commands

SYNOPSIS

```
cntlbyte(byte);
unsigned char byte;          /* set SCSI control byte */
                             /* last byte in command */
```

DESCRIPTION

This function sets the SCSI control byte (last byte of the SCSI CDB) which is generated by the I/O Driver. The control byte may be vendor-unique so check the target's manual to find the correct control byte value.

DEFAULT VALUE: 0x00

RETURNS: N.A.

EXECUTION TYPE: I/O Driver

ERROR MESSAGES: NONE

NAME

comp - compare command (10-byte command)

SYNOPSIS

```
return = comp(lenL);
unsigned return;              /* return code */
unsigned long lenL;           /* parameter list length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 10-byte compare command.

COMMAND DESCRIPTOR BLOCK FOR 10-BYTE COMPARE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	39							
1	lun(lun);			00				
2	00							
3	lenL (MSB)							
4	lenL							
5	lenL (LSB)							
6	00							
7	00							
8	00							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUSUPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

compwr - compare write and read buffers

SYNOPSIS

```
return = compwr(st_add,len);
int return;                          /* return code */
unsigned st_add;                     /* compare starting address */
unsigned len;                        /* length of compare
                                        0x0000 = 64K */
```

DESCRIPTION

Compares the write and read buffers. This function assumes that the buffers are backplane starting at the given address, 'st_add', for the defined 'len.' If 'st_add'+'len' exceeds length of the write, the compare is to the end of the buffer.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(0)  successful, compared
      1  error, not compared
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> compwr(100,1280)
Read Buffer Not Open                                  Date/Time Stamp

EXP. ER> compwr(1028,512)
Write Buffer Not Open                                 Date/Time Stamp

EXP. ER> compwr(2460,0)
Invalid Starting Address                              Date/Time Stamp

EXP. ER> compwr(0,0)
Compare Error: 1st Error @ 0020;  Wr/Ref = 04;  Rd = 02
               Total Bytes in error = 0100   Date/Time Stamp
```

NAME

copy - copy command

SYNOPSIS

```
return = copy(lenL);
unsigned return;            /* return code */
unsigned long lenL;         /* parameter list length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the copy command.

COMMAND DESCRIPTOR BLOCK FOR COPY COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	18							
1	lun(lun);			ØØ				
2	lenL (MSB)							
3	lenL							
4	lenL (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

copyver - copy and verify command (10-byte command)

SYNOPSIS

```
return = copyver(bytck,lenL);
unsigned return;            /* return code */
unsigned int bytck;         /* byte check bit */
unsigned long lenL;         /* parameter list length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 10-byte copy and verify command.

COMMAND DESCRIPTOR BLOCK FOR 10-BYTE COPY AND VERIFY COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	3A							
1	lun(lun);			0			bytck	0
2	00							
3	lenL (MSB)							
4	lenL							
5	lenL (LSB)							
6	00							
7	00							
8	00							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

copy_user_string - copy **user_input()** string to specified string

SYNOPSIS

```
copy_user_string("tgt_string");
char *tgt_string;              /* string to copy user_input()
                                  string */
```

DESCRIPTION

This function copies the last string entered by the **user_input()** function. The string returned from this function is not defined if **user_input()** was not called (with a string argument).

DEFAULT VALUES: N.A.

RETURNS: N.A

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

datain0 - Data In to Test Adapter High-Speed Buffer
datain1 - Data In to backplane memory via DMA transfer
datain2 - Data In to backplane memory via TR transfer
datain3 - Data In to backplane memory via PIO transfer

SYNOPSIS

```
datain0(countL,mode);
datain1(countL,mode);
datain2(countL,mode);
datain3(countL,mode);

unsigned long countL;   /* number of bytes to transfer */
int mode;               /* message mode
                           0 = error on any phase change
                           1 = accept save data pointers,
                               disconnect messages, support
                               reselection sequences to data
                               phase continuation.  Return
                               error on any other type of
                               phase change
                           2 = accept save data pointers,
                               disconnect messages, support
                               reselection sequences to data
                               phase continuation.  Return
                               without error on any other
                               type of phase change
                           3 = return on any phase change
                               without error */
```

DESCRIPTION

Transfers the specified number of bytes from the TARGET into the test adapter On-Board Buffer. With the message mode set to 0, any phase change will cause an implicit error. With the mode set to 1, the function will handle the disconnect/reconnect sequence which returns to a DATA OUT phase. Any other phase change prior to completion will cause an implicit error message. If the mode is set to 2, the function will handle all disconnect/reconnect sequences and will terminate with good completion if the desired number of bytes has been transferred or a phase change (other than for disconnection) occurs. A mode of 3 will return without error on any phase change.

> **NOTE:** The **xfermode()** function must be executed to open the correct buffer.

DEFAULT VALUE: N.A.

RETURNS:

0x0000 requested number of bytes transferred (mode 0 or 1)
requested number of bytes transferred or phase change occurred (mode 2 or 3)
0x0009 SCSI bus reset detected

~datain0 ~datain1 ~datain2 ~datain3

```
     0x000D  invalid bus free detected
     0x0200  no active buffer
     0x0005  I/O time-out
     0x000C  invalid phase change
     0x0011  nonsupported message
     0x000B  reselection aborted
```

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

```
     Initiator Status Byte:
          0x00  good completion
          0x09  SCSI bus reset detected
          0x0D  invalid bus free detected
          0x05  I/O time-out
          0x0C  invalid phase change
          0x11  nonsupported message
          0x0B  reselection aborted
          0x0F  SCSI inbound parity error
     I/O Status Byte:
          0x02  no active buffer
```

ERROR MESSAGES:

```
   IMP. ER> datain0(0x10000L,0)
   No Data In Phase
   0 Bytes Transferred                                     Date/Time Stamp

   IMP. ER> datain0(0x10000L,1)
   Unexpected Phase Change
   1234 Bytes Transferred                                  Date/Time Stamp

   IMP. ER> datain0(0x10000L,2)
   No Disc./Reconnect Messages
   1234 Bytes Transferred                                  Date/Time Stamp

   IMP. ER> datain0(0x10000L,2)
   No Active Buffer                                        Date/Time Stamp
```

NAME

datain4 - Compare SCSI data with Test Adapter High-Speed Buffer
datain5 - Compare SCSI data with backplane memory

SYNOPSIS

```
datain4(countL,mode);
datain5(countL,mode);

unsigned long countL;   /* Number of bytes to transfer */
int mode;               /* message mode
                           0 = error on any phase change
                           1 = accept save data pointers,
                               disconnect messages, support
                               reselection sequences to data
                               phase continuation.  Return
                               error on any other type of
                               phase change
                           2 = accept save data pointers,
                               disconnect messages, support
                               reselection sequences to data
                               phase continuation.  Return
                               without error on any other
                               type of phase change
                           3 = return on any phase change
                               without error */
```

DESCRIPTION

Transfers the specified number of bytes from the TARGET and compares (on-the-fly) with the test adapter On-Board Buffer. With the message mode set to 0, any phase change will cause an implicit error. With the mode set to 1, the function will handle the disconnect/reconnect sequence which returns to a DATA OUT phase. Any other phase change prior to completion will cause an implicit error message. If the mode is set to 2, the function will handle all disconnect/reconnect sequences and will terminate with good completion if the desired number of bytes has been transferred or a phase change (other than for disconnection) occurs. A mode of 3 will return without error on any phase change.

Only the first miscompare will be reported, after that data will be compared on-the-fly without any further miscompare messages.

> **NOTE:** The **xfermode()** function must be executed to open the correct buffer.

DEFAULT VALUE: N.A.

RETURNS:

Øx0000	requested number of bytes transferred (mode Ø or 1) requested number of bytes transferred or phase change occurred (mode 2 or 3)
Øx0009	SCSI bus reset detected
Øx000D	invalid bus free detected
Øx0200	no active buffer
Øx0005	I/O time-out
ØxFF00	I/O aborted (Data Compare Error)
Øx000C	invalid phase change
Øx0011	nonsupported message
Øx000B	reselection aborted

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

Øx00	good completion
Øx09	SCSI bus reset detected
Øx0D	invalid bus free detected
Øx05	I/O time-out
Øx0C	invalid phase change
Øx11	nonsupported message
Øx0B	reselection aborted
Øx0E	buffer miscompare
Øx0F	SCSI inbound parity error

I/O Status Byte:

Øx02	no active buffer
ØxFF	I/O aborted (Data Compare Error)

ERROR MESSAGES:

```
IMP. ER> datain4(Øx10000L,Ø)
No Data In Phase
Ø Bytes Transferred                              Date/Time Stamp

IMP. ER> datain4(Øx10000L,1)
Unexpected Phase Change
1234 Bytes Transferred                           Date/Time Stamp

IMP. ER> datain4(Øx10000L,2)
No Disc./Reconnect Messages
1234 Bytes Transferred                           Date/Time Stamp

IMP. ER> datain4(Øx10000L,2)
No Active Buffer                                 Date/Time Stamp

IMP. ER> datain4(Øx10000L,2)
Actual Data Ø3, Expected Data Ø7
Actual Data ØC, Expected Data Ø8
Actual Data Ø8, Expected Data Ø9
Data Compare Error Occurred                      Date/Time Stamp
```

NAME

dataoutØ - Data Out from Test Adapter High Speed Buffer
dataout1 - Data Out from backplane memory via DMA transfer
dataout2 - Data Out from backplane memory via TR transfer
dataout3 - Data Out from backplane memory via PIO transfer

SYNOPSIS

```
error = dataoutØ(countL,mode);
error = dataout1(countL,mode);
error = dataout2(countL,mode);
error = dataout3(countL,mode);

unsigned long countL;   /* Number of bytes to transfer */
int mode;               /* message mode
                           Ø = error on any phase change
                           1 = accept save data pointers,
                               disconnect messages, support
                               reselection sequences to data
                               phase continuation.  Return
                               error on any other type of
                               phase change
                           2 = accept save data pointers,
                               disconnect messages, support
                               reselection sequences to data
                               phase continuation.  Return
                               without error on any other
                               type of phase change
                           3 = return on any phase change
                               without error */
```

DESCRIPTION

Transfers the specified number of bytes from the SDS-1 test adapter On-Board Buffer to the TARGET. With the disconnect mode set to Ø, any phase change will cause an implicit error. With the mode set to 1, the function will handle the disconnect/reconnect sequence which will return to the DATA OUT phase. Any other phase change prior to completion will cause an implicit error message. If the mode is set to 2, the function will handle all disconnect/reconnect sequences and will terminate with good completion if the desired number of bytes have been transferred or a phase change other than for disconnection occurs. (This feature is valuable for completion of a data transfer which was intentionally interrupted like for a parity error check.) A mode of 3 will return without error on any phase change.

> **NOTE:** The **xfermode()** function must be executed to open the correct buffer.

DEFAULT VALUE: N.A.

~dataoutØ ~dataout1 ~dataout2 ~dataout3

RETURNS:

Øx0000	requested number of bytes transferred (mode Ø or 1) requested number of bytes transferred or phase change occurred (mode 2 or 3)
Øx0009	SCSI bus reset detected
Øx000D	invalid bus free detected
Øx0200	no active buffer
Øx0C00	invalid phase change
Øx0011	nonsupported message
Øx000B	reselection aborted

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

Øx00	good completion
Øx09	SCSI bus reset detected
Øx0D	invalid bus free detected
Øx0C	invalid phase change
Øx11	nonsupported message
Øx0B	reselection abort
Øx0F	SCSI inbound parity error

I/O Status Byte:

Øx02	no active buffer

ERROR MESSAGES:

```
IMP. ER> dataoutØ(Øx1ØØØØL,Ø)
No Data Out Phase
Ø Bytes Transferred                                 Date/Time Stamp

IMP. ER> dataoutØ(Øx1ØØØØL,1)
Unexpected Phase Change
1234 Bytes Transferred                              Date/Time Stamp

IMP. ER> dataoutØ(Øx1ØØØØL,2)
No Disc./Reconnect Messages
1234 Bytes Transferred                              Date/Time Stamp

IMP. ER> dataoutØ(Øx1ØØØØL,2)
No Active Buffer                                    Date/Time Stamp
```

NAME

debug - set debug level

SYNOPSIS

```
debug(level);
int level;                    /* debug level */
```

DESCRIPTION

This function will halt execution and enter the Debugger (with the current display format). At this point the user may perform any Debugger **TRACE** State command.

The Debugger Skip command will cause the function to be skipped and the debug level to remain unchanged.

The following is a brief description of the effects of each debug level:

LEVELS	DESCRIPTION
Ø	Disable Debugger and run at full speed
1, 2, 3	Enable Debugger and stop on next instruction with debug level 1, 2 or 3.

Changing the debug level will also repaint the screen, causing the Trace Display to be cleared.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME
delayms - generate a delay specified in milliseconds

SYNOPSIS
```
delayms(ms_delay);
int ms_delay;                   /* number of milliseconds to
                                   delay */
```

DESCRIPTION
Generates a delay equal to the number of milliseconds requested by the user.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

delays - generate a delay specified in seconds

SYNOPSIS

```
delays(sec_delay);
int sec_delay;                  /* number of seconds to delay
                                */
```

DESCRIPTION

Generates a delay equal to the number of seconds requested by the user.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

delta_time - obtain the real time elapsed between 2 bus state log entries

SYNOPSIS

```
return = delta_time("state1",count1,"state2",count2);
unsigned long return;        /* elapsed time in
                                microseconds */
char *state1;                /* state description */
int count1;                  /* # of "state1" occurrences
                             */
char *state2;                /* state description */
int count2;                  /* # of "state2" occurrences
                             */
```

DESCRIPTION

This function looks backward in the bus state log from the current time for 'count1' occurrences of "state1." It then looks forward in the state log for 'count2' occurrences of "state2" and returns the elapsed time between these two events in microseconds. The search backward for "state1" stops at the entry indicating test initialization. A return of zero indicates an error; **get_f_status("IO")** must be called to determine the type of error (these error codes are defined below under I/O Status). Below is a definition of the values of "state1" and "state2" strings.

"ARB_START"	--> start of arbitration
"ARB_END"	--> completion (success) of arbitration
"SEL_ASSERT"	--> assertion of SEL by HOST
"SEL_RESPONSE"	--> response to SEL by TARGET (BSY assertion)
"CMD_START"	--> detection of COMMAND OUT phase
"CMD_END"	--> transfer of last command byte complete
"DATAIN"	--> detection of DATA IN phase
"DATAOUT"	--> detection of DATA OUT phase
"RESEL"	--> reselection complete
"MSG_OUT"	--> detection of MESSAGE OUT phase
"MSG_IN"	--> detection of MESSAGE IN phase
"STATUS"	--> detection of STATUS IN phase
"BUS_FREE"	--> detection of BUS FREE (BSY, SEL false)
"RESET_DET"	--> detection of SCSI reset not generated by SDS-1
"RESET_ASSRT"	--> reset asserted by SDS-1
"TEST_INIT"	--> commencement of execution of Stand-Alone Test

DEFAULT VALUE: N.A.

RETURNS:

ØL	Error (see I/O status codes)
else	Returns elapsed time in microseconds

I/O Status:
    Øx4Ø specified value of "state1" not found
    Øx41 specified value of "state2" not found
    Øx42 illegal string specified for "state1" or "state2"

ERROR MESSAGES:

```
IMP. ER> delta_time(state1,count1,state2,count2);
State 1 not found                               Date/Time Stamp

IMP. ER> delta_time(state1,count1,state2,count2);
State 2 not found                               Date/Time Stamp

IMP. ER> delta_time(state1,count1,state2,count2);
Illegal state specifier                         Date/Time Stamp
```

NAME

dispbuf - display specified buffer to screen

SYNOPSIS

```
dispbuf("buffer",start_add,length);
char *buffer;                   /* buffer type to display */
unsigned start_add;             /* starting address */
unsigned length;                /* display length (in bytes)
                                */
```

DESCRIPTION

Generates a buffer display for the requested buffer to the screen. Below are the different buffer types that can be specified by "buffer":

"R"	Read Buffer
"W"	Write Buffer
"RW"	Read/Write Buffer
"OBB"	On-Board Buffer
"L"	Log Buffer
"S"	Sense Buffer

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

dmarst - reset current DMA pointer to start of buffer

SYNOPSIS

```
return = dmarst("r/w");
int return;                     /* return code */
char *r/w;                      /* read or write buffer */
```

DESCRIPTION

Resets the buffer DMA pointer of the current write or read buffer. If the requested buffer has not been assigned by a **xfermode()** function, an error is returned. Read or write operations leave their respective DMA pointer pointing to the next byte in the buffer so that subsequent operations will continue to fill (or read from) the buffer at the next address. However, there are times when it is necessary to reset the DMA pointer. This function does not change any values in the buffer itself.

When performing read and compare operations, the write buffer (also known as reference buffer) pointer must be reset or set to a known location.

When performing hardware compare operations, resetting the read buffer pointer will cause an error since there is no read buffer.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful completion

1 error occurred

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

EXP. ER> dmarst("R")
Read Buffer Not Open — Date/Time Stamp

EXP. ER> dmarst("w")
Write Buffer Not Open — Date/Time Stamp

EXP. ER> dmarst("i")
Invalid Argument — Date/Time Stamp

NAME

dmaset - set current DMA pointer to new value

SYNOPSIS

```
return = dmaset("r/w",address);
int return;                     /* return code */
char *r/w;                      /* "r" = read buffer
                                   "w" = write buffer */
int unsigned address;           /* address to set */
```

DESCRIPTION

Sets the current Write or Read DMA address pointer within the selected buffer (write or read) to the specified address (see **dmarst()**). If an error condition occurs, a value of 1 is returned, otherwise a NULL(Ø) value is returned. This function does not change any values in the buffer itself.

When performing read and compare operations, the write buffer (also known as reference buffer) pointer must be reset or set to a known location.

When performing hardware compare operations, setting the read buffer pointer will cause an error since there is no read buffer.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful completion
1 error occurred

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

EXP. ER> dmaset("R")
Read Buffer Not Open — Date/Time Stamp

EXP. ER> dmaset("w")
Write Buffer Not Open — Date/Time Stamp

EXP. ER> dmaset("i")
Invalid Argument — Date/Time Stamp

NAME

dmaset_va - set a virtual address

SYNOPSIS

```
returnL = dmaset_va("r_w",addressL);
unsigned long returnL;        /* returns a long address */
char *r_w;                    /* String defining buffer
                                 "R" = Read "W" = write */
unsigned long addressL;       /* virtual address (only 28
                                 lower bits are used) */
```

DESCRIPTION

This function defines a 28-bit virtual address to be used with write or read operations in the HSHCV transfer mode. In this mode, the On-Board Buffer can create a 2**28 bit nonrepeating pattern which can be viewed as a 256MB virtual memory. The **dmaset_va()** allows the user to set any address in this range for use with subsequent write and read operations.

When performing read and compare operations, the write buffer (also known as reference buffer) pointer must be reset or set to a known location.

When performing hardware compare operations, setting the read buffer pointer will cause an error since there is no read buffer.

DEFAULT VALUES: NONE

RETURNS:

new virtual DMA address (unsigned long)	successful
ØxFFFFFFFFL	error occurred

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

EXP. ER> dmaset_va("r_w",addressL);
Invalid Argument   Date/Time Stamp

EXP. ER> dmaset_va("r_w",addressL);
Buffer Not Open   Date/Time Stamp

NAME

dmaset_vblk - set a virtual address from block info

SYNOPSIS

```
returnL = dmaset_vblk("r_w");
unsigned long returnL;          /* addressL */
char *r_w;                      /* String defining buffer
                                   "R" = Read "W" = write */
```

DESCRIPTION

This function calculates a 28-bit virtual address to be used with write or read operations in the HSHCV transfer mode. The calculation is based on the block size established by **blk_size()** and the current starting block set by **set_blk()**, **inc_blk()** or **random_blk()**. If a value greater than 2**28 is calculated, only the lower 28 bits are used.

The user can create unique data for every block on a large disk by using a different seed in the **fillpr()** function for the second 256MB and yet a different seed for the third 256MB. This implies that the user must look at the current starting block (returned by **set_blk()**, **inc_blk()**, or **random_blk()** and decide if a new OBB fill pattern is required). The max number of bytes supported by the SDS-1 is 2**32 * 2**16 (start_block * block_size).

When performing read and compare operations, the write buffer (also known as reference buffer) pointer must be reset or set to a known location.

When performing hardware compare operations, setting the read buffer pointer will cause an error since there is no read buffer.

DEFAULT VALUES: NONE

RETURNS:

new virtual DMA address (unsigned long)	successful
ØxFFFFFFFFL	error occurred

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

EXP. ER> dmaset_vblk("I");
Invalid Argument   Date/Time Stamp

EXP. ER> dmaset_vblk("w");
Buffer Not Open   Date/Time Stamp

NAME

eea - set explicit error action

SYNOPSIS

```
eea("action");
char *action;                  /* error action to be taken on
                                  explicit errors */
```

DESCRIPTION

Sets the action to be taken on an explicit error. An explicit error is an error that requires an explicit test to determine that an error has occurred (such as a **esvalid()** function).

The error action types are defined below:

```
CONT - no action (ignore error)
HALT - in nonbatch mode:  halt and enter Debugger
       in batch mode:  exit to next SAT
LOGC - log error and continue up to the set_er_limits()
       function limit;  otherwise,
       in nonbatch mode:  enter the Debugger
       in batch mode:  exit to DOS
LOGH - in nonbatch mode:  log error and enter Debugger
       in batch mode:  log error and exit to DOS
```

CONT and HALT types are not available in the Menu Interface. Also see Sections SAT.5 and DEBUG.1.3 .

DEFAULT VALUE: LOGC

RETURNS: N.A.

ERROR MESSAGES:

```
IMP. ER> eea("LAGC");
Undefined Error Action Parameter                 Date/Time Stamp
```

NAME

erase - erase command

SYNOPSIS

```
return = erase(long);
unsigned return;                /* return code */
unsigned long;                  /* long bit */
```

DESCRIPTION

This function will form and execute the command descriptor block for the erase command.

COMMAND DESCRIPTOR BLOCK FOR ERASE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	19							
1	lun(lun);			0				long
2	00							
3	00							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

errdelay - enable/disable error delay

SYNOPSIS

```
return = errdelay(bit);
unsigned return;                  /* function return */
int bit;                          /* delay on/off
                                     Ø = off
                                     1 = on */
```

DESCRIPTION

This function allows the user to disable the 5-second delay which normally occurs when any implicit or explicit error is detected.

DEFAULT VALUE: Error delay enabled

RETURNS: Ø (always)

ERROR MESSAGES: NONE

NAME

error_ok - decrement error count

SYNOPSIS

```
return = error_ok("display");
unsigned return;                /* function return */
char *display;                  /* "NODSPL" --> no display
                                   "DISPLAY" --> display
                                      message on the console
                                      showing execution of
                                      this function */
```

DESCRIPTION

This function allows the errors to occur in a test which would normally generate a non-zero error count and hence cause the test to fail. Calling this function decrements the error count.

If the "display" string equals "DISPLAY", the following message will be displayed on the console:

```
******>ERROR OK<******
```

DEFAULT VALUE: N/A

RETURNS: Ø (always)

ERROR MESSAGES: NONE

NAME

eseom - extended sense end of media check

SYNOPSIS

```
return = eseom(n);
int return;                  /* return code */
int n;                       /* bit value to compare */
```

DESCRIPTION

Compares the end of media (EOM) bit in the current extended sense buffer with the 'n' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain extended sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful, values are equal
1	values are not equal
2	if not extended sense data
3	if no sense buffer open

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> eseom(1)
EOM Bit Reset                                   Date/Time Stamp

EXP. ER> eseom(Ø)
EOM Bit Set                                     Date/Time Stamp

EXP. ER> eseom(Ø)
Non-Extended Sense                              Date/Time Stamp

EXP. ER> eseom(1)
No Sense Buffer Open                            Date/Time Stamp
```

NAME
esfm - extended sense file mark check

SYNOPSIS
```
return = esfm(n);
int return;                         /* return code */
int n;                              /* bit value to compare */
```

DESCRIPTION
Compares the file mark bit in the current extended sense buffer with the 'n' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain extended sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful, values are equal
1	values are not equal
2	if not extended sense data
3	if no sense buffer open

ERROR MESSAGES:
EXPLICIT ERROR MESSAGES

```
EXP. ER> esfm(1)
Filemark Bit Reset                                  Date/Time Stamp

EXP. ER> esfm(Ø)
Filemark Bit Set                                    Date/Time Stamp

EXP. ER> esfm(Ø)
Non-Extended Sense                                  Date/Time Stamp

EXP. ER> esfm(1)
No Sense Buffer Open                                Date/Time Stamp
```

NAME

esili - extended sense illegal length indicator check

SYNOPSIS

```
return = esili(n);
int return;                    /* return code */
int n;                         /* bit value to compare */
```

DESCRIPTION

Compares the illegal length indicator bit in the current extended sense buffer with the 'n' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain extended sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful, values are equal
1 values are not equal
2 if not extended sense data
3 if no sense buffer open

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> esili(1)
Illegal Length Indicator Bit Reset              Date/Time Stamp

EXP. ER> esili(Ø)
Illegal Length Indicator Bit Set                Date/Time Stamp

EXP. ER> esili(Ø)
Non-Extended Sense                              Date/Time Stamp

EXP. ER> esili(1)
No Sense Buffer Open                            Date/Time Stamp
```

NAME

esinfob - extended sense information bytes compare

SYNOPSIS

```
return = esinfob(minL,maxL);
int return;                     /* return code */
long minL;                      /* minimum value */
long maxL;                      /* maximum value */
```

DESCRIPTION

Compares the information bytes in the SCSI extended sense buffer with the 'minL' and 'maxL' limits. If the bytes are out of the specified range, the explicit error action will be taken. The sense buffer must contain valid extended sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(0)  successful, bytes within range
      1  bytes out of range
      2  if not extended sense data
      3  if ADVALID false
      4  if no sense buffer open
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> esinfob(20321f50,2032200)
Information Bytes Not Valid (valid bit not set)

EXP. ER> esinfob(10000,ff000)
Information Bytes Out of Range, Info Bytes = 1002abe

EXP. ER> esinfob(30745,33200)
Non-Extended Sense                          Date/Time Stamp

EXP. ER> esinfob(2100,5000)
No Sense Buffer Open                        Date/Time Stamp
```

NAME

eskey - extended sense key check for equal

SYNOPSIS

```
return = eskey(value);
int return;                    /* return code */
unsigned int value;            /* comparison value */
```

DESCRIPTION

Compares the value of the sense key with the comparison value, after first checking to be sure the sense data is in fact extended sense.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful, values are equal
      1  values are not equal
      2  if not extended sense data
      4  no sense buffer open
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> eskey(Ø2)
Extended Sense Key Miscompared, Sense Key = Ø1

EXP. ER> eskey(ØØ)
Non-Extended Sense                              Date/Time Stamp

EXP. ER> eskey(Ø3)
No Sense Buffer Open                            Date/Time Stamp
```

NAME

eskeynot - extended sense key check for not equal

SYNOPSIS

```
return = eskeynot(value);
int return;                  /* return code */
unsigned int value;          /* comparison value */
```

DESCRIPTION

Compares the value of the sense key with the comparison value, after first checking to be sure the sense data is in fact extended sense.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful, values are equal
      1  values are not equal
      2  if not extended sense data
      3  no sense buffer open
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> eskeynot(Ø6)
Extended Sense Key Compared, Sense Key = Ø6

EXP. ER> eskeynot(Øf)
Non-Extended Sense                               Date/Time Stamp

EXP. ER> eskeynot(Ø4)
No Sense Buffer Open                             Date/Time Stamp
```

NAME

esvalid - extended sense valid check

SYNOPSIS

```
return = esvalid(n);
int return;                       /* return code */
int n;                            /* bit value to compare */
```

DESCRIPTION

Compares the valid bit in the current extended sense buffer with the 'n' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain extended sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful, values are equal
1	values are not equal
2	if not extended sense data
3	if no sense buffer open

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> esvalid(1)
Valid Bit Reset                                   Date/Time Stamp

EXP. ER> esvalid(Ø)
Valid Bit Set                                     Date/Time Stamp

EXP. ER> esvalid(Ø)
Non-Extended Sense                                Date/Time Stamp

EXP. ER> esvalid(1)
No Sense Buffer Open                              Date/Time Stamp
```

NAME

exp_status - set expected target status

SYNOPSIS

```
exp_status(value);
unsigned char value;          /* expected TARGET status value
                                 value after application of
                                 stat_mask() */
```

DESCRIPTION

This function sets what the expected TARGET status is after an I/O Driver operation. This value is compared against the target status after the target status is masked by the **stat_mask**() value. A nonzero result will cause the implicit error action to be taken.

Also see Section IODVR.4.1 .

DEFAULT VALUE: 0x00 (no errors allowed)

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

fail - print a Fail line on scrolling screen and in report, Date and Time Stamp line and increment ERRORLEVEL

SYNOPSIS

```
fail("fail_string");
char *fail_string;              /* message to be displayed by
                                   the error handler */
```

DESCRIPTION

The **fail()** function produces a report entry FAIL in the right-hand column of the output Test Results report along with a Date and Time stamp. The "fail_string" is a message that is displayed in the error handler. The internal ERRORLEVEL is incremented by **fail()**.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

fillbcb - fill buffer with block count byte

SYNOPSIS

```
return = fillbcb(st_byt,blk_len,st_add,len);
int return;                    /* return code */
int unsigned st_byt;           /* starting block count byte
                                  (ØØh - FFh range) */
int unsigned blk_len;          /* length of block */
int unsigned st_add;           /* buffer starting address */
int unsigned len;              /* number of bytes to fill */
```

DESCRIPTION

Fills the write buffer with data blocks which contain the byte block number as their data pattern. For example:

```
fillbcb(Øx23,Øx2ØØ,Ø,Øx8ØØ);

ØØØØ:  23 23 23  . . .  23
Ø1fØ:  23 23 23  . . .  23
Ø2ØØ:  24 24 24  . . .  24
Ø3fØ:  24 24 24  . . .  24
Ø4ØØ:  25 25 25  . . .  25
Ø5fØ:  25 25 25  . . .  25
Ø6ØØ:  26 26 26  . . .  26
Ø7fØ:  26 26 26  . . .  26
```

The block fill data will rollover at FFh to ØØh. The fill will be for the length specified or to the end of the fill buffer whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful completion
      1  if block size is zero
```

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> fillbcb(ØØ,1Ø,Ø5ØØ,Ø1ØØ)
Starting Address Greater Than Buffer Length     Date/Time Stamp

IMP. ER> fillbcb(fØ,1ØØ,Ø,1ØØØ)
Fill Buffer Not Set                             Date/Time Stamp
```

NAME
fillbcw - fill buffer with block count word

SYNOPSIS
```
return = fillbcw(st_wrd,blk_len,st_add,len);
int return;                  /* return code */
int unsigned st_wrd;         /* starting block count word
                                (0000h - FFFFh range) */
int unsigned blk_len;        /* length of block */
int unsigned st_add;         /* buffer starting address */
int unsigned len;            /* number of bytes to fill */
```

DESCRIPTION
Fills the write buffer with data blocks which contain the word block number as their data pattern. For example:

```
fillbcw(0x1000,0x100,0,0x400);

0000:  10 00 10 00 10 00  . . .  10 00
00f0:  10 00 10 00 10 00  . . .  10 00
0100:  10 01 10 01 10 01  . . .  10 01
01f0:  10 01 10 01 10 01  . . .  10 01
0200:  10 02 10 02 10 02  . . .  10 02
02f0:  10 02 10 02 10 02  . . .  10 02
0300:  10 03 10 03 10 03  . . .  10 03
03f0:  10 03 10 03 10 03  . . .  10 03
```

The block fill data will rollover at FFFFh to 0000h. The fill will be for the length specified or to the end of the fill buffer whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:
NULL(0) successful completion
1 if block size is zero

ERROR MESSAGES:
IMPLICIT ERROR MESSAGE

```
IMP. ER> fillbcw(0000,10,0500,0100)
Starting Address Greater Than Buffer Length     Date/Time Stamp

IMP. ER> fillbcw(fff0,100,0,1000)
Fill Buffer Not Set                             Date/Time Stamp
```

NAME

fillbyte - fill with specified byte

SYNOPSIS

```
return = fillbyte(char,st_add,len);
int return;                     /* return code/status */
char byte;                      /* fill byte */
unsigned st_add;                /* starting addr for fill */
unsigned len;                   /* length of fill */
```

DESCRIPTION

Fills the write buffer with given data byte.

The 'st_add' should be in the range of the buffer length.

DEFAULT VALUES: N.A.

RETURNS:

ØxFFFF error occurred
Øx0000 successful

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

IMPLICIT ERROR MESSAGES

IMP. ER> fillbyte(Ø,ØxFFFF,ØxEF)
Starting Address Greater than Buffer Size

IMP. ER> fillbyte(Ø,ØxFFFF,ØxEF)
Fill Buffer Not Set Date/Time Stamp

NAME

filld - fill buffer with decrementing pattern

SYNOPSIS

```
return = filld(st_byt,st_add,len);
int return;                       /* return code */
int unsigned st_byt;              /* starting count for
                                     decrementing pattern
                                     (FFh - 00h range) */
int unsigned st_add;              /* buffer starting address */
int unsigned len;                 /* number of bytes to fill */
```

DESCRIPTION

Fills the write buffer with a decrementing pattern starting with the byte count specified in 'st_byt' and rolling over at FFh to 00h. The fill will be for the length specified or to the end of the fill buffer, whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:

0 successful completion
1 error occurred

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> filld(00,0500,0100)
Starting Address Greater Than Buffer Length     Date/Time Stamp

IMP. ER> filld(f0,0000,1000)
Fill Buffer Not Set                             Date/Time Stamp
```

NAME

filli - fill buffer with incrementing pattern

SYNOPSIS

```
return = filli(st_byt,st_add,len);
int return;                     /* return code */
int unsigned st_byt;            /* starting count for
                                   incrementing pattern
                                   (ØØh - FFh range) */
int unsigned st_add;            /* buffer starting address */
int unsigned len;               /* number of bytes to fill */
```

DESCRIPTION

Fills the write buffer with an incrementing pattern starting with the byte count specified in 'st_byt' and rolling over at FFh to ØØh. The fill will be for the length specified or to the end of the fill buffer whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:

Ø successful completion
1 error occurred

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> filli(ØØ,Ø5ØØ,Ø1ØØ)
Starting Address Greater Than Buffer Length     Date/Time Stamp

IMP. ER> filli(fØ,ØØØØ,1ØØØ)
Fill Buffer Not Set                             Date/Time Stamp
```

NAME

fillk - fill buffer with constant pattern

SYNOPSIS

```
return = fillk("string",st_add,len);
int return;                    /* return code */
char *string;                  /* string containing up to
                                  20 hex characters to be
                                  repeated in the buffer */
int unsigned st_add;           /* buffer starting address */
int unsigned len;              /* number of bytes to fill */
```

DESCRIPTION

Fills the write buffer with the constant data pattern specified by the hex characters in "string". For example;

fillk("17e57",0,8); would result in the following:

0000: 17 E5 71 7E 57 17 E5 71

The number of hex digits specified in the "string" does not need to be even. Characters other than hex digits will result in an error condition. The fill will be for the length specified or to the end of the fill buffer whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

> **NOTE:** This function will ignore commas, spaces and semicolons in the "string" argument.

DEFAULT VALUE: N.A.

RETURNS:

0 successful completion
1 error occurred

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> fillk("3f, 56,c3;",0500,0100)
Starting Address Greater Than Buffer Length    Date/Time Stamp

IMP. ER> fillk("75 4f da 1",0000,1000)
Fill Buffer Not Set                            Date/Time Stamp

IMP. ER> fillk("14 35 3s 4",0030,0100)
Non-Hex Character in Fill Pattern              Date/Time Stamp

IMP. ER> fillk("d325 e3f5 23b5 3da1 fc95 4520",0000,0000)
More Than 20 Characters in Fill Pattern        Date/Time Stamp
```

NAME

fillpr - fill buffer with pseudo random pattern

SYNOPSIS

```
return = fillpr(seed,st_add,len);
int return;                      /* return code */
unsigned seed;                   /* seed for pseudo random
                                    pattern */
int unsigned st_add;             /* buffer starting address */
int unsigned len;                /* number of bytes to fill */
```

DESCRIPTION

Fills the write buffer with a pseudo random data pattern based on the seed supplied. (The pattern can always be recreated using the same seed.) The fill will be for the length specified, or to the end of the fill buffer, whichever comes first.

The 'seed' argument should be an integer value.

The 'st_add' argument should be in the range of the buffer length.

```
NOTE:  User Beware:  fillpr(5,Ø,1Ø);
                     fillpr(5,Ø,1Ø);
is not the same as:
                     fillpr(5,Ø,2Ø);
```

DEFAULT VALUE: N.A.

RETURNS:

Ø successful completion
1 error occurred

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> fillpr(ØØ16,Ø5ØØ,Ø1ØØ)
Starting Address Greater Than Buffer Length    Date/Time Stamp

IMP. ER> fillpr(8ØØØ,ØØØØ,1ØØØ)
Fill Buffer Not Set                            Date/Time Stamp
```

NAME

fixed - set or reset the fixed bit in SCSI sequential commands

SYNOPSIS

```
fixed(bit);
unsigned char bit;              /* fixed bit in sequential
                                   commands */
```

DESCRIPTION

Sets or resets the fixed bit in SCSI sequential commands.

DEFAULT VALUE: 1

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

forcbusy - force test adapter BUSY on bus

SYNOPSIS

```
return = forcbusy();
int return;
```

DESCRIPTION

The **forcbusy()** function allows the HOST to assert its BUSY signal on the SCSI bus in conjunction with TARGET's BUSY. This feature is used for arbitration test with **arblose()** and **arbwin()** functions. It holds the bus after TARGET disconnect to set up an arbitration test between test adapter arbitration machine and the reconnecting TARGET.

DEFAULT VALUE: N.A.

RETURNS:

- Ø busy has been asserted
- 1 TARGET busy was not detected and test adapter busy was not forced

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE: NONE

ERROR MESSAGES:

IMP. ER> forcbusy()
Bus Is Free, Cannot Assert Busy Date/Time Stamp

EXAMPLE:

Set up an arbitration experiment in which the disconnected controller will first lose arbitration and then win arbitration. (This example assumes the command is a READ with physical arm motion on a hard disk drive and TARGET ID = 4).

```
arb2();
sel4();          /* identify message for disconnect */
cdb61();
forcbusy();      /* assert busy with TARGET */
msgin(02);       /* save data pointer */
msgin(04);       /* disconnect message */
delayms(200);    /* delay 200 msec to allow TARGET to
                    reconnect */
arblose(7);      /* have test adapter arbitrate as ID = 7
                    which guarantees TARGET loss of
                    arbitration */
arbwin(3);       /* have test adapter arbitrate as ID=3
                    TARGET will win the arbitration */
resel();         /* verify valid reselection sequence */
msgin();         /* identify message */
msgin(03);       /* restore data pointer */
datainl();       /* read data */
statin();
bfreearm();      /* set up to catch bus free */
msgin();         /* command complete */
delayms(n);      /* n msec delay to allow TARGET to
                    release bus */
bfreeck();       /* check to see if bus is free */
```

NAME

forceattn - force SCSI bus attention

SYNOPSIS

```
forceattn(n);
int n;                         /* attention state
                                  1 = asserted
                                  0 = deasserted */
```

DESCRIPTION

This function forces the SCSI bus attention to the defined state. The **msgout()** function should be used with this function. **msgout()** will deassert attention during its handshake. **msgout_atnf()** will hold attention asserted and can be used for multiple byte messages.

DEFAULT VALUE: N.A.

RETURNS: N.A.

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE: NONE

ERROR MESSAGES: NONE

NAME

forcperr - force parity error

SYNOPSIS

```
forcperr(n);
BYTE n;                          /* byte count (req/ack count)
                                    until parity error is
                                    forced */
```

DESCRIPTION

Forces a parity error after number of byte counts output to TARGET, typically on information out from test adapter. Parity errors on incoming information can be simulated by simply generating a hardware error message out.

To force a parity error on a byte during transfer, the transfer may be initiated with a **dataoutx()** for n bytes. The transfer will stop after n bytes. The **forcperr()** function may then be executed and the transfer begun again with the **dataoutx()** function.

DEFAULT VALUE: N.A.

RETURNS: N.A.

EXECUTION TYPE: Microprogramming

STATISTIC/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

format - format unit command

SYNOPSIS

```
return = format(fd,cmpl,dflist,intrleave);
unsigned return;                /* return code */
unsigned int fd;                /* format data bit */
unsigned int cmpl;              /* complete list bit */
unsigned int dflist;            /* defect list format */
unsigned int intrleave;         /* interleave */
```

DESCRIPTION

This function will form and execute the command descriptor block for the format unit command.

COMMAND DESCRIPTOR BLOCK FOR FORMAT COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	04							
1	lun(lun);			fd	cmpl	dflist		
2	00							
3	intrleave (MSB)							
4	intrleave (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
get_byte - get byte from buffer/address

SYNOPSIS
```
return = get_byte("r_w_s",address);
int return;                  /* return code/byte returned
                             */
char *r_w_s;                 /* buffer reference string
                                "R" = read
                                "W" = write
                                "S" = sense */
unsigned address;            /* address to get data from */
```

DESCRIPTION
This function returns the requested byte from the write, read or sense buffer.

DEFAULT VALUES: N.A.

RETURNS:
ØxFFØØ error (less than Ø)
ØxØØBB successful, where BB = requested byte

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:
IMPLICIT ERROR MESSAGES

IMP. ER> get_byte(S,Øx38Ø) Date/Time Stamp
Address Greater than Buffer Size

IMP. ER> get_byte(R,Øx38Ø) Date/Time Stamp
Buffer Not Open

IMP. ER> get_byte(T,Øx38Ø) Date/Time Stamp
Invalid Buffer

NAME
    get_f_stats - get function statistics

SYNOPSIS

```
returnL = get_f_stats("counter_id");
unsigned long returnL;        /* value of requested counter
                              */
char *counter_id;             /* string defining stats
                                 counter:
                                 "BW" = Bytes Written Count
                                 "BR" = Bytes Read Count
                                 "BC" = Bytes Compared Count
                                 "CE" = Compare Error Count
                              */
```

DESCRIPTION
    This function returns the requested function statistics information. Function stats are defined as the BW, BR, BC or CE count for the last I/O Driver operation or microprogramming data transfer function. This function is unaffected by **statsen()**.

DEFAULT VALUES: N.A.

RETURNS:

ØxFFFFFFFFL	error (-1)
requested counter info.	successful

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:
    IMPLICIT ERROR MESSAGES

    IMP. ER> get_f_stats(DE)   Date/Time Stamp
    Invalid Counter Reference

NAME

get_f_status - get function status

SYNOPSIS

```
return = get_f_status("status_id");
int return;                      /* error code/status */
char *status_id;                 /* string defining status
                                    byte:
                                    "IO" = I/O Driver Error
                                           Status
                                    "IE" = Initiator Error
                                           Status
                                    "TE" = Target Error Status
                                    "TM" = Last Target Message
                                 */
```

DESCRIPTION

This function returns the requested function status information. Function status is defined as the IO, IE, TE or TM bytes from the last I/O Driver operation. Function status is not defined for Microprogramming.

DEFAULT VALUES: N.A.

RETURNS:

ØxFFXX error (less than Ø)
Øx00BB successful, where BB = requested byte

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

IMPLICIT ERROR MESSAGES

IMP. ER> get_f_status(TF) Date/Time Stamp
Invalid Status Reference

NAME

get_g_stats - get global statistics

SYNOPSIS

```
returnL = get_g_stats("counter_id");
unsigned long returnL;         /* value of requested counter
                               */
char *counter_id;              /* string defining stats
                                  counter:
                                  "OP" = Operation Count
                                  "IE" = Initiator Error
                                         Count
                                  "CK" = Target Error Count
                                  "BW" = Bytes Written Count
                                  "BR" = Bytes Read Count
                                  "BC" = Bytes Compared Count
                                  "CE" = Compare Error Count
                               */
```

DESCRIPTION

This function returns the requested global statistics information.

DEFAULT VALUES: N.A.

RETURNS:

0xFFFFFFFFL	error (-1)
requested counter info.	successful

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

IMPLICIT ERROR MESSAGES

IMP. ER> get_g_stats(BD) Date/Time Stamp
Invalid Counter Reference

NAME

get_infoin - get current SCSI inbound information byte

SYNOPSIS

```
return = get_infoin();
int  return;
```

DESCRIPTION

Returns the current SCSI inbound information byte. If BUSY, REQ and I/O are not asserted, an error code is returned. The **get_infoin()** function should follow a **get_phase()** function.

DEFAULT VALUE: N.A.

RETURNS:

0xFF00	busy and req and I/O not asserted
0x00bb	bb = information byte

EXECUTION TYPE: Microprogramming

STATISTIC/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NOTE: The user is responsible for any error reporting required by this function. It is intended as an advanced programming tool and as such does not generate any implicit or explicit error messages.

NAME

get_phase - get current SCSI bus phase

SYNOPSIS

```
return = get_phase(req_wait);
unsigned req_wait;            /* wait for request:
                                 0 = no wait
                                 1 = wait */
int  return;                  /* error code/bus phase */
```

DESCRIPTION

Returns the current SCSI bus phase as defined above. If BUSY and REQ ('req_wait' = 0) are not asserted, an error is asserted. **ioto()** will apply to the 'req_wait' condition.

DEFAULT VALUE: N.A.

RETURNS:

```
0xFF00  busy not asserted
0xFE00  request not asserted ('req_wait' = 0);
0x000b  good completion:  bit 2 =  C/D
                          bit 1 =  I/O
                          bit 0 =  MSG
```

EXECUTION TYPE: Microprogramming

STATISTIC/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NOTE: The user is responsible for any error reporting required by this function. It is intended as an advanced programming tool and as such does not generate any implicit or explicit error messages.

NAME

get_user_int - return **user_input()** integer input

SYNOPSIS

```
return = get_user_int();
unsigned return;                /* integer from user_input() */
```

DESCRIPTION

This function returns the last integer entered by the **user_input()** function. The value returned from this function is not defined if **user_input()** was not called (with an integer argument).

DEFAULT VALUES: N.A.

RETURNS:

last **user_input()** integer

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

get_user_long - returns long input from **user_input()**

SYNOPSIS

```
returnL = get_user_long();
unsigned long returnL;        /* long from user_input() */
```

DESCRIPTION

This function returns the last long (32-bit) integer entered by the **user_input()** function. The value returned from this function is not defined if **user_input()** was not called (with a long argument).

DEFAULT VALUES: N.A.

RETURNS:

last 'long' obtained via **user_input()**

ERROR MESSAGES: NONE

NAME

group - print group line in fixed window, generate TOC entry, increment 'group_ref_counter' and Date and Time Stamp line

SYNOPSIS

```
group("Group Name");
```

DESCRIPTION

The **group()** function generates a group title and Table of Contents entry. The reference number associated with the group name is generated from the test reference number and the 'group_ref_counter.' The 'group_ref_counter' is set to Ø at SAT initialization and incremented each time a **group()** function is executed. 'group_ref_counter' is an internal SDS-1 variable.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME
    iea - set implicit error action

SYNOPSIS
```
iea("action");
char *action;                    /* error action to be taken */
```

DESCRIPTION
    Sets the action to be taken on an implicit error. An implicit error is an error that requires no explicit test to determine that an error has occurred (such as a data compare error).

The error action types are defined below:

```
CONT - no action (ignore error)
HALT - in nonbatch mode:  halt and enter Debugger
       in batch mode:  exit to next SAT
LOGC - log error and continue up to the set_er_limits()
       function limit;  otherwise,
       in nonbatch mode:  enter the Debugger
       in batch mode:  exit to DOS
LOGH - in nonbatch mode:  log error and enter Debugger
       in batch mode:  log error and exit to DOS
```

CONT and HALT types are not available in the Menu Interface. Also see Sections SAT.5 and DEBUG.1.3 .

DEFAULT VALUE: LOGC

RETURNS: N.A.

ERROR MESSAGES:
```
IMP. ER> iea("LAGC")
Undefined Error Action Parameter                Date/Time Stamp
```

NAME

iid - set initiator ID on defined Test Adapter

SYNOPSIS

```
return = iid(ha,newid);
int return;                     /* return value */
unsigned ha;                    /* always 0 */
unsigned newid;                 /* new initiator ID */
```

DESCRIPTION

Sets the SDS-1 SCSI initiator ID (test adapter always 0 since there is only one).

DEFAULT VALUE: 0

RETURNS:

1	error occurred
NULL(0)	successful

ERROR MESSAGES: NONE

NAME

inc_blk - increments starting block for **_blk()** commands

SYNOPSIS

```
return = inc_blk(increment);
unsigned long return;          /* new starting block value */
unsigned increment;            /* increment to starting block
                                  address */
```

DESCRIPTION

This function defines the starting block to be used in the **readr_blk()**, **writer_blk()**, **writerl0_blk()**, **readrl0_blk()** and **dmaset_vblk()** functions.

DEFAULT VALUES: NONE

RETURNS:

new starting block (unsigned long)

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

inc_len - increments transfer length for **_blk()** commands

SYNOPSIS

```
return = inc_len(increment);
unsigned return;              /* new transfer length */
unsigned increment;           /* increment to transfer
                                 length */
```

DESCRIPTION

This function defines the transfer length to be used in the **readr_blk()**, **writer_blk()**, **writer10_blk()** and **readr10_blk()** functions.

DEFAULT VALUES: NONE

RETURNS:

new transfer length (unsigned)

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME
inquiry - inquiry command

SYNOPSIS
```
return = inquiry(len);
unsigned return;                /* return code */
unsigned len;                   /* allocation length */
```

DESCRIPTION
This function will form and execute the command descriptor block for the inquiry command.

COMMAND DESCRIPTOR BLOCK FOR INQUIRY COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	12							
1	lun(lun);			ØØ				
2	ØØ							
3	ØØ							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

ioto - set primary time-out count

SYNOPSIS

```
ioto(value);
int value;                       /* I/O Driver time-out
                                    (in seconds) */
```

DESCRIPTION

Sets the primary I/O Driver and Microprogramming time-out count.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

io6 - 6-byte SCSI command

SYNOPSIS

```
return = io6(b0,b1,b2,b3,b4,b5);
unsigned return;                  /* return code */
int b0,b1,b2,b3,b4,b5;            /* command bytes */
```

DESCRIPTION

This function will form and execute the command descriptor block for any six-byte SCSI command as defined in the arguments. This function will allow execution of commands that may not be possible to perform due to constraints of the other functions.

COMMAND DESCRIPTOR BLOCK FOR SIX-BYTE COMMANDS

bit / byte	7	6	5	4	3	2	1	0
0	b0							
1	b1							
2	b2							
3	b3							
4	b4							
5	b5							

For a complete description of the commands refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

iol0 - 10-byte SCSI command

SYNOPSIS

```
return = iol0(b0,b1,b2,b3,b4,b5,b6,b7,b8,b9);
unsigned return;                /* return code */
int b0,b1,b2,b3,b4,b5,b6,b7,b8,b9;
                                /* command bytes */
```

DESCRIPTION

This function will form and execute the command descriptor block for any 10-byte SCSI command as defined in the arguments. This function will allow execution of commands that may not be possible to perform due to constraints of the other functions.

COMMAND DESCRIPTOR BLOCK FOR 10-BYTE COMMANDS

bit / byte	7	6	5	4	3	2	1	0
0	b0							
1	b1							
2	b2							
3	b3							
4	b4							
5	b5							
6	b6							
7	b7							
8	b8							
9	b9							

For a complete description of the commands refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

io12 - 12-byte SCSI command

SYNOPSIS

```
return = io12(b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11);
unsigned return;              /* return code */
int b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11;
                              /* command bytes */
```

DESCRIPTION

This function will form and execute the command descriptor block for any 12-byte SCSI command as defined in the arguments. This function will allow execution of commands that may not be possible to perform due to constraints of the other functions.

COMMAND DESCRIPTOR BLOCK FOR 12-BYTE COMMANDS

bit / byte	7	6	5	4	3	2	1	0
0	b0							
1	b1							
2	b2							
3	b3							
4	b4							
5	b5							
6	b6							
7	b7							
8	b8							
9	b9							
10	b10							
11	b11							

For a complete description of the commands refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful
1	error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME

ldunlds - load/unload command

SYNOPSIS

```
return = ldunlds(immed,reten,load);
unsigned return;              /* return code */
int immed;                    /* immediate bit */
int reten;                    /* retension bit */
int load;                     /* load bit */
```

DESCRIPTION

This function will form and execute the command descriptor block for the load/unload command.

COMMAND DESCRIPTOR BLOCK FOR LOAD/UNLOAD COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	1B							
1	lun(lun);			0				immed
2	00							
3	00							
4	00						reten	load
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

line_mode - select single-ended or differential SCSI mode

SYNOPSIS

```
return = line_mode("mode");
unsigned return;                    /* function return */
char *mode;                         /* S --> single-ended
                                       D --> differential */
```

DESCRIPTION

This function allows the user to select either set of SCSI line drivers and receivers and their associated port. Only one port (single-ended or differential) is enabled at any one time. The port which is disabled will be 'invisible' on its bus, i.e., it will not drive any lines high or low.

DEFAULT VALUE: Single-ended

RETURNS: Ø (always)

ERROR MESSAGES: NONE

NAME

loadbuf - load the current fill buffer with the contents of the specified disk file

SYNOPSIS

```
return = loadbuf("file",st_add,length);
int return;                      /* return code */
char *file;                      /* load buffer file name */
int unsigned st_add;             /* buffer starting address */
int unsigned length;             /* length of buffer to load */
```

DESCRIPTION

Loads buffer from disk file into the current fill buffer.

Typical format of load file:

```
0000:  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
0010:  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff fe
```

or

```
:  df 3d 54 ef c4 36 3c a0 31 d8 33 55 24 51 ca 3c
:  72 28 26 28 cb a0 f2 df c7 00 34 fa ac e1 ff ec
```

To create the load file, the requirements below must be followed:

a) A colon ":" must appear on every line before the 16 character bytes are listed.
b) Every character byte must be represented by two hex digits.
c) At least one space must separate the character bytes and only space(s) may appear between the character bytes.
d) Only 16 character bytes per line is allowed. The last line may contain less than 16 if the length is exact or no other character follows the last character byte (characters such as CR, LF, etc. will cause an error).

When the buffer is loaded from the disk, the data is in ASCII format. The **loadbuf()** function allows the user to create or modify and then load his own unique data pattern from disk.

Data will be loaded for the specified length or until the end of buffer or end of disk file whichever comes first.

DEFAULT VALUE: N.A.

RETURNS:

# of bytes loaded	successful
NULL(0)	error occurred

ERROR MESSAGES:
IMPLICIT ERROR MESSAGE

```
IMP. ER> loadbuf("file0",0000,0100)
Open Failed on Input File                            Date/Time Stamp

IMP. ER> loadbuf("file1",0000,0100)
Starting Address Greater Than Buffer Length          Date/Time Stamp

IMP. ER> loadbuf("file2",0200,0500)
Fill Buffer Not Set                                  Date/Time Stamp

IMP. ER> loadbuf("file3",0000,1000)
Read from File Failed                                Date/Time Stamp

IMP. ER> loadbuf("file4",0100,0200)
Invalid File Format                                  Date/Time Stamp

IMP. ER> loadbuf("file5",1000,0000)
Illegal Hex Character or Invalid File Format         Date/Time Stamp

IMP. ER> loadbuf("file6",0000,0000)
Not Enough Bytes per Line or Bytes to Load           Date/Time Stamp
```

NAME

logc - print a log line to the console (log device)

SYNOPSIS

```
logc("log string");
```

DESCRIPTION

The **logc()** function provides a means of logging information to the system console or log device.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME
    logp - print a log line to the printer and console

SYNOPSIS
```
logp("log string");
```

DESCRIPTION
    The **logp()** function provides a means of logging information to the system printer.

    In addition to going to the printer (or output file) the string will be displayed on the system console or log device as are all printer output lines.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

lun - set target LUN for the I/O Driver identify message and SCSI commands

SYNOPSIS

```
return = lun(lun);
int return;                    /* return code */
unsigned char lun;             /* target LUN (Ø -> 7) */
```

DESCRIPTION

This function sets the target LUN for SCSI commands issued through the I/O Driver. This LUN is to be inserted in byte one of the CDB (upper three bits) by the I/O Driver. The LUN sets the target LUN that the HOST is expected to communicate with.

DEFAULT VALUE: Ø

RETURNS:

NULL(Ø) successful, new test adapter number
1 error, test adapter does not exist

ERROR MESSAGES:

IMP. ER> lun(lun)
Illegal LUN Date/Time Stamp

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME

modesen - mode sense command

SYNOPSIS

```
return = modesen(alloc_len);
unsigned return;                  /* return code */
int alloc_len;                    /* allocation length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the mode sense command.

COMMAND DESCRIPTOR BLOCK FOR MODE SENSE COMMAND

bit byte	7	6	5	4	3	2	1	Ø
Ø	1A							
1	lun(lun);			ØØ				
2	ØØ							
3	ØØ							
4	alloc_len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

mode_sel - mode select command

SYNOPSIS

```
return = mode_sel(list_len);
unsigned return;                /* return code */
int list_len;                   /* parameter list length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the mode select command.

COMMAND DESCRIPTOR BLOCK FOR MODE SELECT COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	15							
1	lun(lun);			00				
2	00							
3	00							
4	list_len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

EXAMPLE:

Use the **fillk()** function to create the mode select parameter list in the write buffer:

```
fillk("00,00,00,08,00,00,00,00",0x0000,0x0008);
fillk("00,00,01,00,01,01,32,02",0x0008,0x0008);
fillk("01,33,01,33,00,01,00,00",0x0010,0x0008);
mode_sel(0x16);
```

NAME

modsels - mode select command

SYNOPSIS

```
return = modsels(list_len);
unsigned return;             /* return code */
unsigned list_len;           /* parameter list length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the mode select command. Refer to the **mode_sel()** function description on how to setup the parameter list.

COMMAND DESCRIPTOR BLOCK FOR MODE SELECT COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	15							
1	lun(lun);			00				
2	00							
3	00							
4	list_len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful completion
1 error occurred

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

modsens - mode sense command

SYNOPSIS

```
return = modsens(len);
unsigned return;                /* return code */
unsigned len;                   /* allocation length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the mode sense command.

COMMAND DESCRIPTOR BLOCK FOR MODE SENSE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	1A							
1	lun(lun);			00				
2	00							
3	00							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

msgin - single byte message in

SYNOPSIS

```
error = msgin(mi);
int error;                    /* return value */
BYTE mi;                      /* expected message in */
```

DESCRIPTION

Receives a message from the TARGET (via transmit/receive state machine) and verifies it against the expected message passed in the 'mi' argument. If the current information phase is not MESSAGE IN, an implicit error message is generated. If the actual and expected messages do not match, an implied error message is generated.

DEFAULT VALUE: N.A.

RETURNS:

0x0000	message is successful and matched
0x0009	SCSI bus reset detected
0x0005	I/O time-out
0x000D	invalid bus free detected
0x000F	parity error
0x0011	nonsupported message

EXECUTION TYPE: Microprogramming

STATISTIC/STATUS UPDATE:

Initiator Status Byte:

0x00	message is successful and matched
0x09	SCSI bus reset detected
0x05	I/O time-out
0x0D	invalid bus free detected
0x0F	parity error
0x11	nonsupported message

ERROR MESSAGES:

```
IMP. ER> msgin(mi)
No Message In Phase;                              Date/Time Stamp

IMP. ER> msgin(mi)
SCSI Bus Parity Error                             Date/Time Stamp

IMP. ER> msgin(0x00)
Actual Message 02 , Expected Message 00           Date/Time Stamp
```

NOTE: Use multiple single byte message functions in to create multiple byte message in.

NAME

msgout - single byte message out

SYNOPSIS

```
return = msgout(mo);
int return;                     /* return value */
BYTE mo;                        /* message out */
```

DESCRIPTION

Transfers the specified message (via transmit/receive state machine) to the TARGET. If the current information phase is not MESSAGE OUT, an implied error message is generated. Attention will always be deasserted after **msgout()**. Multiple message outs should be sent using **msgout_atnf()**.

DEFAULT VALUE: N.A.

RETURNS:

0x0000	message out successful
0x0009	SCSI bus reset detected
0x0005	I/O time-out
0x000D	invalid bus free detected
0x000C	invalid SCSI phase change

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

0x00	message out successful
0x09	SCSI bus reset detected
0x05	I/O time-out
0x0D	invalid bus free detected
0x0C	invalid SCSI phase change

ERROR MESSAGES:

IMP. ER> msgout(mo)
No Message Out Phase   Date/Time Stamp

NOTE: Use multiple single byte message out functions to create multiple byte message out.

NAME

msgout_atnf - single byte message out, force ATTN true

SYNOPSIS

```
return = msgout_atnf(mo);
int return;                         /* return value */
BYTE mo;                            /*  message out */
```

DESCRIPTION

Transfers the specified message (via transmit/receive state machine) to the TARGET. At the completion of the MESSAGE OUT transfer, attention will be asserted on the bus. If the current information phase is not MESSAGE OUT, an implicit error message is generated.

DEFAULT VALUE: N.A.

RETURNS:

0x0000 message out successful
0x0009 SCSI bus reset detected
0x0005 I/O time-out
0x000D invalid bus free detected
0x000C invalid SCSI phase change

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

0x00 message out successful
0x09 SCSI bus reset detected
0x05 I/O time-out
0x0D invalid bus free detected
0x0C invalid SCSI phase change

ERROR MESSAGES:

IMP. ER> msgout(mo)
No Message Out Phase   Date/Time Stamp

NAME

opcnt - check the number of operations completed

SYNOPSIS

```
return = opcnt(minL,maxL);
int return;                    /* return code */
unsigned long minL;            /* minimum value */
unsigned long maxL;            /* maximum value */
```

DESCRIPTION

Compares the number of operations completed with the 'minL' and 'maxL' limits. If the number is out of the specified range, the explicit error action will be taken.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(0)  successful, number within range
      1  error, number out of range
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> opcnt(121f50,122200)
I/O Operations Out of Range, I/O Count = 2c4ba2
```

NAME

overbcb - overlay buffer with block count byte
(make first byte of each block a block count)

SYNOPSIS

```
return = overbcb(st_byt,blk_len,st_add,len);
int return;                 /* return code */
int unsigned st_byt;        /* starting block count byte
                               (00h - FFh range) */
int unsigned blk_len;       /* length of block */
int unsigned st_add;        /* buffer starting address */
int unsigned len;           /* number of bytes to fill */
```

DESCRIPTION

For the write buffer, **overbcb()** writes the byte specified by 'st_byt' at 'st_add', then increments the 'st_byt' and writes that value at 'st_add'+'blk_len'. This basic function allows the user to fill the write buffer with any data pattern and then set the initial word of each block to a block number (which has a range of 00h - FFh).

```
fillpr(34,0,0x400);
overbcb(0x10,0x100,0,0x400);

0000:  10 34 85 34  . . .  89 86
00f0:  74 43 42 67  . . .  98 34
0100:  11 34 34 83  . . .  99 34
01f0:  19 01 10 87  . . .  17 83
0200:  12 83 45 4C  . . .  EF FF
02f0:  89 91 65 34  . . .  34 76
0300:  13 87 66 34  . . .  34 76
03f0:  34 56 10 03  . . .  89 87
```

The block count byte will rollover at FFh to 00h. The fill will be for the length specified or to the end of the fill buffer, whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:

```
      1  successful completion
NULL(0)  error occurred
```

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> overbcb(00,0010,0500,0100)
Starting Address Greater Than Buffer Length    Date/Time Stamp

IMP. ER> overbcb(f0,0100,0000,1000)
Fill Buffer Not Set                            Date/Time Stamp
```

NAME

overbcdw - overlay buffer with block count double word (set first double word of each block to block count)

SYNOPSIS

```
return = overbcdw(st_dblwrdL,blk_len,st_add,len);
int return;                    /* return code */
unsigned long st_dblwrdL;      /* starting block count
                                  double word
                                  (ØL - ØxFFFFFFFFL range) */
int unsigned blk_len;          /* length of block */
int unsigned st_add;           /* buffer starting address */
unsigned len;                  /* number of bytes to fill */
```

DESCRIPTION

For the write buffer, **overbcdw()** writes the double word specified by 'st_dblwrdL' at 'st_add', then increments the 'st_dblwrdL' and writes that value at 'st_add'+'blk_len'. This basic function allows the user to fill the write buffer with any data pattern and then set the initial double word of each block to a block number (which has a range of ØL - ØxFFFFFFFFL).

```
fillpr(34,Ø,Øx4ØØ);
overbcdw(Øx1ØØØØØØØL,Øx1ØØ,Ø,Øx4ØØ);

ØØØØ:  1Ø ØØ ØØ ØØ A4 Ø2  . . .  89 86
ØØfØ:  74 43 42 67 33 F8  . . .  98 34
Ø1ØØ:  1Ø ØØ ØØ Ø1 8B EC  . . .  99 34
Ø1fØ:  19 Ø1 1Ø 87 71 DA  . . .  17 83
Ø2ØØ:  1Ø ØØ ØØ Ø2 67 3Ø  . . .  EF FF
Ø2fØ:  89 91 65 34 8B CC  . . .  34 76
Ø3ØØ:  1Ø ØØ ØØ Ø3 81 26  . . .  34 76
Ø3fØ:  34 56 1Ø Ø3 6Ø DØ  . . .  89 87
```

The block count double word will rollover at FFFFFFFFh to Øh. The fill will be for the length specified or to the end of the fill buffer, whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:

```
      1  successful completion
NULL(Ø)  error occurred
```

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> overbcdw(0000L,0010,0500,0100)
Starting Address Greater Than Buffer Length     Date/Time Stamp

IMP. ER> overbcdw(1030fff0L,0100,0000,1000)
Fill Buffer Not Set                              Date/Time Stamp
```

NAME

overbcw - overlay buffer with block count word
(make first word of each block a block count)

SYNOPSIS

```
return = overbcw(st_wrd,blk_len,st_add,len);
int return;                    /* return code */
int unsigned st_wrd;           /* starting block count word
                                  (0000h - FFFFh range) */
int unsigned blk_len;          /* length of block */
int unsigned st_add;           /* buffer starting address */
int unsigned len;              /* number of bytes to fill */
```

DESCRIPTION

For the write buffer, **overbcw()** writes the word specified by 'st_wrd' at 'st_add', then increments the 'st_wrd' and writes that value at 'st_add'+'blk_len'. This basic function allows the user to fill the write buffer with any data pattern and then set the initial word of each block to a block number (which has a range of 0000h - FFFFh).

```
fillpr(34,0,0xffff);
overbcw(0x3020,0x100,0,0xffff);

0000:  30 20 85 34  . . .  89 86
00f0:  74 43 42 67  . . .  98 34
0100:  30 21 34 83  . . .  99 34
01f0:  19 01 10 87  . . .  17 83
0200:  30 22 45 4C  . . .  EF FF
02f0:  89 91 65 34  . . .  34 76
0300:  30 23 66 34  . . .  34 76
03f0:  34 56 10 03  . . .  89 87
  :                 :
  :                 :
```

The block count word will rollover at FFFFh to 0000h. The fill will be for the length specified or to the end of the fill buffer, whichever comes first.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:

```
      1  successful completion
NULL(0)  error occurred
```

ERROR MESSAGES:
IMPLICIT ERROR MESSAGE

```
IMP. ER> overbcw(0000,0010,0500,0100)
Starting Address Greater Than Buffer Length     Date/Time Stamp

IMP. ER> overbcw(fff0,0100,0000,1000)
Fill Buffer Not Set                             Date/Time Stamp
```

NAME

page - cause a page eject in Test Results report

SYNOPSIS

```
page();
```

DESCRIPTION

The **page()** function causes a Test Results report Page Eject.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

paragph - print a paragraph line in the fixed window, generate TOC entry, increment 'paragraph_ref_counter' and Date and Time stamp line

SYNOPSIS

paragph("Paragraph Name");

DESCRIPTION

The **paragph()** function generates a paragraph title and Table of Contents entry. The reference number associated with the paragraph name is generated from the test reference number, 'group_ref_counter' and the 'paragraph_ref_counter'. The 'paragraph_ref_counter' is reset each time a **group()** function is encountered and incremented at each **paragph()** function. It also resets at SAT initialization. 'paragraph_ref_counter' and 'group_ref_counter' are internal SDS-1 variables.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

parity - enable/disable SCSI parity checking and generation

SYNOPSIS

```
return = parity(n);
int return;                    /* return code */
unsigned char n;               /* Ø = off, 1 = on */
```

DESCRIPTION

Enables or disables the SCSI bus parity function (which turns on parity checking and parity generation) for both I/O Driver and Microprogramming operations.

DEFAULT VALUE: Ø (parity off)

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

pass - print Pass line on scrolling screen and in report and Date and Time Stamp line

SYNOPSIS

```
pass();
```

DESCRIPTION

The **pass()** function produces a Report Entry PASS in the right-hand column of the output Test Results report along with a Date and Time Stamp. The internal error level variable is not changed by **pass()**.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

pause - stop SAT execution and wait for **return** key

SYNOPSIS

```
pause("message");
char *message;                /* message to be displayed in
                                 the report scrolling
                                 window (logc() type
                                 message) */
```

DESCRIPTION

Stops SAT execution and waits for the user to hit the **return** key.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

prevmedr - prevent/allow media removal command

SYNOPSIS

```
return = prevmedr(prvent);
unsigned return;                  /* return code */
int prvent;                       /* prevent bit */
```

DESCRIPTION

This function will form and execute the command descriptor block for the prevent/allow media removal command.

COMMAND DESCRIPTOR BLOCK FOR PREVENT/ALLOW MEDIA REMOVAL COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	1E							
1	lun(lun);			00				
2	00							
3	00							
4	00							prvent
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

prevmeds - prevent/allow media removal command

SYNOPSIS

```
return = prevmeds(prevent);
unsigned return;               /* return code */
int prevent;                   /* prevent bit */
```

DESCRIPTION

This function will form and execute the command descriptor block for the prevent/allow media removal command.

COMMAND DESCRIPTOR BLOCK FOR PREVENT/ALLOW MEDIA REMOVAL COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	1E							
1	lun(lun);			00				
2	00							
3	00							
4	00							prevent
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
put_byte - set buffer address to specified value

SYNOPSIS
```
return = put_byte("r_w_s",address,byte);
int return;                    /* return code/status */
char *r_w_s;                   /* buffer reference string:
                                  "R" = read
                                  "W" = write
                                  "S" = sense  */
unsigned address;              /* address to modify */
char byte;                     /* byte to put */
```

DESCRIPTION
This function writes the defined byte to the write, read or sense buffer.

DEFAULT VALUES: N.A.

RETURNS:
```
0xFF00  error (less than 0)
   0x0  successful
```

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:
IMPLICIT ERROR MESSAGES

```
IMP. ER> put_byte(S,0x380,08)                  Date/Time Stamp
Address Greater than Buffer Size

IMP. ER> put_byte(R,0x380,08)                  Date/Time Stamp
Buffer Not Open

IMP. ER> put_byte(T,0x380,08)                  Date/Time Stamp
Invalid Buffer
```

NAME

random_blk - defines a random starting block for **_blk()** commands

SYNOPSIS

```
return = random_blk(minL,maxL);
unsigned long return;          /* new starting block value */
unsigned long minL;            /* minimum value for starting
                                  block address */
unsigned long maxL;            /* maximum value for starting
                                  block address */
```

DESCRIPTION

This function defines the starting block to be used in the **readr_blk()**, **writer_blk()**, **writerl0_blk()**, **readrl0_blk()** and **dmaset_vblk()** functions. When generating random addresses within a loop, this function must be executed for each new random address.

DEFAULT VALUES: NONE

RETURNS:

new starting block (unsigned long)

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

random_len - defines a random transfer length for **_blk()** commands

SYNOPSIS

```
return = random_len(min,max);
unsigned return;            /* new transfer length */
unsigned min;               /* minimum value for transfer
                               length */
unsigned max;               /* maximum value for transfer
                               length */
```

DESCRIPTION

This function defines the transfer length to be used in the **readr_blk()**, **writer_blk()**, **writerl0_blk()** and **readrl0_blk()** functions. When generating random addresses within a loop, this function must be executed for each new random address.

DEFAULT VALUES: NONE

RETURNS:

new transfer length (unsigned)

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

rbufbyte - compare read buffer byte within limits

SYNOPSIS

```
return = rbufbyte(address,lo,hi);
int return;                     /* return code */
int unsigned address;           /* address of byte to compare
                                */
int unsigned lo;                /* low byte value in range */
int unsigned hi;                /* high byte value in range */
```

DESCRIPTION

Compares the byte at 'address' in the current read buffer with the range specified by 'lo' and 'hi.' If the byte is not within the range, an explicit error action will be invoked.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful, byte within range
      1  error, byte out of range
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> rbufbyte(ØØ5Ø,18,f9)
No Read Buffer Open                               Date/Time Stamp

EXP. ER> rbufbyte(21ØØ,Ø2,Øb)
Byte Out of Range, Byte = Øc                      Date/Time Stamp

EXP. ER> rbufbyte(Ø5ØØ,1Ø,2Ø)
Address Greater Than Buffer Size                  Date/Time Stamp
```

NAME

rbufword - compare read buffer word within limits

SYNOPSIS

```
return = rbufword(address,lo,hi);
int return;                  /* return code */
int unsigned address;        /* address of word to compare
                             */
int unsigned lo;             /* low word value in range */
int unsigned hi;             /* high word value in range */
```

DESCRIPTION

Compares the word at 'address' in the current read buffer with the range specified by 'lo' and 'hi.' If the word is not within the range, an explicit error action will be invoked.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful, word within range
1	error, word out of range

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

EXP. ER> rbufword(13fe,Ø1ØØ,Ø2ØØ)
No Read Buffer Open — Date/Time Stamp

EXP. ER> rbufword(ØØØØ,1ØØØ,15ØØ)
Word Out of Range, Word = 151Ø — Date/Time Stamp

EXP. ER> rbufword(fØØØ,dfØ2,e256)
Address Greater Than Buffer Size — Date/Time Stamp

NAME
rdblklts - read block limits command

SYNOPSIS
```
return = rdblklts();
unsigned return;            /* return code */
```

DESCRIPTION
This function will form and execute the command descriptor block for the read block limits command.

COMMAND DESCRIPTOR BLOCK FOR READ BLOCK LIMITS COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	05							
1	lun(lun);			00				
2	00							
3	00							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

rd_buffer - read buffer

SYNOPSIS

```
return = rd_buffer(length,bcv,vu2,vu3,vu4,vu5,vu6);
unsigned return;              /* return code */
unsigned length;              /* allocation length */
int bcv;                      /* buffer control valid */
int vu2;                      /* Vendor Unique Byte 2 */
int vu3;                      /* Vendor Unique Byte 3 */
int vu4;                      /* Vendor Unique Byte 4 */
int vu5;                      /* Vendor Unique Byte 5 */
int vu6;                      /* Vendor Unique Byte 6 */
```

DESCRIPTION

This function will form and execute the command descriptor block for the read buffer command.

COMMAND DESCRIPTOR BLOCK FOR READ BUFFER COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	3C							
1	lun(lun);			0				BCV
2	Vendor Unique Byte 2							
3	Vendor Unique Byte 3							
4	Vendor Unique Byte 4							
5	Vendor Unique Byte 5							
6	Vendor Unique Byte 6							
7	Allocation Length MSB							
8	Allocation Length LSB							
9	cntlbyte(byte);							

For a complete description of the command refer to the Common Command Set (CCS) version of the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful completion
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

rd_defect - read defect data

SYNOPSIS

```
return = rd_defect(length,p,g,format);
unsigned return;                    /* return code */
unsigned length;                    /* allocation length */
int p;                              /* primary bit */
int g;                              /* growing bit */
int format;                         /* defect list format */
```

DESCRIPTION

This function will form and execute the command descriptor block for the read defect data command.

COMMAND DESCRIPTOR BLOCK FOR READ DEFECT DATA COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	37							
1	lun(lun);			0				
2					P	G	Format	
3	0							
4	0							
5	0							
6	0							
7	Allocation Length MSB							
8	Allocation Length LSB							
9	cntlbyte(byte);							

For a complete description of the command refer to the Common Command Set (CCS) version of the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful completion
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
readcap - read capacity command

SYNOPSIS
```
return = readcap(reladr,addL,pmi);
unsigned return;              /* return code */
int reladr;                   /* relative address bit */
unsigned long addL;           /* logical block address */
int pmi;                      /* partial medium indicator
                                 bit */
```

DESCRIPTION
This function will form and execute the command descriptor block for the read capacity command.

COMMAND DESCRIPTOR BLOCK FOR READ CAPACITY COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	25							
1	lun(lun);			0				reladr
2	addL (MSB)							
3	addL							
4	addL							
5	addL (LSB)							
6	00							
7	00							
8	00							pmi
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

readr - read command

SYNOPSIS

```
return = readr(start,len);
unsigned return;                    /* return code */
unsigned start;                     /* logical block address */
int len;                            /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the read command with a two byte starting block address. This means that the 6 byte SCSI CDB has 5 bits which are truncated. To use the entire starting block address field, use the **readrl()** function.

COMMAND DESCRIPTOR BLOCK FOR READ COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	08							
1	lun(lun);			00				
2	start (MSB)							
3	start (LSB)							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

readrev - read reverse command

SYNOPSIS

```
return = readrev(len);
unsigned return;                /* return code */
unsigned len;                   /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the read reverse command.

COMMAND DESCRIPTOR BLOCK FOR READ REVERSE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	0F							
1	lun(lun);			0				fixed(n);
2	00							
3	len (MSB)							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

readrl - six byte read command with long starting address

SYNOPSIS

```
return = readrl(startL,len);
unsigned return;              /* return code */
unsigned long startL;         /* logical block address */
unsigned len;                 /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the read command with long starting address.

COMMAND DESCRIPTOR BLOCK FOR READRL COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	ØB							
1	lun(lun);			start(MSB)				
2	start							
3	start (LSB)							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
ØxFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

readrlØ - read command (1Ø-byte command)

SYNOPSIS

```
return = readrlØ(reladr,st_addL,len);
unsigned return;              /* return code */
int reladr;                   /* relative address bit */
unsigned long st_addL;        /* logical block address */
unsigned int len;             /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 1Ø-byte read command.

COMMAND DESCRIPTOR BLOCK FOR 1Ø-BYTE READ COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	28							
1	lun(lun);			Ø				reladr
2	st_addL (MSB)							
3	st_addL							
4	st_addL							
5	st_addL (LSB)							
6	ØØ							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

readr10_blk - 10-byte read command using predefined starting block and length fields

SYNOPSIS

```
return = readr10_blk();
unsigned return;              /* return code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 10-byte read command using the starting block defined by **set_blk()**, **inc_blk()** or **random_blk()** and the length field set up by **set_len()**, **inc_len()** or **random_len()** (note the relative address bit is always set to zero).

COMMAND DESCRIPTOR BLOCK FOR READ10_BLK COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	28							
1	lun(lun);			0				0
2	set_blk(address (MSB))							
3	set_blk(address)							
4	set_blk(address)							
5	set_blk(address (LSB))							
6	00							
7	set_len(xfer_len (MSB))							
8	set_len(xfer_len (LSB))							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

readr_blk - 6-byte read command using predefined starting block and length fields

SYNOPSIS

```
return = readr_blk();
unsigned return;              /* return code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the read command using the starting block defined by **set_blk()**, **inc_blk()** or **random_blk()** and the length field set up by **set_len()**, **inc_len()** or **random_len()**.

COMMAND DESCRIPTOR BLOCK FOR READR_BLK COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	Ø8							
1	lun(lun);			set_blk(addressL MSB)				
2	set_blk(addressL)							
3	set_blk(addressL LSB)							
4	set_len(xfer_len)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
ØxFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
    reads - reads command

SYNOPSIS
```
return = reads(len);
unsigned return;                /* return code */
unsigned len;                   /* transfer length */
```

DESCRIPTION
    This function will form and execute the command descriptor block for the read command with a two byte starting address. This means that the 6 byte SCSI CDB has 5 bits which are truncated. To use the entire starting block address field, use the **readsl()** function.

COMMAND DESCRIPTOR BLOCK FOR READ COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	08							
1	lun(lun);			0				fixed(n);
2	00							
3	len (MSB)							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
    NULL(0) successful
    1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
    Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
    Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

readsl - read sequential command with long transfer length field

SYNOPSIS

```
return = readsl(lenL);
unsigned return;            /* return code */
unsigned long lenL;         /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the read sequential command.

COMMAND DESCRIPTOR BLOCK FOR READSL COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	08							
1	lun(lun);			0				fixed(n);
2	len (MSB)							
3	len							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

reasgnb - reassign blocks command

SYNOPSIS

```
return = reasgnb();
unsigned return;               /* return code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the reassign blocks command.

COMMAND DESCRIPTOR BLOCK FOR REASSIGN BLOCKS COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	07							
1	lun(lun);			00				
2	00							
3	00							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

recbufds - recover buffer data command

SYNOPSIS

```
return = recbufds(len);
unsigned return;                /* return code */
unsigned len;                   /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the recover buffer data command.

COMMAND DESCRIPTOR BLOCK FOR RECOVER BUFFER DATA COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	14							
1	lun(lun);			0				fixed(n);
2	00							
3	len (MSB)							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

recvdiag - receive diagnostic results command

SYNOPSIS

```
return = recvdiag(len);
unsigned return;                /* return code */
unsigned int len;               /* allocation length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the receive diagnostic results command.

COMMAND DESCRIPTOR BLOCK FOR RECEIVE DIAGNOSTIC RESULTS COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	1C							
1	lun(lun);			00				
2	00							
3	len (MSB)							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

releaser - release command for random access device

SYNOPSIS

```
return = releaser(3rd,3rdid,ext,resid);
unsigned return;              /* return code */
unsigned int 3rd;             /* 3rd party bit */
unsigned int 3rdid;           /* 3rd party device ID */
unsigned int ext;             /* extend bit */
unsigned int resid;           /* reservation ID */
```

DESCRIPTION

This function will form and execute the command descriptor block for the release command.

COMMAND DESCRIPTOR BLOCK FOR RELEASE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	17							
1	lun(lun);			3rd	3rdid			ext
2	resid							
3	00							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

releases - release command for sequential access device

SYNOPSIS

```
return = releases(3rd,3rdid);
unsigned return;            /* return code */
unsigned int 3rd;           /* 3rd party bit */
unsigned int 3rdid;         /* 3rd party device ID */
```

DESCRIPTION

This function will form and execute the command descriptor block for the release command.

COMMAND DESCRIPTOR BLOCK FOR RELEASE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	17							
1	lun(lun);			3rd	3rdid			0
2	00							
3	00							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

resel - verify reselection by a disconnecting TARGET

SYNOPSIS

```
return = resel();
int return;                    /* return value */
```

DESCRIPTION

The reselection sequence begins with BUSY deasserted and SEL, IO, and correct initiator ID asserted. The **resel()** function will complete the reselection handshake and return when physical path has been established. This function does not handle the IDENTIFY message in.

DEFAULT VALUE: N.A.

RETURNS:

Ø×ØØØØ good reselection sequence
ØxØØØ5 function time-out
ØxØØØA error in reselection process

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

ØxØØ good completion

ERROR MESSAGES:

IMP. ER> resel()
Invalid Reselection Sequence Date/Time Stamp

IMP. ER> resel()
Function Time-Out Date/Time Stamp

NAME

resel_wt - wait for reselection phase

SYNOPSIS

```
return = resel_wt();
unsigned return;                /* function return */
```

DESCRIPTION

This function is called when the SCSI bus is free. It returns when the test adapter detects a reselect phase on the bus (i.e., BSY false, SEL true, I/O- true). The function returns the value on the data bus at this time, which will be the sum of the target's ID and the ID of the host which is being selected (in bit significant form).

This function is intended to be used in a test which is simulating a multi-host environment. Typically this function will be used when more than one I/O thread is disconnected. Note that the test adapter will not respond to the reselect; if a reselection is desired, it is up to the user's program to setup the proper ID and call **resel()**.

DEFAULT VALUE: N.A.

RETURNS:

Øx00bb reselect detected; bb = data byte on the bus
Øx0500 I/O time-out
Øx0900 SCSI bus reset detected

EXECUTION TYPE: Microprogramming

ERROR MESSAGES:

IMP. ER> resel_wt()
Function Time-Out Date/Time Stamp

IMP. ER> resel_wt()
SCSI Reset Occurred Date/Time Stamp

NAME

reserves - reserve command for sequential access device

SYNOPSIS

```
return = reserves(3rd,3rdid);
unsigned return;              /* return code */
unsigned int 3rd;             /* 3rd party bit */
unsigned int 3rdid;           /* 3rd party device ID */
```

DESCRIPTION

This function will form and execute the command descriptor block for the reserve command.

COMMAND DESCRIPTOR BLOCK FOR RESERVE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	16							
1	lun(lun);			3rd	3rdid			0
2	00							
3	00							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

reservr - reserve command for random access device

SYNOPSIS

```
return = reservr(3rd,3rdid,ext,resid,list);
unsigned return;               /* return code */
unsigned int 3rd;              /* 3rd party bit */
unsigned int 3rdid;            /* 3rd party device ID */
unsigned int ext;              /* extend bit */
unsigned int resid;            /* reservation ID */
unsigned int list;             /* extent list length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the reserve command.

COMMAND DESCRIPTOR BLOCK FOR RESERVE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	16							
1	lun(lun);			3rd	3rdid			ext
2	resid							
3	list (MSB)							
4	list (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

reset - resets I/O Driver and SCSI bus

SYNOPSIS

```
return = reset();
int return;                      /* return value */
```

DESCRIPTION

The **reset()** function performs I/O Driver initialization functions which also resets the SCSI bus.

DEFAULT VALUE: N.A.

RETURNS:

0 successful
1 error

ERROR MESSAGES: NONE

NAME

rewind - rewind command

SYNOPSIS

```
return = rewind(immed);
unsigned return;              /* return code */
int immed;                    /* immediate bit */
```

DESCRIPTION

This function will form and execute the command descriptor block for the rewind command.

COMMAND DESCRIPTOR BLOCK FOR REWIND COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	01							
1	lun(lun);			0				immed
2	00							
3	00							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

rezero - rezero unit command

SYNOPSIS

```
return = rezero();
unsigned return;              /* return code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the rezero unit command.

COMMAND DESCRIPTOR BLOCK FOR REZERO UNIT COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	Ø1							
1	lun(lun);			ØØ				
2	ØØ							
3	ØØ							
4	ØØ							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

rptbuf - write buffer to report log

SYNOPSIS

```
rptbuf("buffer",start_add,length);
char *buffer;                    /* buffer type to write */
unsigned start_add;              /* starting address */
unsigned length;                 /* display length (in bytes)
                                 */
```

DESCRIPTION

Generates a buffer display for the requested buffer to the log device. Below are the different buffer types that can be specified by "buffer":

"R"	Read Buffer
"W"	Write Buffer
"RW"	Read/Write
"OBB"	On-Board Buffer
"L"	Log Buffer
"S"	Sense Buffer

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

rptsen - write sense buffer to report log

SYNOPSIS

```
rptsen();
```

DESCRIPTION

The **rptsen()** function will generate a sense buffer display for the current sense information in the sense buffer. The exact number of bytes transferred during the last sense command is displayed in the log.

```
0000: 00 02 00 de 04 de e6 9d a8 8b 34 00 01 00 32 00
0010: 01
```

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

rptstats - write statistics to report log

SYNOPSIS

```
rptstats(header_on_off);
char header_on_off;             /* Ø = off
                                   1 = on */
```

DESCRIPTION

Generates a statistics entry in the log report. The 'header_on_off' flag determines whether a header line will be printed above the statistics line. The example below is shown with the 'header_on_off' set to 1.

The following statistics are displayed:

I/O Operations
Target Checks
Bytes Written
Bytes Read
Bytes Compared
Compare Errors
Date/Time Stamp

```
IO OPs   TGT CKs BYTs WR BYTs RD BYTs CP  CP ERs
-------- ------- ------- ------- -------- --------
       6       Ø       Ø       Ø      1ØØ        Ø      8-2 9:ØØ
```

NOTE: All counts are in hex notation without the leading Øx.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME
rpttmr - write timers to report log

SYNOPSIS
rpttmr();

DESCRIPTION
The **rpttmr()** function will generate a timer display for the user timer and elapsed timer. The display format is as follows:

```
Elapsed Timer = 50.34;    User_Timer = 34.85;
```

The resolution is in seconds.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME

savebuf - save the contents of the current fill buffer to the specified disk file

SYNOPSIS

```
return = savebuf("file",st_add,length);
int return;                    /* return code */
char *file;                    /* disk file name where buffer
                                  is to be saved */
int unsigned st_add;           /* buffer starting address */
int unsigned length;           /* length of buffer to save */
```

DESCRIPTION

Saves the contents of the fill buffer.

Format of save file:

```
0000:  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
0010:  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff fe
```

When the buffer is saved to disk, the data is in ASCII format and can be edited. This file can then be loaded back by using the **loadbuf()** function.

NOTE:
1) This function will create the file if it does not exist.
2) When opening an existing disk file, the contents of the file is destroyed.

DEFAULT VALUE: N.A.

RETURNS:

# of bytes saved	successful completion
NULL(0)	error occurred

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> savebuf("save1",f000,0100)
Starting Address Greater Than Buffer Length     Date/Time Stamp

IMP. ER> savebuf("save2",0200,0500)
Fill Buffer Not Set                             Date/Time Stamp

IMP. ER> savebuf("save3",0100,0100)
Open Failed on Input File                       Date/Time Stamp

IMP. ER> savebuf("save4",1000,0100)
Write to File Failed                            Date/Time Stamp
```

NAME

sbb - sense byte compare

SYNOPSIS

```
return = sbb(address,min,max);
int return;                     /* return code */
int min;                        /* minimum value in range */
int max;                        /* maximum value in range */
int address;                    /* byte offset in buffer */
```

DESCRIPTION

Compares the byte at offset 'address' in the current sense buffer with the 'min' and 'max' argument values. If the byte is out of range, the explicit error action will be invoked. The sense buffer can either be nonextended or extended.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0)	successful, bytes compared
1	bytes miscompared
2	if not sense data

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> sbb(18,50,50)
Sense Byte Out of Range, Byte = 4f              Date/Time Stamp

EXP. ER> sbb(21,02,0f)
No Sense Buffer Open                            Date/Time Stamp
```

NAME

sbw - sense word compare

SYNOPSIS

```
return = sbw(address,min,max);
int return;                    /* return code */
int min;                       /* minimum value in range */
int max;                       /* maximum value in range */
int address;                   /* word offset in buffer */
```

DESCRIPTION

Compares the word at offset 'address' in the current sense buffer with the 'min' and 'max' argument values. If the word is out of range, the explicit error action will be invoked. The sense buffer can either be nonextended or extended.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful, words compared
      1  words miscompared
      2  if not sense data
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> sbw(Øc,1f5Ø,215Ø)
Sense Word Out of Range, Word = 1f2c            Date/Time Stamp

EXP. ER> sbw(Ø6,21ØØ,21ØØ)
No Sense Buffer Open                            Date/Time Stamp
```

NAME

searchde - search data equal command (10-byte command)

SYNOPSIS

```
return = searchde(inv,rcdfmt,spndat,reladr,st_addL,len);
unsigned return;              /* return code */
unsigned int inv;             /* invert bit */
unsigned int rcdfmt;          /* record format */
unsigned int spndat;          /* spanned data */
unsigned int reladr;          /* relative address bit */
unsigned long st_addL;        /* logical block address */
unsigned int len;             /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the search data equal command.

COMMAND DESCRIPTOR BLOCK FOR SEARCH DATA EQUAL COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	31							
1	lun(lun);			inv	rcdfmt		spndat	reladr
2	st_add (MSB)							
3	st_add							
4	st_add							
5	st_add (LSB)							
6	00							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

searchdh - search data high command (10-byte command)

SYNOPSIS

```
return = searchdh(inv,rcdfmt,spndat,reladr,st_addL,len);
unsigned return;             /* return code */
unsigned int inv;            /* invert bit */
unsigned int rcdfmt;         /* record format */
unsigned int spndat;         /* spanned data */
unsigned int reladr;         /* relative address bit */
unsigned long st_addL;       /* logical block address */
unsigned int len;            /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the search data high command.

COMMAND DESCRIPTOR BLOCK FOR SEARCH DATA HIGH COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	30							
1	lun(lun);			inv	rcdfmt		spndat	reladr
2	st_add (MSB)							
3	st_add							
4	st_add							
5	st_add (LSB)							
6	00							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

searchdl - search data low command (10-byte command)

SYNOPSIS

```
return = searchdl(inv,rcdfmt,spndat,reladr,st_addL,len);
unsigned return;                 /* return code */
unsigned int inv;                /* invert bit */
unsigned int rcdfmt;             /* record format */
unsigned int spndat;             /* spanned data */
unsigned int reladr;             /* relative address bit */
unsigned long st_addL;           /* logical block address */
unsigned int len;                /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the search data low command.

COMMAND DESCRIPTOR BLOCK FOR SEARCH DATA LOW COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	32							
1	lun(lun);			inv	rcdfmt		spndat	reladr
2	st_add (MSB)							
3	st_add							
4	st_add							
5	st_add (LSB)							
6	00							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
seek - seek command

SYNOPSIS
```
return = seek(add);
unsigned return;                /* return code */
unsigned add;                   /* logical block address */
```

DESCRIPTION
This function will form and execute the command descriptor block for the seek command with a two-byte starting block address. This means than the 6-byte SCSI CDB has 5 bits which are truncated. To use the entire starting block address field, use the **seekl()** function.

COMMAND DESCRIPTOR BLOCK FOR SEEK COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	ØB							
1	lun(lun);			ØØ				
2	add (MSB)							
3	add (LSB)							
4	ØØ							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

seekl - seek command with long address field

SYNOPSIS

```
return = seek(addL);
unsigned return;              /* return code */
unsigned long addL;           /* logical block address */
```

DESCRIPTION

This function will form and execute the command descriptor block for the seek command.

COMMAND DESCRIPTOR BLOCK FOR SEEKL COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	0B							
1	lun(lun);			add (MSB)				
2	add							
3	add (LSB)							
4	00							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
seek1Ø - seek command (1Ø-byte command)

SYNOPSIS
```
return = seek1Ø(addL);
unsigned return;              /* return code */
unsigned long addL;           /* logical block address */
```

DESCRIPTION
This function will form and execute the command descriptor block for the 1Ø-byte seek command.

COMMAND DESCRIPTOR BLOCK FOR 1Ø-BYTE SEEK COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	2B							
1	lun(lun);			ØØ				
2	addL (MSB)							
3	addL							
4	addL							
5	addL (LSB)							
6	ØØ							
7	ØØ							
8	ØØ							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

selmode - set select mode for I/O Driver operations

SYNOPSIS

```
return = selmode("mode");
int return;                    /* return value */
char *mode;                    /* DUMB = dumb select
                                  SMART = send ID message:
                                             11ØØØLLL
                                        (where LLL = LUN)
                               */
```

DESCRIPTION

The **selmode()** function will set the selection mode for I/O Driver operations. DUMB selection should be used with the NONE arbitration option. This combination will select (with a single ID bit) direct from bus free (SASI type operation).

Also see Section IODVR.3.4 .

DEFAULT VALUE: SMART

RETURNS:

Ø successful
1 error

ERROR MESSAGES:

IMPLICIT ERROR MESSAGES

IMP. ER> selmode("G")
Illegal Selection Mode Date/Time Stamp

NAME

sell - simple selection sequence for nonarbitrating environments (single-bit select - target ID only)

SYNOPSIS

```
return = sell(tid);
int return;                  /* return value */
BYTE tid;                    /* target ID number */
```

DESCRIPTION

Selects the requested SCSI target (tid) from the BUS FREE state. This function can only be used in systems where other devices are not arbitrating for the SCSI bus.

DEFAULT VALUE: N.A.

RETURNS:

Øx0000 selection successful
ØxØØ12 SCSI bus busy
Øx0006 selection time-out

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

ØxØØ good completion
Øx12 SCSI bus busy
ØxØ6 selection time-out

ERROR MESSAGES:

IMP. ER> sell(7)
Selection Time-Out   Date/Time Stamp

NAME

sel2 - selection with no message out
(double-bit select - target and initiator IDs)

SYNOPSIS

```
return = sel2(tid,iid);
int return;                    /* return value */
BYTE tid;                      /* target ID */
BYTE iid;                      /* initiator ID */
```

DESCRIPTION

Selects the requested SCSI target (tid) from the BUS FREE state. This function can only be used in systems where other devices are not arbitrating for the SCSI bus. The function will return a nonzero value if busy is not detected within a SCSI selection time-out.

DEFAULT VALUE: N.A.

RETURNS:

0x0000	selection successful
0x0012	SCSI bus busy
0x0006	selection time-out

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

0x00	good completion
0x12	SCSI bus busy
0x06	selection time-out

ERROR MESSAGES:

IMP. ER> sel2(7)
Selection Time-Out   Date/Time Stamp

NAME

sel3 - smart selection function for arbitrating environments

SYNOPSIS

```
return = sel3(tid);
int return;                      /* return value */
BYTE tid;                        /* target ID number */
```

DESCRIPTION

Selects the requested SCSI target (tid) with the INITIATOR ID bit set from the BUS BUSY state. This function can only be used after one of the arbitration functions. The function will return a nonzero value if a SCSI selection time-out (250ms) occurs during the selection process.

DEFAULT VALUE: N.A.

RETURNS:

0x0000 selection successful
0x0012 SCSI bus busy
0x0006 selection time-out

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

0x00 good completion
0x12 SCSI bus busy
0x06 selection time-out

ERROR MESSAGES:

IMP. ER> sel3(7)
Selection Time-Out Date/Time Stamp

NAME

sel4 - smart selection with message out

SYNOPSIS

```
return = sel4(tid,msgout);
int return;                      /* return value */
BYTE tid;                        /* target ID number */
BYTE msgout;                     /* bit 7  Always 1
                                    bit 6  Ø = no disconnect
                                               support
                                    bit 5
                                    bit 4
                                    bit 3
                                    bit 2  Operation LUN MSB
                                    bit 1  Operation LUN
                                    bit Ø  Operation LUN LSB */
```

DESCRIPTION

Selects the requested SCSI target (tid) with the Initiator ID bit set from the BUS BUSY state with attention asserted and pass 'msgout' to the TARGET. This function can only be used after one of the arbitration functions. The function will return a nonzero value if a SCSI selection time-out (25Øms) occurs during the selection process.

DEFAULT VALUE: N.A.

RETURNS:

Øx0000	selection successful
Øx0012	SCSI bus busy
Øx0009	SCSI bus reset detected
Øx0006	selection time-out
Øx0005	I/O time-out
Øx000D	invalid bus free detected
Øx0001	not message out

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:

Initiator Status Byte:

ØxØØ	good completion
ØxØ9	SCSI bus reset detected
ØxØ6	selection time-out
ØxØ5	I/O time-out
ØxØD	invalid bus free detected
ØxØ1	not message out

ERROR MESSAGES:

```
IMP. ER> sel4(7,8Ø)
Selection Time-Out                                  Date/Time Stamp
```

NAME

senddiag - send diagnostic command

SYNOPSIS

```
return = senddiag(selftst,devof,unitof,len);
unsigned return;            /* return code */
unsigned int selftst;       /* self test bit */
unsigned int devof;         /* device off-line bit */
unsigned int unitof;        /* unit off-line bit */
unsigned int len;           /* parameter list length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the send diagnostic command.

COMMAND DESCRIPTOR BLOCK FOR SEND DIAGNOSTIC COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	1D							
1	lun(lun);			0		selftst	devof	unitof
2	00							
3	len (MSB)							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
sense - request sense command

SYNOPSIS

```
return = sense(len);
unsigned return;              /* return code */
int len;                      /* allocation length */
```

DESCRIPTION
This function will form and execute the command descriptor block for the request sense command.

COMMAND DESCRIPTOR BLOCK FOR REQUEST SENSE COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	Ø3							
1	lun(lun);			ØØ				
2	ØØ							
3	ØØ							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

Also see Section IODVR.6 .

DEFAULT VALUE: N.A.

RETURNS:
NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

serclass - sense error class check

SYNOPSIS

```
return = serclass(class);
int return;                    /* return code */
int class;                     /* class value to compare */
```

DESCRIPTION

Compares the error class in the current sense buffer with the 'class' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain sense information or an error will be returned.

For sense data, an error class of six or less is valid.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful, values are equal
1	values are not equal
2	if not extended sense data
3	if no sense buffer open

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> serclass(6)
Error Class Does Not Match, Error Class = 3

EXP. ER> serclass(Ø)
Extended Sense                              Date/Time Stamp

EXP. ER> serclass(1)
No Sense Buffer Open                        Date/Time Stamp
```

NAME

serrcd - sense error code check

SYNOPSIS

```
return = serrcd(code);
int return;                    /* return code */
int code;                      /* code value to compare */
```

DESCRIPTION

Compares the error code in the current sense buffer with the 'code' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful, values are equal
      1  values are not equal
      2  if not extended sense data
      3  if no sense buffer open
```

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> serrcd(6)
Error Code Does Not Match, Error Code = 5

EXP. ER> serrcd(Ø)
Extended Sense                                Date/Time Stamp

EXP. ER> serrcd(f)
No Sense Buffer Open                          Date/Time Stamp
```

NAME

setbuf - fill buffer with string of specified ASCII data

SYNOPSIS

```
return = setbuf("string",st_add);
int return;                       /* return code */
char*string;                      /* ASCII string containing
                                     fill data */
int unsigned st_add;              /* buffer starting address */
```

DESCRIPTION

Fills buffer with ASCII data specified in "string" at 'st_add'.

```
setbuf("This is a test 1 2 3",0);

0000:  54 48 49 53 20 49 53 20 41 20 54 45 53 54 20 31
0010:  20 32 20 33
```

The first byte of the string will be stored at the starting address.

The 'st_add' argument should be in the range of the buffer length.

DEFAULT VALUE: N.A.

RETURNS:

```
      1  successful
NULL(0)  error
```

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> setbuf("Here I am",0100)
Starting Address Greater Than Buffer Length    Date/Time Stamp

IMP. ER> setbuf("123456",0000)
Fill Buffer Not Set                            Date/Time Stamp
```

NAME

setfill_buf - set buffer to be used for fill functions

SYNOPSIS

```
return = setfill_buf("r_w_s");
int return;                     /* completion status */
char *r_w_s;                    /* String defining buffer:
                                   "R" = Read
                                   "W" = write
                                   "S" = Sense */
```

DESCRIPTION

Specifies the buffer to be used by subsequent fill commands. **xfermode()** sets a default value to "W." This command should only be used if the user wishes to fill a buffer other than the write buffer.

DEFAULT VALUE: set to "W" by **xfermode()**

RETURNS:

0	error
nonzero	successful

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

IMPLICIT ERROR MESSAGES

IMP. ER> setfill_buf(R) Date/Time Stamp
Buffer Not Open

NAME

setlimts - set limits command (10-byte command)

SYNOPSIS

```
return = setlimts(rdinh,wrinh,st_addL,len);
unsigned return;              /* return code */
unsigned int rdinh;           /* read inhibit bit */
unsigned int wrinh;           /* write inhibit bit */
unsigned long st_addL;        /* logical block address */
unsigned int len;             /* number of blocks */
```

DESCRIPTION

This function will form and execute the command descriptor block for the set limits command.

COMMAND DESCRIPTOR BLOCK FOR SET LIMITS COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	33							
1	lun(lun);			0			rdinh	wrinh
2	st_addL (MSB)							
3	st_addL							
4	st_addL							
5	st_addL (LSB)							
6	00							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

set_blk - sets starting block for **_blk()** commands

SYNOPSIS

```
returnL = set_blk(valueL);
unsigned long returnL;          /* return valueL */
unsigned long valueL;           /* starting block address */
```

DESCRIPTION

This function defines the starting block to be used in the **readr_blk()**, **writer_blk()**, **writer10_blk()**, **readr10_blk()** and **dmaset_vblk()** functions.

DEFAULT VALUES: NONE

RETURNS:

defined starting block (unsigned long)

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

set_er_limits - set error limit

SYNOPSIS

```
set_er_limits(limit);
unsigned limit;                /* maximum limit count */
```

DESCRIPTION

The **set_er_limits()** function will set the error limit that will cause the SAT to abort and return to DOS. This overrides the **eea()** and **iea()** value of LOGC (Log and Continue). The default error limit is 100d.

DEFAULT VALUE: 100

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

set_len - sets transfer length for **_blk()** commands

SYNOPSIS

```
return = set_len(value);
unsigned return;                /* return value */
unsigned value;                 /* transfer length */
```

DESCRIPTION

This function defines the transfer length to be used in the **readr_blk()**, **writer_blk()**, **writerl0_blk()** and **readrl0_blk()** functions.

DEFAULT VALUES: NONE

RETURNS

defined transfer length (unsigned)

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES: NONE

NAME

sladdr - check range of logical block address

SYNOPSIS

```
return = sladdr(minL,maxL);
int return;                    /* return code */
unsigned long minL;            /* minimum value */
unsigned long maxL;            /* maximum value */
```

DESCRIPTION

Compares the logical block address in the SCSI sense buffer with the 'minL' and 'maxL' limits. If the address is out of the specified range, the explicit error action will be taken. The sense buffer must contain standard sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0)	successful, address within range
1	address out of range
2	extended sense data
3	ADVALID false
4	no sense buffer open

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

```
EXP. ER> sladdr(121f50,122200)
Logical Block Address Not Valid (valid bit not set)

EXP. ER> sladdr(10000,ff000)
Logical Block Address Out of Range, Address = 1104cdl

EXP. ER> sladdr(30745,33200)
NonExtended Sense                              Date/Time Stamp

EXP. ER> sladdr(2100,5000)
No Sense Buffer Open                           Date/Time Stamp
```

NAME

space - space command

SYNOPSIS

```
return = space(code,count);
unsigned return;                /* return code */
int code;                       /* code */
unsigned count;                 /* number of filemarks */
```

DESCRIPTION

This function will form and execute the command descriptor block for the space command.

COMMAND DESCRIPTOR BLOCK FOR SPACE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	11							
1	lun(lun);			0			code	
2	00							
3	count (MSB)							
4	count (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

state_data - obtain the data associated with a particular state log entry

SYNOPSIS

```
returnL = state_data("state",count);
unsigned long returnL;        /* data associated occurrence
                                 'count' of event "state" */
char *state;                  /* state description */
int count;                    /* # of "state" occurrences */
```

DESCRIPTION

This function looks backward in the bus state log from the current time for 'count' occurrences of "state". If the specified state is found, it returns the data associated with the state. This data may be a byte count (in the case of DATA IN or DATA OUT), or it may be a single byte of data (in all other cases). Errors are reported via **get_f_status("IO");** the user cannot tell from the return value of the function whether or not an error occurred. The definitions of the "state" strings are defined below:

```
"ARB_START"      --> arbitration
"SEL_ASSERT"     --> assertion of SEL by host
"CMD"            --> command out
"DATAIN"         --> data in phase
"DATAOUT"        --> data out phase
"RESEL"          --> reselection
"MSG_OUT"        --> message out
"MSG_IN"         --> message in
"STATUS"         --> status
```

DEFAULT VALUE: N.A.

RETURNS:

either a byte count or a data byte

I/O status:

- 0x00 normal termination (success)
- 0x40 specified value of "state" not found
- 0x41 illegal string specified for "state"

ERROR MESSAGES:

```
IMP. ER> state_data(state,count);
State not found                                 Date/Time Stamp

IMP. ER>state_data(state,count);
Illegal state specifier                         Date/Time Stamp
```

NAME
    statin - single byte status input

SYNOPSIS

```
statin(si);
BYTE si;                        /* expected status in */
```

DESCRIPTION
    Receives an ending status byte from the TARGET (via transmit/receive state machine) and verifies it against the expected status passed in the 'si' argument. If the current information phase is not status in then an implied error message is generated. If the actual and expected status do not match, an implied error message is generated.

DEFAULT VALUE: N.A.

RETURNS:

0x0000	status is successful and matched
0x0009	SCSI bus reset detected
0x0005	I/O time-out
0x000D	invalid bus free detected
0x000C	invalid SCSI phase change
0x000F	SCSI parity error
0x0011	non-supported message

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE:
    Initiator Status Byte:

0x00	status is successful and matched
0x09	SCSI bus reset detected
0x05	I/O time-out
0x0D	invalid bus free detected
0x0C	invalid SCSI phase change
0x0F	SCSI parity error
0x11	non-supported message

ERROR MESSAGES:

```
IMP. ER> statin(si)
No Status In Phase                              Date/Time Stamp

IMP. ER> statin(0x00)
Actual Status 02 , Expected Status 00           Date/Time Stamp

IMP. ER> statin(si)
SCSI Bus Parity Error                           Date/Time Stamp
```

NAME

statsen - enable/disable statistics gathering

SYNOPSIS

```
statsen(bit);
int bit;                        /* 0 = no stats
                                   1 = gather stats from
                                     I/O Driver calls */
```

DESCRIPTION

Enables or disables statistics gathering from I/O Driver calls. These statistics pertain to data transfer such as: the number of bytes written, number of bytes read, number of compares, number of miscompares, number of commands executed and other implementation dependent values. These statistics appear on the left side of the I/O Driver Status Window. The **get_f_status()** function is unaffected by this function.

Also see Section IODVR.5 .

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

stats_reset - reset global statistics counters

SYNOPSIS

```
return = stats_reset("counter_id");
int return;                     /* completion code */
char *counter_id;               /* string defining stats counter
                                   to reset:
                                   "OP" = Operation Count
                                   "IE" = Initiator Error Count
                                   "CK" = Target Error Count
                                   "BW" = Bytes Written Count
                                   "BR" = Bytes Read Count
                                   "BC" = Bytes Compared Count
                                   "CE" = Compare Error Count
                                   "A" = All stats counters
                                */
```

DESCRIPTION

This function resets the requested global statistics counter shown in the I/O Driver Status Window.

DEFAULT VALUES: N.A.

RETURNS:

```
ØxFFFF  error (-1)
     Ø  reset is successful
```

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

IMPLICIT ERROR MESSAGES

IMP. ER> stats_reset(BD) Date/Time Stamp
Invalid Counter Reference

NAME

stats_window - select statistics window display

SYNOPSIS

```
return = stats_window("window_string");
int return;                          /* return code/status */
char *window_string;                 /* window reference string:
                                        "G" = global statistics
                                        "F" = function statistics
                                     */
```

DESCRIPTION

This function allows the user to select either the global or function statistics to be displayed in the statistics window but only if the statistics is enabled.

DEFAULT VALUES: N.A.

RETURNS:

0xFF00 error (<0)
0x0000 successful

EXECUTION TYPE: N.A.

STATISTICS/STATUS UPDATE: N.A.

ERROR MESSAGES:

IMPLICIT ERROR MESSAGES

```
IMP. ER> status_window("T")                     Date/Time Stamp
Invalid Status Code
```

NAME

stat_mask - set TARGET status mask

SYNOPSIS

```
stat_mask(byte);
unsigned char byte;          /* status mask value:
                                1 = allow comparison of bit
                                Ø = mask bit
                                    (force bit to Ø) */
```

DESCRIPTION

This function sets the mask that will be applied to the target status byte before it is compared with the **exp_status()** value. It is set to 1 (all bits checked) on initial entry and must be set if any bits are to be masked (forced to Ø).

Also see Section IODVR.4.1 .

DEFAULT VALUE: ØxFF (all bits in TARGET status tested)

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

strstop - start/stop unit command

SYNOPSIS

```
return = strstop(immed,start);
unsigned return;                /* return code */
int immed;                      /* immediate bit */
int start;                      /* start bit */
```

DESCRIPTION

This function will form and execute the command descriptor block for the start/stop unit command.

COMMAND DESCRIPTOR BLOCK FOR START/STOP UNIT COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	1B							
1	lun(lun);			Ø				immed
2	ØØ							
3	ØØ							
4	ØØ							start
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

subpar - print subparagraph line in the fixed window, generate TOC entry and Date and Time Stamp line

SYNOPSIS

```
subpar("Sub-Paragraph Name","ref_string");
```

DESCRIPTION

The **subpar()** function allows the user to go beyond the two levels of structure established by the **group()** and **paragph()** functions. The **subpar()** function does not increment the 'paragraph_ref_counter' but adds the "ref_string" to it to form a the sub-paragraph reference number. A TOC entry will be generated by the **subpar()** function. 'paragraph_ref_counter' is an internal SDS-1 variable.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

summary - print a summary line to the console, report summary log (console and report lines are Date and Time stamped)

SYNOPSIS

```
summary("summary_string");
```

DESCRIPTION

The **summary()** function produces a summary log Entry which will be included in Appendix B of the Test Results report and in the body of the report itself. The summary log entry will have a reference number associated with it. This reference number is generated by the **test()**, **group()**, **paragph()** and **subpar()** functions.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME
svalid - sense valid check

SYNOPSIS
```
return = svalid(n);
int return;                    /* return code */
int n;                         /* bit value to compare */
```

DESCRIPTION
Compares the valid bit in the current sense buffer with the 'n' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful, values are equal
1	values are not equal
2	if not extended sense data
3	if no sense buffer open

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

EXP. ER> svalid(1)
Valid Bit Reset   Date/Time Stamp

EXP. ER> svalid(Ø)
Valid Bit Set   Date/Time Stamp

EXP. ER> svalid(Ø)
Extended Sense   Date/Time Stamp

EXP. ER> svalid(1)
No Sense Buffer Open   Date/Time Stamp

NAME

svu - sense vendor unique check

SYNOPSIS

```
return = svu(value);
int return;                     /* return code */
unsigned int value;             /* value to compare */
```

DESCRIPTION

Compares the vendor unique value in the current sense buffer with the 'value' argument value. If the values do not match, the explicit error action will be taken. The sense buffer must contain sense information or an error will be returned.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø)	successful, values are equal
1	values are not equal
2	if not extended sense data
3	if no sense buffer open

ERROR MESSAGES:

EXPLICIT ERROR MESSAGES

EXP. ER> svu(7)
Vendor Unique Does Not Match, Vendor Unique = 2

EXP. ER> svu(Ø)
Extended Sense   Date/Time Stamp

EXP. ER> svu(5)
No Sense Buffer Open   Date/Time Stamp

NAME

test - print the test line in the fixed window and generate a table of contents (TOC) entry

SYNOPSIS

```
test("FILENAME Test Title");
```

DESCRIPTION

The **test()** function must be the first library function called within a Stand-Alone Test program. This function performs library initialization for the other functions. In addition, **test()** provides the Test Title for the Test Results report. The format of this title string is shown above. FILENAME is the file name of the TEST.EXE file which is executing. This word will appear in the foot of each page in the Test Results report. The entire title string will appear as the Test title in both the document body and Table of Contents.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

testur - test unit ready command

SYNOPSIS

```
return = testur();
unsigned return;                /* return code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the test unit ready command.

COMMAND DESCRIPTOR BLOCK FOR TEST UNIT READY COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	ØØ							
1	lun(lun);			ØØ				
2	ØØ							
3	ØØ							
4	ØØ							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

tid - set target ID

SYNOPSIS

```
return = tid(newid);
int return;                   /* return code */
unsigned char newid;          /* new target ID (Ø - 7) */
```

DESCRIPTION

Sets the SCSI target ID to be used by the current test adapter for subsequent commands. This identifies which target the host will attempt to select during the selection phase. This command is used in conjunction with the **iid()** function to setup a logical thread in a multi-target environment.

DEFAULT VALUE: Ø

RETURNS:

1	error, target ID is not in the range of Ø to 7
NULL(Ø)	successful, new target ID

ERROR MESSAGES:

IMP. ER> tid(newid)
Illegal Target I.D. Date/Time Stamp

NAME
tksel - track select command

SYNOPSIS
```
return = tksel(tk_val);
unsigned return;               /* return code */
unsigned tk_val;               /* track value */
```

DESCRIPTION
This function will form and execute the command descriptor block for the track select command.

COMMAND DESCRIPTOR BLOCK FOR TRACK SELECT COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	0B							
1	lun(lun);			00				
2	00							
3	00							
4	tk_val							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

tmrlmt - user timer limit check

SYNOPSIS

```
return = tmrlmt(lo,hi);
int return;                    /* return code */
int lo;                        /* low limit (in seconds) */
int hi;                        /* high limit (in seconds) */
```

DESCRIPTION

Checks to see if current timer value is within the 'lo' and 'hi' limits specified.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful, within range
1 out of range

ERROR MESSAGES:

EXPLICIT ERROR MESSAGE

EXP. ER> tmrlmt(2Ø,4Ø)
Timer (Current Value = 5Ø) out of limits Date/Time Stamp

NAME

tmrset - preset user timer

SYNOPSIS

```
tmrset(value);
unsigned value;                  /* time to preset in seconds
                                    i.e. 1 = preset to 1 second
                                 */
```

DESCRIPTION

This function will preset the user timer with the specified value. The 'value' is the number of seconds to be used as the starting count. This function does not start the timer, the **tmrstart()** function will start the timer.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME
tmrstart - start user timer with incrementing or decrementing values

SYNOPSIS
```
tmrstart("up_down");
char *up_down;                /* U = Up count (increment)
                                 D= Downcount (decrement)
                              */
```

DESCRIPTION
The **tmrstart()** function will start the timer counting up or down as defined by "up_down". The timer should be preset by the the **tmrset()** function before starting. The timer will count down to 0 and if counting up, will count up to 0xFFFF seconds.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME

tmrstop - stop user timer

SYNOPSIS

```
tmrstop();
```

DESCRIPTION

This function stops the user timer.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

NAME
    tmrvalue - return timer value

SYNOPSIS
```
int_time = tmrvalue();
int int_time;                    /* return current time */
```

DESCRIPTION
    Returns the current value of user timer.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME

ucinc - increment or decrement user counter count

SYNOPSIS

```
return = ucinc(cntr,value);
int return;                    /* return code */
int cntr;                      /* Ø = user counter Ø
                                  1 = user counter 1 */
int value;                     /* value */
```

DESCRIPTION

Increments or decrements user counter count.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful
      1  error
```

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> ucinc(4,1Ø)
Illegal User Counter Reference #                Date/Time Stamp
```

NAME

ucname - set user count name

SYNOPSIS

```
return = ucname(cntr,"name");
int return;                 /* return code */
int cntr;                   /* Ø = user counter Ø
                               1 = user counter 1 */
char *name;                 /* user counter name */
```

DESCRIPTION

The **ucname()** function will set user counter Ø or 1 to the specified name.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful
      1  error
```

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> ucname(2,"ctrØ")
Illegal User Counter Reference #                Date/Time Stamp
```

NAME

ucrst - reset user counter count

SYNOPSIS

```
return = ucrst(cntr);
int return;                    /* return code */
int cntr;                      /* Ø = user counter Ø
                                  1 = user counter 1 */
```

DESCRIPTION

This function will reset user counter count.

DEFAULT VALUE: N.A.

RETURNS:

```
NULL(Ø)  successful
      1  error
```

ERROR MESSAGES:

IMPLICIT ERROR MESSAGE

```
IMP. ER> ucrst(3)
Illegal User Counter Reference #               Date/Time Stamp
```

NAME
ureset - generate a SCSI reset pulse longer than 25 usec

SYNOPSIS
```
ureset();
```

DESCRIPTION
This function will raise the reset signal for more than 25 usec then deassert it and clear SCSI bus signals.

DEFAULT VALUE: N.A.

RETURNS: N.A

EXECUTION TYPE: Microprogramming

STATISTICS/STATUS UPDATE: NONE

ERROR MESSAGES: NONE

NAME

user_input - user action/response request

SYNOPSIS

```
user_input("string","type");
char *string;                  /* string to be displayed at
                                  the bottom of the screen
                                  (in debug window) */
char *type;                    /* "S" = string
                                  "N" = numeric (unsigned int)
                                  "L" = unsigned long */
```

DESCRIPTION

Stops SAT execution and waits for the user to enter a specific response such as a character string (up to 30 characters) or unsigned integer or long. This response can then be checked by the **chk_user_string()**, **chk_user_limits()**, **get_user_int()** or **get_user_long()** functions.

If "type" is numeric (int) or long, the numbers to be entered may be in decimal or hex. Use the "0x" notation for hex numbers, otherwise the number defaults to decimal.

DEFAULT VALUE: N.A.

RETURNS: N.A.

ERROR MESSAGES: NONE

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME

verify10 - verify command (10-byte command)

SYNOPSIS

```
return = verify10(bytck,reladr,st_addL,len);
unsigned return;              /* return code */
unsigned int bytck;           /* byte check bit */
unsigned int reladr;          /* relative address bit */
unsigned long st_addL;        /* logical block address */
unsigned int len;             /* verification length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 10-byte verify command.

COMMAND DESCRIPTOR BLOCK FOR 10-BYTE VERIFY COMMAND

bit byte	7	6	5	4	3	2	1	0
0	2F							
1	lun(lun);			0			bytck	reladr
2	st_addL (MSB)							
3	st_addL							
4	st_addL							
5	st_addL (LSB)							
6	00							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

verifys - verify command

SYNOPSIS

```
return = verifys(bytcmp,len);
unsigned return;                /* return code */
int bytcmp;                     /* byte compare bit */
unsigned len;                   /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the verify command.

COMMAND DESCRIPTOR BLOCK FOR VERIFY COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	13							
1	lun(lun);			0			bytcmp	fixed(n);
2	00							
3	len (MSB)							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

(THIS PAGE INTENTIONALLY LEFT BLANK)

NAME
writer - write command

SYNOPSIS
```
return = writer(start,len);
unsigned return;              /* return code */
unsigned start;               /* logical block address */
int len;                      /* transfer length */
```

DESCRIPTION
This function will form and execute the command descriptor block for the write command with a two byte starting block address. This means that the 6 byte SCSI CDB has 5 bits which are truncated. To use the entire starting block address field, use the **writerl()** function.

COMMAND DESCRIPTOR BLOCK FOR WRITE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	0A							
1	lun(lun);			00				
2	start (MSB)							
3	start (LSB)							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

writerl - six-byte write command with long starting address

SYNOPSIS

```
return = writerl(startL,len);
unsigned return;            /* return code */
unsigned long startL;       /* logical block address */
unsigned len;               /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the write command.

COMMAND DESCRIPTOR BLOCK FOR WRITERL COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	ØA							
1	lun(lun);			start(MSB)				
2	start							
3	start (LSB)							
4	len							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
ØxFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

writer10 - write command (10-byte command)

SYNOPSIS

```
return = writer10(reladr,st_addL,len);
unsigned return;                /* return code */
int reladr;                     /* relative address bit */
unsigned long st_addL;          /* logical block address */
unsigned int len;               /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 10-byte write command.

COMMAND DESCRIPTOR BLOCK FOR 10-BYTE WRITE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	2A							
1	lun(lun);			0				reladr
2	st_addL (MSB)							
3	st_addL							
4	st_addL							
5	st_addL (LSB)							
6	00							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

writer10_blk - 10-byte write command using predefined starting block and length fields

SYNOPSIS

```
return = writer10_blk();
unsigned return;              /* return code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 10-byte write command using the starting block defined by **set_blk()**, **inc_blk()** or **random_blk()** and the length field set up by **set_len()**, **inc_len()** or **random_len()** (note the relative address bit is always set to zero).

COMMAND DESCRIPTOR BLOCK FOR WRITER10_BLK COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	2A							
1	lun(lun);			0				0
2	set_blk(address (MSB))							
3	set_blk(address)							
4	set_blk(address)							
5	set_blk(address (LSB))							
6	00							
7	set_len(xfer_len (MSB))							
8	set_len(xfer_len (LSB))							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

writer_blk - six-byte write command using predefined starting block and length fields

SYNOPSIS

```
return = writer_blk();
unsigned return;              /* return code */
```

DESCRIPTION

This function will form and execute the command descriptor block for the write command using the starting block defined by **set_blk()**, **inc_blk()** or **random_blk()** and the length field set up by **set_len()**, **inc_len()** or **random_len()**.

COMMAND DESCRIPTOR BLOCK FOR WRITER_BLK COMMAND

bit / byte	7	6	5	4	3	2	1	Ø
Ø	ØA							
1	lun(lun);			set_blk(addressL MSB)				
2	set_blk(addressL)							
3	set_blk(addressL LSB)							
4	set_len(xfer_len)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(Ø) successful
ØxFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME
writes - write command

SYNOPSIS
```
return = writes(len);
unsigned return;                /* return code */
unsigned len;                   /* transfer length */
```

DESCRIPTION
This function will form and execute the command descriptor block for the write command with a two byte starting block address. This means that the 6 byte SCSI CDB has 5 bits which are truncated. To use the entire starting block address field, use the **writerl()** function.

COMMAND DESCRIPTOR BLOCK FOR WRITE COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	ØA							
1	lun(lun);			Ø				fixed(n);
2	ØØ							
3	len (MSB)							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:
NULL(Ø) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

writesl - write sequential command with long transfer length field

SYNOPSIS

```
return = writesl(lenL);
unsigned return;              /* return code */
unsigned long lenL;           /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the write sequential command.

COMMAND DESCRIPTOR BLOCK FOR WRITESL COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	0A							
1	lun(lun);			0				fixed(n);
2	len (MSB)							
3	len							
4	len (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

wrtfilm - write filemarks command

SYNOPSIS

```
return = wrtfilm(count);
unsigned return;                /* return code */
unsigned count;                 /* number of file marks */
```

DESCRIPTION

This function will form and execute the command descriptor block for the write filemarks command.

COMMAND DESCRIPTOR BLOCK FOR WRITE FILEMARKS COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	10							
1	lun(lun);			00				
2	00							
3	count (MSB)							
4	count (LSB)							
5	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:

Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:

Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

wrtvfyl0 - write and verify command (10-byte command)

SYNOPSIS

```
return = wrtvfyl0(bytck,reladr,st_addL,len);
unsigned return;            /* return code */
unsigned int bytck;         /* byte check bit */
unsigned int reladr;        /* relative address bit */
unsigned long st_addL;      /* logical block address */
unsigned int len;           /* transfer length */
```

DESCRIPTION

This function will form and execute the command descriptor block for the 10-byte write and verify command.

COMMAND DESCRIPTOR BLOCK FOR 10-BYTE WRITE AND VERIFY COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	2E							
1	lun(lun);			0			bytck	reladr
2	st_addL (MSB)							
3	st_addL							
4	st_addL							
5	st_addL (LSB)							
6	00							
7	len (MSB)							
8	len (LSB)							
9	cntlbyte(byte);							

For a complete description of the command refer to the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful
1 error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

wrt_buffer - write buffer

SYNOPSIS

```
return = wrt_buffer(length,bcv,vu2,vu3,vu4,vu5,vu6);
unsigned return;            /* return code */
unsigned length;            /* allocation length */
int bcv;                    /* buffer control valid */
int vu2;                    /* Vendor Unique Byte 2 */
int vu3;                    /* Vendor Unique Byte 3 */
int vu4;                    /* Vendor Unique Byte 4 */
int vu5;                    /* Vendor Unique Byte 5 */
int vu6;                    /* Vendor Unique Byte 6 */
```

DESCRIPTION

This function will form and execute the command descriptor block for the write buffer command.

COMMAND DESCRIPTOR BLOCK FOR WRITE BUFFER COMMAND

bit / byte	7	6	5	4	3	2	1	0
0	3B							
1	lun(lun);			0				BCV
2	Vendor Unique Byte 2							
3	Vendor Unique Byte 3							
4	Vendor Unique Byte 4							
5	Vendor Unique Byte 5							
6	Vendor Unique Byte 6							
7	Allocation Length MSB							
8	Allocation Length LSB							
9	cntlbyte(byte);							

For a complete description of the command refer to the Common Command Set (CCS) version of the "SMALL COMPUTER SYSTEM INTERFACE (SCSI)" by American National Standard for information systems.

DEFAULT VALUE: N.A.

RETURNS:

NULL(0) successful completion
0xFFFF error

EXECUTION TYPE: I/O Driver

STATISTICS/STATUS UPDATE:
Global Stats, Function Stats and Function Status (see I/O DRIVER Status Bytes)

ERROR MESSAGES:
Implicit and Explicit Errors from Target Status, Initiator Status and I/O Status (also see Section IODVR.4)

NAME

xfermode - initialize I/O Driver and Microprogramming buffers and set up I/O Driver data transfer mechanism

SYNOPSIS

```
xfermode("mode",buf_size);
unsigned buf_size;            /* size of read/writebuffers
                                 in bytes (all OBB type
                                 transfers must specify 16K
                                 buffer sizes)
                                 SAT max size = 32K
                                 MENU max size = 16K */
char *mode;                   /* Transfer Mode/Host Memory
                                 Configuration */
```

DESCRIPTION	MODE	WRITE/ REF BUS	READ BUF	COMP REF SOURCE	COMP DATA SOURCE	TYPE OF COMPARE
High Spd Hdw Comp Virtual (*1)	HSHCV	OBB	None	OBB	SCSI BUS	OTF
High Speed R/W	HSRW	OBB	OBB	None	None	None
High Speed Copy	HSCOPY	OBB	OBB	None	None	None
High Spd Hdw Comp	HSHC	OBB	None	OBB	SCSI BUS	OTF
High Spd Sft Comp	HSSC	WMBUF	OBB	WMBUF	OBB	SOFTWARE
DMA R/W	DMARW	WMBUF	RMBUF	N.A.	N.A.	N.A.
DMA Hdw Comp	DMAHC	WMBUF	None	WMBUF	SCSI BUS	OTF
DMA Sftw Comp	DMASC	WMBUF	RMBUF	WMBUF	RMBUF	SOFTWARE
DMA Copy	DMACOPY	WMBUF	WMBUF	N.A.	N.A.	N.A.
Trans/Rec R/W	TRRW	WMBUF	RMBUF	None	None	None
Trans/Rec Sftw Compare	TRSC	WMBUF	RMBUF	WMBUF	RMBUF	SOFTWARE
Prog. I/O R/W	PIORW	WMBUF	RMBUF	None	None	None
Prog. I/O Sftw Compare	PIOSC	WMBUF	RMBUF	WMBUF	RMBUF	SOFTWARE

Legend:
- HS = High-Speed DMA Test Adapter On-Board Buffer
- HC = Hardware Data Comparison
- SC = Software Data Comparison
- OTF = On-The-Fly Hardware Data Comparison
- OBB = Test Adapter On-Board Buffer (16K)
- WMBUF = Write/Reference Buffer Memory Based
- RMBUF = Read Buffer Memory Based
- TR = Transmit/Receive State Machine for Req/Ack Handshake
- PIO = Programmed I/O Req/Ack Handshake

(*1) Utilizes 16K OBB to generate 256MB virtual memory emulation.

DESCRIPTION

The **xfermode()** function sets up the I/O Driver and Microprogramming buffer configuration. It also defines the data transfer mechanism for the I/O Driver/test adapter combination. **xfermode()** allows simulation of a number of different test adapter types as well as providing a test adapter data comparison function.

An **xfermode()** call will generate calls to the following buffer configuration functions:

setfill_buf("W"), dmaset("W") and dmaset("R")

Also see section IODVR.3.6 .

DEFAULT VALUE: DMARW 0x8000 (32K write/ref and read buffers)

RETURNS: always returns a zero

ERROR MESSAGES:

```
EXP. ER> xfermode("HSSCOPY",0x1000)
Illegal Transfer Mode                              Date/Time Stamp
```

APPENDIX B

MISCELLANEOUS

~B.Ø MISCELLANEOUS

~B.1 SDS-1 SYSTEM SOFTWARE DEFINITION

ADAPTEC PROPRIETARY SOFTWARE

The Adaptec Proprietary Software supplied with the SDS-1 is defined in APNDXB-T1.

TABLE ~APNDXB-T1. SDS-1 SYSTEM SOFTWARE

File	Description
c:\autoexec.bat	DOS Autoexecute on Boot File
c:\satlib\endts.exe	End Test Sequence for Test Results
c:\satlib\menu.exe	SDS-1 Menu Interface
c:\satlib\reshelp.exe	Resident Portion of SDS-1 Help System
c:\satlib\rptgen.exe	Report Generator
c:\satlib\rtfl.exe	Resident Test Function Library
c:\satlib\sdshelp.exe	Help System Entry from DOS
c:\satlib\titlepg.exe	Title Page for Reports
c:\c\msc\lib\ltfl.lib	Linked Portion of Test Function Library
c:\sdsbit\sdstest.bat	SDS-1 Hardware Diagnostic (*1)
c:\rm\helpindx.dat	Help Index for reshelp.exe
c:\rm\sdsrmimg.doc	SDS-1 Reference Manual Image
c:\revhist\doupdate.bat	Batch file copied from update diskette to perform system software update
c:\revhist\logprt.bat	Revision Log Print
c:\revhist\revision.log	System Software Revision Log
c:\revhist\revnotes.???	Revision (system update) notes (??? = revision number)
c:\showrev.bat	Show Revision Text prior to revision update
c:\user1\blankdv.bat	Blank Design Verification Batch File
c:\user1\blanksat.c	SAT Template
c:\user1\satmain.obj	Stand-Alone Test front end
c:\user1\tp.bat	Test Procedure Batch File

(*1) Requires SDS-1 Diagnostic Hardware Option

THIRD PARTY SOFTWARE

Additional third party software is supplied with the SDS-1. Refer to the subdirectory (Table APNDXB-T2) for its location on the system disk and a description of each item.

~B.2 DRIVE C: DIRECTORY TREE

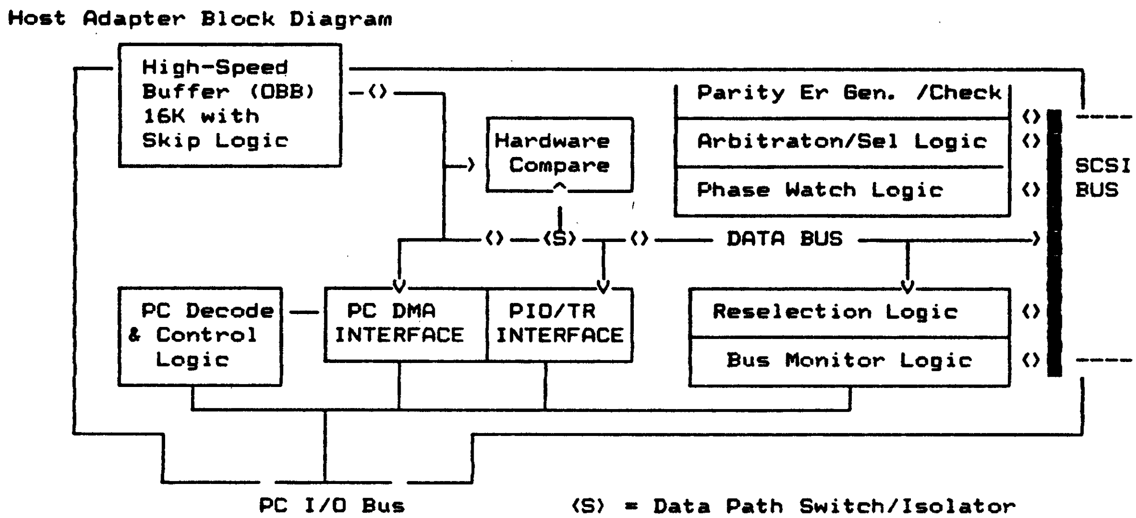
The SDS-1 system disk (C:) is comprised of the following subdirectories which are defined in Table APNDXB-T2.

TABLE ~APNDXB-T2. SDS-1 SYSTEM DRIVE DIRECTORY TREE

Subdirectory	Contents
C:\	Root directory, autoexec.bat and other start-up programs
C:\user1	User SAT generation subdirectory Includes: **BLANKDV.C** **BLANKSAT.C** **SATMAIN.OBJ** **TP.BAT**
C:\satlib	User Stand-Alone Test Library/system software Includes: **ENDTS.EXE** **MENU.EXE** **RESHELP.EXE** **RPTGEN.EXE** **RTFL.EXE** **SDSHELP.EXE** **TITLEPG.EXE**
C:\rm	SDS-1 Reference Manual Includes: **HELPINDX.DAT** **SDSRMIMG.DOC**
C:\revhist	SDS-1 Software Revision history Includes: **REVISION.LOG**
C:\sdsbit	SDS-1 hardware diagnostic: **SDSTEST.BAT**
C:\sdsbit\examples	SDS-1 programming examples
C:\c	Start of "C" subdirectories
C:\c\msc	Microsoft "C" compiler
C:\c\msc\lib	.LIB Libraries includes **LTFL.LIB**
C:\c\msc\lib\sys	System .LIB libraries used by "C" compiler
C:\dos	IBM PC-DOS utilities
C:\dos\ast	Utilities relating to multifunction card used by SDS-1
C:\dos\sk	Subdirectory containing SideKick program/utilities
C:\paint	PC PAINT PLUS program/utilities

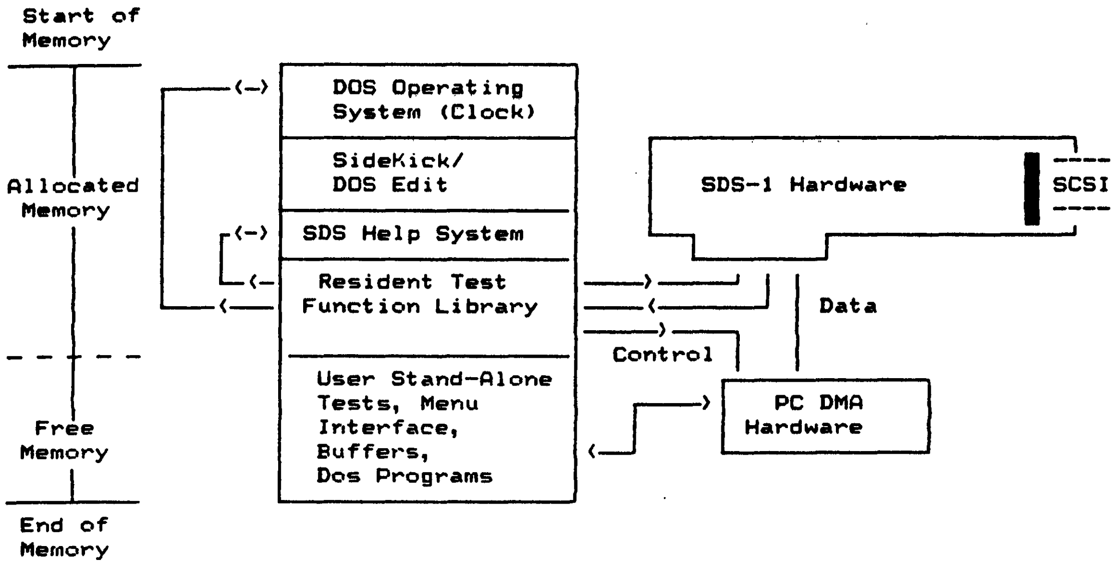
~B.3 SCSI HARDWARE INTERFACE

FIGURE ~APNDXB-F1. SCSI INTERFACE HARDWARE BLOCK DIAGRAM



~B.4 SDS-1 SOFTWARE MEMORY MAP

FIGURE ~APNDXB-F2 SYSTEM SOFTWARE MEMORY MAP



B.5 DESIGN VERIFICATION EXAMPLE

This section contains a Design Verification Example for a random access device. This is by no means a complete design verification, but rather, the start of such a test. The following is a list of the files and reports created for the Design Verification Process Example (also included are the locations of these files and reports).

B.5.1 DESIGN VERIFICATION BATCH FILE

```
ECHO OFF
TITLEPG %0 -TI="Random Access Device Design Verification" -CD="10-17-85" -RN=EM-
RANDOM-TR-01 -FO=RANDOM.TR
          REM Created 10/09/85
          REM Last Revision:    08/01/86  Correct ERRORLEVEL Logic

wrcsat -TN=
    IF ERRORLEVEL 1 GOTO BAD1

wrc401 -TN=
    IF ERRORLEVEL 1 GOTO BAD2

docdemo -TN=

obbwrcv -TN=
    IF ERRORLEVEL 1 GOTO BAD3

    ENDTS -M1="Successful Completion"  -M2="All SATs Passed"
    GOTO END

:BAD1
    ENDTS -M1="DMA Write/Read Test Failed"
    GOTO END

:BAD2
    ENDTS -M1="WRC40L Failed"
    GOTO END

:BAD3
    ENDTS -M1="OBBWRCV Failed"

:END
    ERASE *.TMP

ECHO ON
```

B.5.2 TEST PROCEDURE REPORT

"Random Access Device Design Verification"

Test Procedure

8-03-86 18:03:54

File Reference: SDS-1TP-01

Batch File: randdv.BAT
Created: "10-17-85" -RN=EM-RANDOM-TR-01 -FO=RANDOM.TR
Last Rev: 8-01-86 8:59

Created By: ______________________________

Reviewed By: ______________________________

"Random Access Device Design Verification" 8-03-86

TABLE OF CONTENTS

Page i-1

"Random Access Device Design Verification" 8-03-86

"Random Access Device Design Verification" 8-03-86

1.0 wrcsat.C Last Revision: 6-17-86 13:46

Purpose: Verify SCSI Write and Read Commands

1.0.1 1MB Write/Read/Compare

Procedure:
Fill Write Buffer with incrementing pattern
Write 1MB (256-byte) blocks to disk starting at block 100;
Read and Compare 1MB of data

"Random Access Device Design Verification" 8-03-86

2.0 wrc401.C Last Revision: 8-01-86 9:33

Purpose: Verify SCSI D.S. operation in PIO, TR and DMA Write/Read Modes

2.1 Write/Read

Procedure:

Set up initial Conditions for Write/Read Tests
    Transfer mode
    fill write buffer
    Set Loop Count (# of passes through Each Mode of Test);

2.1.1 DMA Data Transfer
    Fill Disk with 1MB of Data
    Read Disk data from disk

2.1.2 TR Data Transfer
    Fill Disk with 64K bytes of data
    Read Disk data from disk

2.1.3 PIO Data Transfer
    Fill Disk with 64K bytes of data
    Read Disk data from disk

"Random Access Device Design Verification" 8-03-86

3.0 docdemo.C Last Revision: 6-17-86 15:44

Purpose: Demonstrate Library Documentation Functions and
Report Generator Functions

3.1 Group 1 Example

Purpose: Show Basic Documentation Library Calls

Procedure:

3.1.1 logc() example
Generate a Logc message

3.1.2 logp() example
Generate a Logp paragraph

3.1.3 summary() example
Generate a Summary Message

"Random Access Device Design Verification" 8-03-86

3.2 Group 2 Example

Purpose: Show Subparagraph levels

Procedure:

3.2.1 Standard Paragraph
Generate a Logp message

3.2.1.1 Subparagraph Level 1
Generate a Logp paragraph

3.2.1.1.1 Subparagraph Level 2
Generate a Logp paragraph

3.2.1.1.1.1 Subparagraph Level 3
Generate a Logp paragraph

3.2.1.1.1.1.1 Subparagraph Level 4
Generate a Logp paragraph

"Random Access Device Design Verification" 8-03-86

3.3 Group 3 Example

Purpose: Demo Full Pages and Time Stamp Roll Off

Procedure:

3.3.1 logp fill page
Eject to top of new page
Fill a page (48 lines) using logp() functions
Show how next subpar will start on top of page

3.3.1.A End of logp fill page

3.3.2 Time Stamping
Run a string of logp() function
which increase in length

3.3.2.1 logp lines

Run a group of subpara() lines which
increase in length

3.3.2.1.1 Time Stamping Cuts off Titles ****

3.3.2.1.1.1 Time Stamping Cuts off Titles *******

3.3.2.1.1.1.1 Time Stamping Cuts off Titles also *********

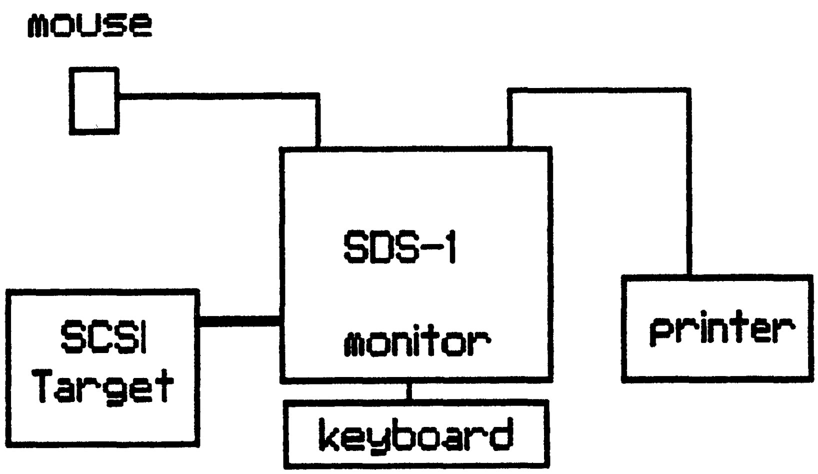
docdemo.C Page 5

"Random Access Device Design Verification" 8-03-86

3.4 Art Include Example

Purpose: Show Text/Graphics Integration Capability
of the Report Generator

System Level Block Diagram



docdemo.C Page 6

"Random Access Device Design Verification" 8-03-86

4.0 obbwrcv.C Last Revision: 6-17-86 13:47

Purpose: Demostrates OBB virtual memory, _blk functions and variable ack delay

Procedure:
1. Use get_byte() function to determine block limits
2. Read/Write Testing
   a. Fill drive via HSHCV mode with write10() func
   b. Read entire drive using _blk functions
   c. Read with random starting address and lengths
   d. Time reads in sequential manner
   e. Time reads with random starting addresses
   f. Time loop with everything random

System #1 Host i.d. = 7;
Target i.d. = 4;

Functions Tested:
set_blk
random_blk
inc_blk
set_len
random_len
inc_len

obbwrcv.C Page 7

4.1 Self-Configuration Example

Demonstrate get_byte() function
determine block limits

4.2 Read/Write Testing

4.2.1 Fill Drive via HSHCV

Fill Drive with write10() cmd
using HSHCV transfer mode

4.2.2 Read Drive w/_blk cmds

Read and Compare Entire Disk
using _blk command and HSHCV mode
of transfer

4.2.3 Read w/Random Addrs & Lens

Perform 100 read operations with
random starting addresses and
lengths

4.2.4 Time Seq Reads (3 mins)

Utilizing the user timer to
determine the number of
operations and bytes read which
can be executed in three minutes

4.2.5 Time Reads w/Random Addrs

Utilize random_blk() to read
randomly over entire disk (in
a 3 minute timed loop)

4.2.6 Timed Loop with All Random

Randomly select the type of
operation:
 6-byte read,
 6-byte write,
 10-byte read,
 or 10-byte write
Likewise randomly select the
starting block and transfer
length, executing all in a 10
minute timed loop

APPENDIX A

INPUT BATCH FILE

"Random Access Device Design Verification"   8-03-86

```
ECHO OFF
TITLEPG %0 -TI="Random Access Device Design Verification" -CD="10-!!>
          REM Created 10/09/85
          REM Last Revision:    08/01/86  Correct ERRORLEVEL Logic

wrcsat -TN=
    IF ERRORLEVEL 1 GOTO BAD1

wrc401 -TN=
    IF ERRORLEVEL 1 GOTO BAD2

docdemo -TN=

obbwrcv -TN=
    IF ERRORLEVEL 1 GOTO BAD3

    ENDTS -M1="Successful Completion"  -M2="All SATs Passed"
    GOTO END

:BAD1
    ENDTS -M1="DMA Write/Read Test Failed"
    GOTO END

:BAD2
    ENDTS -M1="WRC40L Failed"
    GOTO END

:BAD3
    ENDTS -M1="OBBWRCV Failed"

:END
    ERASE *.TMP

ECHO ON
```

APPENDIX B

Revision Log

"Random Access Device Design Verification" 8-03-86

1.0. Rev Log wrcsat.C

```
        Created: 10-9-85
Initial Release: 10-9-85
       Revision: 10-10-85  Corrected Buffer Size
                 01-23-86  SDS-1 Manual format
                 06-17-86  Enable parity
```

2.0. Rev Log wrc401.C

```
        Created: 8-24-85
Initial Release: 8-24-84
       Revision: 10-09-85 Modified for DEMO
                 01-23-86 SDS-1 Manual format
                 06-17-86 Enable parity
                 08-01-86 Include error_ok
```

3.0. Rev Log docdemo.C

```
        Created:  12-01-84
Initial Release:  08-24-85
       Revision:  10-09-85  Modified for Demo
                  01-23-86  SDS-1 Manual format
                  06-17-86  Replaced art work with system level pi!!)
```

4.0. Rev Log obbwrcv.C

```
        Created:  01-16-86
Initial Release:  N.A.
       Revision:  1.000
                  06-17-86  Enable parity
```

(THIS PAGE INTENTIONALLY LEFT BLANK)

B.5.3 TEST RESULTS REPORT

Random Access Device Design Verification

Test Data Report

8-01-86 9:36:41

File Reference: EM-RANDOM-TR-01

Batch File: randdv.BAT
Created: 10-17-85
Last Rev: 8-01-86 8:59

Created By: ________________________________

Reviewed By: ________________________________

Comments: ____

____

____

____

____

____

Random Access Device Design Verification 8-01-86

TABLE OF CONTENTS

Page i-1

Random Access Device Design Verification 8-01-86

Random Access Device Design Verification	8-01-86
1.0 Disk Data Testing	08-01-86 09:37:15
1.0.1 1MB Write/Read/Compare	08-01-86 09:37:16
1.0 Disk Data Testing PASS	08-01-86 09:37:36

Disk Page 1

```
Random Access Device Design Verification                    8-01-86

2.0 Write/Read Test  LUN 0 Device 4              08-01-86 09:37:52

2.0.1 DMA Data Transfer                          08-01-86 09:37:53
IOABORT IMPLICIT ERROR                           08-01-86 09:38:23
Cmp Error: Ref Buf(0x0000 = 0x43); SCSI Data = 0x0F;

IMP. ER> readr(0f80,80)                          08-01-86 09:38:23 FAIL
I/O Driver Error: Buffer Miscompare (0x80)
                    Bytes   Read    =     8000
                    Bytes Compared  =     7FFF
                    Miscompare Count =       1
Initiator Error: Buffer Miscompare (0x0E)

******> ERROR OK <******
 DMA Pass 1 Completed                            08-01-86 09:38:30

2.0.2 TR Data Transfer                           08-01-86 09:38:30
 TR Pass 1 Completed                             08-01-86 09:39:22

2.0.3 PIO Data Transfer                          08-01-86 09:39:22
 PIO Pass 1 Completed                            08-01-86 09:40:20

2.0 Write/Read Test  LUN 0 Device 4 PASS         08-01-86 09:40:20
```

Write/Read   Page 2

Random Access Device Design Verification	8-01-86
3.0 DOCDEMO Documentation Test	08-01-86 09:40:36
3.1 Group 1 Example	08-01-86 09:40:37
3.1.1 logc() example	08-01-86 09:40:38
3.1.2 logp() example	08-01-86 09:40:43
3.1.3 summary() example	08-01-86 09:40:46
SMRY >Summary Line #1 from DOCDEMO	08-01-86 09:40:47
3.2 Group 2 Example	08-01-86 09:40:54
3.2.1 Standard Paragraph	08-01-86 09:40:55
Text Under Standard Paragraph	08-01-86 09:40:56
3.2.1.1 Subparagraph Level 1	08-01-86 09:40:56
Text Under Subparagraph Level 1	08-01-86 09:40:57
3.2.1.1.1 Subparagraph Level 2	08-01-86 09:40:57
Text under Subparagraph Level 2	08-01-86 09:40:58
3.2.1.1.1.1 Subparagraph Level 3	08-01-86 09:40:58
Text under Subparagraph Level 3	08-01-86 09:40:59
3.2.1.1.1.1.1 Subparagraph Level 4	08-01-86 09:40:59
Text under Subparagraph Level 4	08-01-86 09:41:00
3.3 Group 3 Example	08-01-86 09:41:06

DOCDEMO Page 3

Random Access Device Design Verification 8-01-86

3.3.1 logp fill page	08-01-86 09:41:08
Page Fill Example Line 1	08-01-86 09:41:08
Page Fill Example Line 2	08-01-86 09:41:09
Page Fill Example Line 3	08-01-86 09:41:09
Page Fill Example Line 4	08-01-86 09:41:09
Page Fill Example Line 5	08-01-86 09:41:09
Page Fill Example Line 6	08-01-86 09:41:09
Page Fill Example Line 7	08-01-86 09:41:10
Page Fill Example Line 8	08-01-86 09:41:10
Page Fill Example Line 9	08-01-86 09:41:10
Page Fill Example Line 10	08-01-86 09:41:10
Page Fill Example Line 11	08-01-86 09:41:11
Page Fill Example Line 12	08-01-86 09:41:11
Page Fill Example Line 13	08-01-86 09:41:11
Page Fill Example Line 14	08-01-86 09:41:11
Page Fill Example Line 15	08-01-86 09:41:12
Page Fill Example Line 16	08-01-86 09:41:12
Page Fill Example Line 17	08-01-86 09:41:12
Page Fill Example Line 18	08-01-86 09:41:12
Page Fill Example Line 19	08-01-86 09:41:13
Page Fill Example Line 20	08-01-86 09:41:13
Page Fill Example Line 21	08-01-86 09:41:13
Page Fill Example Line 22	08-01-86 09:41:13
Page Fill Example Line 23	08-01-86 09:41:14
Page Fill Example Line 24	08-01-86 09:41:14
Page Fill Example Line 25	08-01-86 09:41:14
Page Fill Example Line 26	08-01-86 09:41:14
Page Fill Example Line 27	08-01-86 09:41:14
Page Fill Example Line 28	08-01-86 09:41:15
Page Fill Example Line 29	08-01-86 09:41:18
Page Fill Example Line 30	08-01-86 09:41:19
Page Fill Example Line 31	08-01-86 09:41:19
Page Fill Example Line 32	08-01-86 09:41:19
Page Fill Example Line 33	08-01-86 09:41:19
Page Fill Example Line 34	08-01-86 09:41:19
Page Fill Example Line 35	08-01-86 09:41:20
Page Fill Example Line 36	08-01-86 09:41:20
Page Fill Example Line 37	08-01-86 09:41:20
Page Fill Example Line 38	08-01-86 09:41:21
Page Fill Example Line 39	08-01-86 09:41:21
Page Fill Example Line 40	08-01-86 09:41:21
Page Fill Example Line 41	08-01-86 09:41:21
Page Fill Example Line 42	08-01-86 09:41:21
Page Fill Example Line 43	08-01-86 09:41:22
Page Fill Example Line 44	08-01-86 09:41:22
Page Fill Example Line 45	08-01-86 09:41:22
Page Fill Example Line 46	08-01-86 09:41:22
Page Fill Example Line 47	08-01-86 09:41:23
Page Fill Example Line 48	08-01-86 09:41:23

DOCDEMO Page 4

```
Random Access Device Design Verification                   8-01-86

3.3.1.A End of logc fill page                    08-01-86 09:41:24

3.3.2 Time Stamping                              08-01-86 09:41:25

3.3.2.1 logp lines                               08-01-86 09:41:26
 The Line Gets Longer *                          08-01-86 09:41:26
 The Line Gets Longer **                         08-01-86 09:41:26
 The Line Gets Longer ***                        08-01-86 09:41:27
 The Line Gets Longer ****                       08-01-86 09:41:27
 The Line Gets Longer *****                      08-01-86 09:41:27
 The Line Gets Longer ******                     08-01-86 09:41:27
 The Line Gets Longer *******                    08-01-86 09:41:28
 The Line Gets Longer ********                   08-01-86 09:41:28
 The Line Gets Longer *********                  08-01-86 09:41:28
 The Line Gets Longer **********                 08-01-86 09:41:28
 The Line Gets Longer ***********                08-01-86 09:41:29
 The Line Gets Longer ************               08-01-86 09:41:29
 The Line Gets Longer *************              08-01-86 09:41:29
 The Line Gets Longer **************             08-01-86 09:41:29
 The Line Gets Longer ***************            08-01-86 09:41:30
 The Line Gets Longer ****************           08-01-86 09:41:30
 The Line Gets Longer *****************          08-01-86 09:41:30
 The Line Gets Longer ******************         08-01-86 09:41:30
 The Line Gets Longer *******************        08-01-86 09:41:31
 The Line Gets Longer ********************       08-01-86 09:41:31
 The Line Gets Longer *********************      08-01-86 09:41:31
 The Line Gets Longer **********************     08-01-86 09:41:31
 The Line Gets Longer ***********************    08-01-86 09:41:32
 The Line Gets Longer ************************   08-01-86 09:41:32
 The Line Gets Longer *************************  08-01-86 09:41:32
 The Line Gets Longer ************************** 08-01-86 09:41:32
 The Line Gets Longer ***************************         09:41:33
 The Line Gets Longer ****************************        09:41:33
 The Line Gets Longer *****************************       09:41:33
 The Line Gets Longer ******************************      09:41:33
 The Line Gets Longer *******************************     09:41:34
 The Line Gets Longer ********************************    09:41:34
 The Line Gets Longer *********************************   09:41:34
 The Line Gets Longer **********************************  09:41:34
 The Line Gets Longer *********************************** 09:41:34
 The Line Gets Longer ************************************
 The Line Gets Longer *************************************
 The Line Gets Longer **************************************
 no pass or fail after this test                 08-01-86 09:41:35

3.3.2.1.1 Time Stamping Cuts off Titles  ****    08-01-86 09:41:36

3.3.2.1.1.1 Time Stamping Cuts off Titles  *******        09:41:37

3.3.2.1.1.1.1 Time Stamping Cuts off Titles also *******  09:41:38
 SMRY >Group 3 Examples passed with flying colors         09:41:38


DOCDEMO                      Page 5
```

Random Access Device Design Verification	8-01-86
3.4 Art Include Example	08-01-86 09:41:39
3.0 DOCDEMO Documentation Test PASS	08-01-86 09:41:40

DOCDEMO Page 6

```
Random Access Device Design Verification                          8-01-86

4.0 Random Function Testing                              08-01-86 09:41:59

4.1 Self Configuration Example                           08-01-86 09:42:00
  Drive Parameters:  Last Block Address = 0x4C7F         08-01-86 09:42:01
                             Block Size = 0x100          08-01-86 09:42:02

4.2 Read/Write Testing                                   08-01-86 09:42:03

4.2.1 Fill Drive via HSHCV                               08-01-86 09:42:04

  I/O OPs   TGT CKs   BYTs WR   BYTs RD   BYTs CP   CP ERs
 -------- -------- -------- -------- -------- --------
        2        0   4c8000        8        0        0      09:42:30

4.2.2 Read Entire Drive Using _blk commands              08-01-86 09:42:31

  I/O OPs   TGT CKs   BYTs WR   BYTs RD   BYTs CP   CP ERs
 -------- -------- -------- -------- -------- --------
        1        0        0   4c8000   4c8000        0      09:42:53
 Last Read Command Statistics:                           08-01-86 09:42:53
                                          Bytes Written = 0x       0
                                             Bytes Read = 0x  4C8000
                                         Bytes Compared = 0x  4C8000
                                         Compare Errors = 0x       0

4.2.3 Read with Random Starting Addresses and Lengths             09:42:55

  I/O OPs   TGT CKs   BYTs WR   BYTs RD   BYTs CP   CP ERs
 -------- -------- -------- -------- -------- --------
       64        0        0  30f1500  30f1500        0      09:46:34

4.2.4 Timed Reads (three minutes) in Sequential Manner            09:46:35
Elasped Time = 00284.23;       User_Timer = 00000.00;             09:46:35
Elasped Time = 00463.24;       User_Timer = 00179.01;             09:49:34

  I/O OPs   TGT CKs   BYTs WR   BYTs RD   BYTs CP   CP ERs
 -------- -------- -------- -------- -------- --------
      252        0        0  2520000  2520000        0      09:49:35

4.2.5 Time Reads (3 mins) with Random Starting Addresses          09:49:36

  I/O OPs   TGT CKs   BYTs WR   BYTs RD   BYTs CP   CP ERs
 -------- -------- -------- -------- -------- --------
      1ee        0        0  1ee0000  1ee0000        0      09:52:35

4.2.6 Timed Loop (10 minutes) With All Random            08-01-86 09:52:36

  I/O OPs   TGT CKs   BYTs WR   BYTs RD   BYTs CP   CP ERs
 -------- -------- -------- -------- -------- --------
        0        0        0        0        0        0      09:52:37
       7f        0  1344700  14dcd00  14dcd00        0      10:02:38


Random                         Page 7
```

Random Access Device Design Verification 8-01-86

107	0	23f3300	29bea00	29bea00	0	10:12:48
1b0	0	3c97d00	3da7100	3da7100	0	10:23:02
260	0	4ebd400	5173300	5173300	0	10:33:05
2ee	0	6167d00	6761500	6761500	0	10:43:08
37d	0	7391700	7ca1700	7ca1700	0	10:53:08

4.0 Random Function Testing PASS 08-01-86 10:53:08

APPENDIX A

INPUT BATCH FILE

Random Access Device Design Verification 8-01-86

```
ECHO OFF
TITLEPG %0 -TI="Random Access Device Design Verification" -CD="10-!!)
          REM Created 10/09/85
          REM Last Revision:    08/01/86  Correct ERRORLEVEL Logic

wrcsat -TN=
    IF ERRORLEVEL 1 GOTO BAD1

wrc401 -TN=
    IF ERRORLEVEL 1 GOTO BAD2

docdemo -TN=

obbwrcv -TN=
    IF ERRORLEVEL 1 GOTO BAD3

    ENDTS -M1="Successful Completion"  -M2="All SATs Passed"
    GOTO END

:BAD1
    ENDTS -M1="DMA Write/Read Test Failed"
    GOTO END

:BAD2
    ENDTS -M1="WRC40L Failed"
    GOTO END

:BAD3
    ENDTS -M1="OBBWRCV Failed"

:END
    ERASE *.TMP

ECHO ON
```

Page A-2

APPENDIX B

TEST DATA SUMMARY REPORT

Random Access Device Design Verification 8-01-86

Random Access Device Design Verification Test Data Summary
EM-RANDOM-TR-01

Batch File: randdv.BAT; Created: 10-17-85
Batch File Last Revision: 8-01-86 8:59
Test Sequence Started: 8-01-86 9:36:41
Test Sequence Concluded: 8-01-86 10:53:21

SMRY)Summary Line #1 from DOCDEMO 08-01-86 09:40:47
SMRY)Group 3 Examples passed with flying colors 09:41:38

ENDTS Messages

Successful Completion
All SATs Passed

Page B-2

B.5.4 SAT SOURCE CODE

wrcsat.C 6-17-86 20:53:19 PAGE 1

```
*
/*
-DOC
-REV
        Created: 10-9-85
Initial Release: 10-9-85
       Revision: 10-10-85  Corrected Buffer Size
                 01-23-86  SDS-1 Manual format
                 06-17-86  Enable parity
-REV

Purpose: Verify SCSI Write and Read Commands

-PT="1MB Write/Read/Compare"

Procedure:
    Fill Write Buffer with incrementing pattern
    Write 1MB (256-byte) blocks to disk starting at block 100;
    Read and Compare 1MB of data

-DOC */



user_test()
{
    test("Disk Data Testing");
    xfermode("DMAHC",0x8000);
    bus_logen(1);
    parity(1);
    tid(4);
    paragph("1MB Write/Read/Compare");
    filli(04,0,0x8000);
    writer10(0,0x1001,4000);
    readr10(0,0x1001,4000);
}
```

```
/*
-DOC
-REV
         Created: 8-24-85
 Initial Release: 8-24-84
        Revision: 10-09-85 Modified for DEMO
                  01-23-86 SDS-1 Manual format
                  06-17-86 Enable parity
                  08-01-86 Include error_ok
-REV

Purpose: Verify SCSI D.S. operation in PIO, TR and DMA Write/Read
         Modes

-GT="Write/Read"

Procedure:

Set up initial Conditions for Write/Read Tests
    Transfer mode
    fill write buffer
    Set Loop Count (# of passes through Each Mode of Test);

-PT="DMA Data Transfer"
    Fill Disk with 1MB of Data
    Read Disk data from disk

-PT="TR Data Transfer"
    Fill Disk with 64K bytes of data
    Read Disk data from disk

-PT="PIO Data Transfer"
    Fill Disk with 64K bytes of data
    Read Disk data from disk

-DOC */
/* -COD */


user_test()
{
    unsigned pass_count;        /* number of passes to execute */
    unsigned i,j,k,l;
    char dummy[100];

    pass_count = 1;

    test("Write/Read Test  LUN 0 Device 4");
    ioto(10);
    paragph("DMA Data Transfer");
    bcu(1);
    xfermode("DMAHC",0x8000);
    arbmode("HDW");
    selmode("SMART");
```

```
        bus_logen(1);
        parity(1);
        statsen(1);
        tid(4);
        lun(0);
        busywait(1);
        sense(0x10);
        dmarst("W");
        fillpr(0,0,0x8000);
        bcu(0);
        for (i = 1; i  (= pass_count; i++) {
                                        /* Pass Count Loop */
            ucname(0,"Write cnt");
            for (j=0;  j ( 32;   j++) {
                                        /* MB Count Loop (write) */
                overbcw(j*0x80,0x100,0,0x8000);
                writer(j*0x80,0x80);
                ucinc(0,0x80);
            }

            sense(0x10);
            ucname(1,"Read cnt");
            for (j=0;  j ( 31;   j++) {
                                        /* MB Count Loop (read) */
                overbcw(j*0x80,0x100,0,0x8000);
                readr(j*0x80,0x80);
                ucinc(1,0x80);
            }
                                        /* Cause a 1 byte Compare Error */
            overbcw(j*0x80,0x100,0,0x8000);
            fillk("43",0x00,0x01);
            readr(j*0x80,0x80);
            error_ok("DISPLAY");

            sprintf(dummy,"DMA Pass %u Completed",i);
            logp(dummy);
        }
        paragph("TR Data Transfer");
        bcu(1);
        xfermode("TRRW",0x8000);
        fillpr(0,0,0x8000);
        bcu(0);
        ioto(60);
        for (i = 1; i  (= pass_count; i++) {
                                        /* Pass Count Loop */
            ucname(0,"Write cnt");
            for (j=0;  j ( 2;   j++) {
                                        /* MB Count Loop (write) */
                overbcw(j*0x80,0x100,0,0x8000);
                writer(j*0x80,0x80);
                ucinc(0,0x80);
            }

            sense(0x10);
```

```
                ucname(1,"Read cnt");
                for (j=0;  j < 2;   j++) {
                                                /* MB Count Loop (read)  */
                    overbcw(j*0x80,0x100,0,0x8000);
                    readr(j*0x80,0x80);
                    ucinc(1,0x80);
                }
                sprintf(dummy,"TR Pass %u Completed",i);
                logp(dummy);
            }
            paragph("PIO Data Transfer");
            bcu(1);
            xfermode("PIORW",0x8000);
            fillpr(0,0,0x8000);
            bcu(0);
            for (i = 1; i  <= pass_count; i++) {
                                                /* Pass Count Loop */
                ucname(0,"Write cnt");
                for (j=0;  j < 2;   j++) {
                                                /* MB Count Loop (write) */
                    overbcw(j*0x80,0x100,0,0x8000);
                    writer(j*0x80,0x80);
                    ucinc(0,0x80);
                }

                sense(0x10);
                ucname(1,"Read cnt");
                for (j=0;  j < 2;   j++) {
                                                /* MB Count Loop (read) */
                    overbcw(j*0x80,0x100,0,0x8000);
                    readr(j*0x80,0x80);
                    ucinc(1,0x80);
                }
                sprintf(dummy,"PIO Pass %u Completed",i);
                logp(dummy);
            }
        }

        /* -COD */
```

```
docdemo.C                                       6-17-86  20:53:51  PAGE     1

/* -DOC

-REV
        Created:  12-01-84
Initial Release:  08-24-85
       Revision:  10-09-85  Modified for Demo
                  01-23-86  SDS-1 Manual format
                  06-17-86  Replaced art work with system level pic
ture
-REV


Purpose:  Demonstrate Library Documentation Functions and
          Report Generator Functions

-DOC */

user_test()
{
                                          /* send test (section) title to */
    test("DOCDEMO Documentation Test");  /* to report generator */
    group_1();                           /* group 1 example */
    delays(5);
    group_2();                           /* group 2 example */
    delays(5);
    group_3();                           /* group 3 example */
    group_4();                           /* group 3 example */


}

/* -DOC
-GT="Group 1 Example"

Purpose: Show Basic Documentation Library Calls

Procedure:

-PT="logc() example"
   Generate a Logc message

-PT="logp() example"
   Generate a Logp paragraph

-PT="summary() example"
   Generate a Summary Message

-DOC */
                                          /* -COD */
group_1()
{
    group("Group 1 Example");

```

```
        paragph("logc() example");
        logc("Message Generated by logc function");
        logc("2nd message followed by three second delay");
        delays(3);

        paragph("logp() example");
        logc("Message to both printer and console by logp function");
        logc("2nd message followed by two second delay");
        delays(2);

        paragph("summary() example");
        summary("Summary Line #1 from DOCDEMO");
        delays(1);

    }

                                            /* -COD */

    /* -DOC
    -GT="Group 2 Example"

    Purpose: Show Subparagraph levels

    Procedure:

    -PT="Standard Paragraph"
       Generate a Logp message

    -PT="Subparagraph Level 1" -RN=1
       Generate a Logp paragraph

    -PT="Subparagraph Level 2" -RN=1.1
       Generate a Logp paragraph

    -PT="Subparagraph Level 3" -RN=1.1.1
       Generate a Logp paragraph

    -PT="Subparagraph Level 4" -RN=1.1.1.1
       Generate a Logp paragraph

    -DOC */
                                            /* -COD */
    group_2()
    {
        group("Group 2 Example");

        paragph("Standard Paragraph");
        logp("Text Under Standard Paragraph");

        subpar("Subparagraph Level 1","1");
        logp("Text Under Subparagraph Level 1");

        subpar("Subparagraph Level 2","1.1");
        logp("Text under Subparagraph Level 2");
```

```

    subpar("Subparagraph Level 3","1.1.1");
    logp("Text under Subparagraph Level 3");

    subpar("Subparagraph Level 4","1.1.1.1");
    logp("Text under Subparagraph Level 4");
}
                                    /* -COD */
/* -DOC
-GT="Group 3 Example"

Purpose: Demo Full Pages and Time Stamp Roll Off

Procedure:

-PT="logp fill page"
   Eject to top of new page
   Fill a page (48 lines) using logp() functions
   Show how next subpar will start on top of page

-PT="End of logp fill page" -RN=A

-PT="Time Stamping"
   Run a string of logp() function
   which increase in length

-PT="logp lines" -RN=1


   Run a group of subpara() lines which
   increase in length


-PT="Time Stamping Cuts off Titles ****" -RN=1.1

-PT="Time Stamping Cuts off Titles  *******" -RN=1.1.1

-PT="Time Stamping Cuts off Titles also *********" -RN=1.1.1.1


-DOC */
                                    /* -COD */
group_3()
{
    int i,j,k;
    char dummy[100];
    char dummy1[100];

    group("Group 3 Example");
    page();
    paragph("logp fill page");
    for (i = 1; i <=48; i++) {
        sprintf(dummy,"Page Fill Example Line %2d",i);
        logp(dummy);
```

```
docdemo.C                                          6-17-86  20:53:51  PAGE     4

          }
          subpar("End of logc fill page","A");

          paragph("Time Stamping");
          subpar("logp lines","1");

          for (i = 1; i <= 38; i++) {
              for (j = 0; j <= (i-1); j++)
                  dummy[j] = '*';
              dummy[j] = '\0';
              sprintf(dummy1,"The Line Gets Longer %s",dummy);
              logp(dummy1);
          }
          logp("no pass or fail after this test");

          subpar("Time Stamping Cuts off Titles  ****","1.1");

          subpar("Time Stamping Cuts off Titles  *******","1.1.1");

          subpar("Time Stamping Cuts off Titles also *******","1.1.1.1");

          summary("Group 3 Examples passed with flying colors");

                                               /* -COD */
      }

      /* -DOC
      -GT="Art Include Example"

      Purpose: Show Text/Graphics Integration Capability
               of the Report Generator


      System Level Block Diagram

      -AI="system.pic"

      -DOC */
      group_4()
      {
      group("Art Include Example");
```

```
obbwrcv.C                                   6-17-86  20:54:03  PAGE     1

/*-DB=;
-DOC
; -REV
;          Created:  01-16-86
; Initial Release:  N.A.
;         Revision:  1.000
;                    06-17-86  Enable parity
; -REV
;
;Purpose:  Demostrates OBB virtual memory, _blk functions and
;          variable ack delay
;
;Procedure:  1. Use get_byte() function to determine block limits
;            2. Read/Write Testing
;                a. Fill drive via HSHCV mode with write10() func
;                b. Read entire drive using _blk functions
;                c. Read with random starting address and lengths
;                d. Time reads in sequential manner
;                e. Time reads with random starting addresses
;                f. Time loop with everything random
;
;   System #1  Host i.d. = 7;
;              Target i.d. = 4;
;
;Functions Tested:     set_blk
;                      random_blk
;                      inc_blk
;                      set_len
;                      random_len
;                      inc_len
;
-DOC */
                                 /* Constant Definitions */
     #define HOST_ID 0x07
     #define TARGET_ID 0x04

user_test()
{
                                 /* Variable Definitions */
    int i;                       /* i variable */
    unsigned long last_block_num;  /* last block number on drive */
    unsigned long f_bw, f_br, f_bc, f_ce;  /* stats variables */
    unsigned block_size;         /* drive block size */
    unsigned long new_start;     /* new starting block address */
    unsigned long down_count;    /* length of disk */
    unsigned long start_blk;     /* starting block */
    unsigned long block;         /* block */
    unsigned long get_f_stats();  /* function status */
    unsigned len,akd;            /* length & ack delay variables */
    unsigned op_type;            /* operation type */
    unsigned tv;                 /* timer value */
    char dummy[100];             /* dummy string */

    test("Random Function Testing");
```

```
        group("Self Configuration Example");
                                        /* -DOC
                                        ; -GT="Self-Configuration Example"
                                        ;
                                        ;Demonstrate get_byte() function
                                        ;determine block limits
                                        ;
                                        ;  -DOC */
        xfermode("DMARW",0x100);        /* DMARW mode w/0x100 buf size */
        reset();                        /* reset I/O Driver and SCSI bus */
        ioto(600);                      /* long time-out w/two systems
                                           competing for bus */
        bcu(1);                         /* buffer/command frame update */
        arbmode("HDW");                 /* hardware arbitration */
        selmode("SMART");               /* select SMART mode */
        parity(1);                      /* SCSI parity enabled */
        bus_logen(1);                   /* state bus log enabled */
        ackdelay(0x0000);               /* 0 ack delay */
        statsen(1);                     /* statistics enabled */
        tid(TARGET_ID);                 /* set target ID */
        iid(0,HOST_ID);                 /* set initiator ID */
        lun(0);                         /* logical unit number is 0 *
        iea("LOGH");                    /* log and halt on error */
        readcap(0,01,0);                /* read capacity */
        last_block_num =  ((unsigned long)get_byte("R",0) << 24) +
                          ((unsigned long)get_byte("R",1) << 16) +
                          ((unsigned long)get_byte("R",2) <<  8) +
                          (unsigned long)get_byte("R",3);

        sprintf(dummy,"Drive Parameters:  Last Block Address = 0x%lX",
            last_block_num);
        logp(dummy);                    /* print last block address msg */
        block_size =  ((unsigned)get_byte("R",6) <<  8) +
                       (unsigned)get_byte("R",7);

        sprintf(dummy,"                                   Block Size = 0x%X",
            block_size);
        logp(dummy);                    /* print block size msg */


        group("Read/Write Testing");

        paragph("Fill Drive via HSHCV");
                                        /* -DOC
                                        ; -GT="Read/Write Testing"
                                        ;
                                        ; -PT="Fill Drive via HSHCV"
                                        ;
                                        ; Fill Drive with write10() cmd
                                        ; using HSHCV transfer mode
                                        ;
                                        ; -DOC */
        xfermode("HSHCV",0x4000);       /* set HSHCV mode & buffer size */
        fillpr(0x87,0,0x4000);          /* fill buffer */
```

```
        down_count = last_block_num +1L;  /* number of blocks */
        start_blk = OL;                    /* starting address */
        while (down_count > 0xFFFFL) {  /* separate write commands if
                                           greater than 0xFFFF */
            writer10(0,start_blk,0xFFFF);  /* write maximum allowed */
            start_blk = start_blk + 0xFFFFL;  /* mod starting addr */
            down_count = down_count - 0xFFFFL; /* decrement blk cnt */
        }
                                       /* handle last write */
        writer10(0,start_blk,(unsigned)down_count);  /* filled disk */
        rptstats(1);                   /* report stats with header on */

        paragph("Read Entire Drive Using _blk commands");
                                       /* -DOC
                                       ; -PT="Read Drive w/_blk cmds"
                                       ;
                                       ; Read and Compare Entire Disk
                                       ; using _blk command and HSHCV mode
                                       ; of transfer
                                       ;
                                       ; -DOC */
        blk_size(block_size);          /* set block size */
        stats_reset("ALL");            /* reset global stats */
        set_blk(0x01);                 /* start at block zero */
        set_len(0XFFFF);               /* read 0xFFFF blocks at a time */
        dmaset_vblk("W");              /* set the virtual starting addr */
        down_count = last_block_num + 1L;  /* get number of blocks */
        while (down_count > 0xFFFFL) {  /* as with the writes, separate
                                           if block number greater than
                                           0xFFFF */
            readr10_blk();             /* read blocks */
            inc_blk(0xFFFF);           /* increment by 0xFFFF */
            down_count = down_count - 0xFFFFL;  /* decrement blk cnt */
        }
        set_len((unsigned)down_count);  /* handle last read */
        readr10_blk();                 /* read blocks */
        rptstats(1);                   /* report stats with header on */

                                       /* Demonstrate get_f_stats() */
        f_bw = get_f_stats("BW");      /* get bytes written */
        f_br = get_f_stats("BR");      /* get bytes read */
        f_bc = get_f_stats("BC");      /* get bytes compared */
        f_ce = get_f_stats("CE");      /* get compare errors */

                                       /* print stats to log device */
        sprintf(dummy,"Last Read Command Statistics:");
        logp(dummy);
        sprintf(dummy,
            "                                Bytes Written = 0x%8lX",
            f_bw);
        logp(dummy);
        sprintf(dummy,
            "                                   Bytes Read = 0x%8lX",
            f_br);
```

```
        logp(dummy);
        sprintf(dummy,
            "                               Bytes Compared = 0x%8lX",
            f_bc);
        logp(dummy);
        sprintf(dummy,
            "                               Compare Errors = 0x%8lX",
            f_ce);
        logp(dummy);

        paragph("Read with Random Starting Addresses and Lengths");
                                    /* -DOC
                                    ; -PT="Read w/Random Addrs & Lens"
                                    ;
                                    ; Perform 100 read operations with
                                    ; random starting addresses and
                                    ; lengths
                                    ;
                                    ; -DOC */
        stats_reset("ALL");         /* reset global statistics */
        for (i =1; i <= 100; i++) {
            len = random_len(1,0x1000);  /* transfer length limit */
            block = random_blk(0L,last_block_num-(unsigned long)len+1);
            dmaset_vblk("W");       /* set memory pointer */
            readr10_blk();          /* perform read */
                                    /* check for transfer length */
            f_br = get_f_stats("BR");  /* check for read failure */
            if (f_br != (unsigned long)block_size*(unsigned long)len) {
                fail();
                sprintf(dummy,
                "Number of bytes read = 0x%08lX; Should be = 0x%08lX;",
                    f_br,(block_size *len));
                logp(dummy);        /* print to log device */
            }
        }
        rptstats(1);                /* report global stats */

        paragph("Timed Reads (three minutes) in Sequential Manner");
                                    /* -DOC
                                    ; -PT="Time Seq Reads (3 mins)"
                                    ;
                                    ; Utilizing the user timer to
                                    ; determine the number of
                                    ; operations and bytes read which
                                    ; can be executed in three minutes
                                    ;
                                    ; -DOC */

        stats_reset("ALL");         /* reset statistics */
        tmrset(0x0);                /* set timer to start at 0 */
        tmrstart("Up");             /* start timer counting up */
        rpttmr();                   /* output timer to log */
        tv = tmrvalue();            /* get current time */
        sprintf(dummy,"Timer Value = 0x%04X",tv);  /* display timer */
```

```
        set_len(0x100);                   /* 256 block transfers */
        set_blk(0x0L);                    /* starting block */
        while ((tv = tmrvalue()) < (unsigned)( 3*60)) {  /* 3 mins */
            dmaset_vblk("W");             /* set the virtual starting addr */
            readr_blk();                  /* perform read */
            new_start = inc_blk(0x100);  /* new starting block */
            if (new_start + 0x100 > last_block_num) {  /* if starting
                                             block is greater than last
                                             block number, */
                set_blk(0x01);            /* start over on drive */
            }
        }
        tmrstop();                        /* end of three minute loop */
        sprintf(dummy,"Timer Value = 0x%04X",tv);  /* display timer */
        rpttmr();                         /* report timer to log */
        rptstats(1);                      /* report statistics */

        paragph("Time Reads (3 mins) with Random Starting Addresses");
                                          /* -DOC
                                          ; -PT="Time Reads w/Random Addrs"
                                          ;
                                          ; Utilize random_blk() to read
                                          ; randomly over entire disk (in
                                          ; a 3 minute timed loop)
                                          ;
                                          ; -DOC */
        stats_reset("ALL");               /* reset statistics */
        tmrset(0x0);                      /* set timer to start at 0 */
        tmrstart("Up");                   /* start timer counting up */

        set_len(0x100);                   /* 256 block transfers */
        set_blk(0x0L);                    /* starting block */
        while (tmrvalue() < (unsigned)( 3*60)) {  /* 3 min count */
            dmaset_vblk("W");             /* set the virtual starting addr */
            readr_blk();                  /* perform read */
                                          /* calculate random block */
            random_blk(0L,(last_block_num - (unsigned long)0xFF));
        }
        tmrstop();                        /* end of three minute loop */
        rptstats(1);                      /* report statistics */


        paragph("Timed Loop (10 minutes) With All Random");
                                          /* -DOC
                                          ; -PT="Timed Loop with All Random"
                                          ;
                                          ; Randomly select the type of
                                          ; operation:
                                          ;      6-byte read,
                                          ;      6-byte write,
                                          ;      10-byte read,
                                          ;   or 10-byte write
                                          ; Likewise randomly select the
                                          ; starting block and transfer
```

```
                                        ; length, executing all in a 10
                                        ; minute timed loop
                                        ; -DOC */
        stats_reset("ALL");             /* reset statistics */
        rptstats(1);                    /* report statistics */

        for (i = 0; i < 6; i++) {       /* one-hour test */
            tmrset(0x0);                /* set timer to start at 0 */
            tmrstart("Up");             /* start timer counting up */

            ioto(1200);                 /* set long for long random acks */
            while (tmrvalue() < (10*60)) {  /* count for ten minutes */
                                        /* calc trans len & start addr */
                len = random_len(1,0x1000);  /* transfer len limit */
                block =
                    random_blk(01,last_block_num-(unsigned long)len+1);
                dmaset_vblk("W");       /* set the virtual starting addr */
                akd = rand();           /* get random ack delay */
                ackdelay(0x0FF & akd);  /* set fixed delay */
                op_type = 0x0003 & rand();  /* use C library random
                                           number to choose type of
                                           operation */
                if (op_type == 0) {
                   readr_blk();         /* six-byte read command */
                }
                else if  (op_type == 1) {
                   writer_blk();        /* six-byte write command */
                }
                else if (op_type == 2) {
                  readr10_blk();        /* 10-byte read command */
                }
                else {
                   writer10_blk();      /* 10-byte write command */
                }
            }
            tmrstop();                  /* end of 10 minute timed loop */
            rptstats(0);                /* report statistics no header */
        }
    }
```

(THIS PAGE INTENTIONALLY LEFT BLANK)

APPENDIX C

CONFIGURATION CONTROL

C.Ø CONFIGURATION CONTROL

C.1 REFERENCE MANUAL CONFIGURATION CONTROL

In order to keep SDS-1 customers current this Reference Manual is a tightly controlled document. Each Page is numbered and stamped with a revision label (Lower Right-Hand Corner.) The following pages represent the current state of this copy of the Reference Manual.

Section	Page Number	1.0	1.1	Type	Revision Level 1.2	Type	_._	_._	_._	_._	_._
Cover		1.0	1.1	C	1.2	C					
Preface		1.0	1.1	TP							
TOC	i	1.0	1.1	T	1.2	T					
TOC	ii	1.0	1.1	T	1.2	T					
TOC	iii	1.0	1.1	T	1.2	T					
TOC	iv	1.0	1.1	T	1.2	T					
TOC	v	1.0	1.1	T	1.2	T					
TOC	vi	1.0	1.1	T	1.2	T					
TOC	vii	1.0	1.1	T	1.2	T					
TOC	viii	1.0	1.1	T	1.2	T					
TOC	ix	1.0	1.1	T	1.2	T					
TOC	x				1.2	T					
INTRO-	1	1.0	1.1	T	1.2	C					
INTRO-	2	1.0	1.1	T	1.2	C					
INTRO-	3	1.0	1.1	T	1.2	C					
INTRO-	4	1.0	1.1	T	1.2	C					
INTRO-	5	1.0	1.1	T	1.2	C					
INTRO-	6	1.0	1.1	T	1.2	T					
INTRO-	7	1.0	1.1	T	1.2	C					
INTRO-	8	1.0			1.2	C					
INTRO-	9	1.0			1.2	C					
INTRO-	10	1.0			1.2	T					
INTRO-	11	1.0			1.2	C					
INTRO-	12	1.0	1.1	T	1.2	T					
INTRO-	13				1.2	T					
INTRO-	14				1.2	T					
MENU-	1	1.0	1.1	T	1.2	C					
MENU-	2	1.0	1.1	T	1.2	C					
MENU-	3	1.0	1.1	T	1.2	C					
MENU-	4	1.0	1.1	T							
MENU-	5	1.0	1.1	T	1.2	C					
MENU-	6	1.0	1.1	T	1.2	C					
MENU-	7	1.0	1.1	T	1.2	C					
MENU-	8	1.0	1.1	T	1.2	C					
MENU-	9	1.0	1.1	T	1.2	C					
MENU-	10	1.0	1.1	T	1.2	C					
MENU-	11	1.0	1.1	T	1.2	C					
MENU-	12	1.0	1.1	T	1.2	C					
MENU-	13	1.0	1.1	T	1.2	C					
MENU-	14		1.1	T	1.2	T					
MENU-	15		1.1	T	1.2	C					
MENU-	16		1.1	T	1.2	C					
MENU-	17		1.1	T	1.2	C					
MENU-	18		1.1	T	1.2	T					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type Revision Level	_._	_._	_._	_._	_._
SAT-	1	1.0			1.2	C					
SAT-	2	1.0			1.2	C					
SAT-	2a				1.2	C					
SAT-	3	1.0			1.2	T					
SAT-	3a				1.2	T					
SAT-	4	1.0			1.2	C					
SAT-	5	1.0			1.2	C					
SAT-	6	1.0			1.2	T					
SAT-	7	1.0			1.2	C					
SAT-	8	1.0			1.2	C					
SAT-	9	1.0			1.2	T					
SAT-	10	1.0			1.2	T					
SAT-	11	1.0			1.2	T					
SAT-	12	1.0									
SAT-	13	1.0									
SAT-	14	1.0									
SAT-	15	1.0									
SAT-	16	1.0			1.2	C					
SAT-	17	1.0			1.2	C					
SAT-	18	1.0			1.2	C					
SAT-	19	1.0									
SAT-	20	1.0									
SAT-	21	1.0			1.2	T					
SAT-	22	1.0			1.2	T					
DV-	1	1.0	1.1	T	1.2	T					
DV-	2	1.0	1.1	T	1.2	T					
DV-	3	1.0	1.1	T	1.2	T					
DV-	4	1.0	1.1	T	1.2	T					
DV-	5				1.2	T					
RPTG-	1	1.0	1.1	C	1.2	T					
RPTG-	2	1.0			1.2	T					
RPTG-	3	1.0			1.2	T					
RPTG-	4	1.0	1.1	T	1.2	C					
RPTG-	5	1.0			1.2	TP					
RPTG-	6	1.0	1.1	T	1.2	T					
RPTG-	7	1.0	1.1	T	1.2	C					
RPTG-	8	1.0	1.1	C	1.2	C					
RPTG-	9	1.0	1.1	T	1.2	C					
RPTG-	9a		1.1	T	1.2	T					
RPTG-	9b		1.1	T	1.2	C					
RPTG-	10	1.0			1.2	C					
RPTG-	11	1.0			1.2	C					
RPTG-	12	1.0			1.2	T					
RPTG-	13	1.0			1.2	C					
RPTG-	14	1.0			1.2	T					
RPTG-	15	1.0			1.2	T					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.Ø	1.1	Type	1.2	Type	_._	_._	_._	_._	_._
					Revision Level						
IODVR-	1	1.Ø			1.2	C					
IODVR-	2	1.Ø			1.2	C					
IODVR-	3	1.Ø			1.2	C					
IODVR-	4	1.Ø			1.2	T					
IODVR-	5	1.Ø			1.2	C					
IODVR-	6	1.Ø			1.2	C					
IODVR-	7	1.Ø			1.2	T					
IODVR-	8	1.Ø			1.2	C					
IODVR-	9	1.Ø			1.2	C					
IODVR-	1Ø	1.Ø			1.2	C					
IODVR-	11	1.Ø			1.2	C					
IODVR-	12	1.Ø			1.2	C					
IODVR-	13	1.Ø			1.2	C					
IODVR-	14	1.Ø			1.2	C					
IODVR-	15	1.Ø			1.2	C					
IODVR-	16	1.Ø			1.2	C					
IODVR-	17	1.Ø			1.2	C					
IODVR-	18	1.Ø			1.2	C					
IODVR-	19	1.Ø			1.2	C					
IODVR-	2Ø	1.Ø									
IODVR-	21	1.Ø	1.1	T	1.2	C					
IODVR-	22	1.Ø			1.2	C					
IODVR-	23	1.Ø			1.2	C					
IODVR-	24	1.Ø			1.2	C					
MP-	1	1.Ø			1.2	C					
MP-	2	1.Ø			1.2	C					
MP-	3	1.Ø			1.2	C					
MP-	4	1.Ø	1.1	T	1.2	C					
MP-	5	1.Ø	1.1	T	1.2	C					
MP-	6	1.Ø			1.2	C					
MP-	7	1.Ø									
STLOG-	1	1.Ø	1.1	T							
STLOG-	2	1.Ø	1.1	T	1.2	T					
STLOG-	3	1.Ø	1.1	T	1.2	C					
STLOG-	4		1.1	T	1.2	C					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type (Revision Level)	_._	_._	_._	_._	_._
DEBUG-	1	1.0			1.2	C					
DEBUG-	2	1.0			1.2	C					
DEBUG-	3	1.0			1.2	C					
DEBUG-	4	1.0			1.2	C					
DEBUG-	5	1.0			1.2	C					
DEBUG-	6	1.0			1.2	C					
DEBUG-	7	1.0									
DEBUG-	8	1.0			1.2	C					
DEBUG-	9	1.0			1.2	T					
DEBUG-	10	1.0			1.2	C					
DEBUG-	11	1.0									
DEBUG-	12	1.0			1.2	T					
DEBUG-	13	1.0			1.2	T					
DEBUG-	14	1.0			1.2	C					
DEBUG-	15	1.0			1.2	C					
DEBUG-	16	1.0			1.2	T					
DEBUG-	17	1.0			1.2	C					
DEBUG-	18	1.0			1.2	C					
DEBUG-	19	1.0			1.2	C					
DEBUG-	20	1.0			1.2	C					
DEBUG-	21	1.0			1.2	T					
DEBUG-	22	1.0			1.2	C					
DEBUG-	23	1.0			1.2	T					
DEBUG-	24	1.0			1.2	C					
DEBUG-	25	1.0			1.2	C					
DEBUG-	26	1.0			1.2	T					
DEBUG-	27	1.0			1.2	C					
DEBUG-	28				1.2	C					
DEBUG-	29				1.2	C					
FLIB-	1	1.0	1.1	T	1.2	C					
FLIB-	2	1.0	1.1	T	1.2	T					
FLIB-	3	1.0			1.2	C					
FLIB-	4	1.0	1.1	T	1.2	C					
FLIB-	5	1.0	1.1	T	1.2	C					
FLIB-	6	1.0			1.2	C					
FLIB-	7	1.0	1.1	T	1.2	C					
FLIB-	8	1.0	1.1	C	1.2	C					
FLIB-	9	1.0	1.1	T	1.2	C					
FLIB-	10	1.0	1.1	T	1.2	T					
FLIB-	11	1.0	1.1	T							
FLIB-	12	1.0	1.1	T	1.2	C					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type	_._	_._	_._	_._	_._
A-	1	1.0									
A-	2	1.0			1.2	C					
A-	3	1.0	1.1	T	1.2	T					
A-	4	1.0	1.1	T	1.2	T					
A-	5	1.0	1.1	T	1.2	C					
A-	6	1.0	1.1	T	1.2	C					
A-	7	1.0	1.1	T	1.2	T					
A-	8	1.0	1.1	T	1.2	C					
A-	9	1.0	1.1	T	1.2	C					
A-	10	1.0	1.1	T	1.2	T					
A-	11	1.0	1.1	T	1.2	T					
A-	12	1.0	1.1	T	1.2	T					
A-	13	1.0	1.1	T	1.2	T					
A-	14	1.0									
ackdelay-	1	1.0	1.1	T	1.2	T					
arblose-	1	1.0	1.1	TP	1.2	C					
arblose-	2	1.0	1.1	TP	1.2	C					
arbmode-	1	1.0	1.1	T	1.2	T					
arbwin-	1	1.0	1.1	TP	1.2	C					
arbwin-	2	1.0	omitted		1.2	C					
arb1-	1	1.0	1.1	T	1.2	C					
arb2-	1	1.0	1.1	T	1.2	C					
arb_or_resel-1			1.1	T	1.2	TP					
autosense-	1	1.0	1.1	T	1.2	T					
awin_res-	1		1.1	T	1.2	C					
bcu-1	1	1.0	1.1	T	1.2	C					
bfreearm-	1	1.0	1.1	T	1.2	C					
bfreeck-	1	1.0	1.1	T	1.2	C					
blk_size-	1	1.0	1.1	T	1.2	C					
busrel-	1	1.0	1.1	T	1.2	C					
busywait-	1	1.0	1.1	T	1.2	T					
bus_logen-	1	1.0	1.1	T	1.2	C					
bytcmp-	1	1.0	1.1	TP	1.2	C					
bytrd-	1	1.0	1.1	TP	1.2	C					
btrwrt-	1	1.0	1.1	TP	1.2	C					
ccs_modsel-1			1.1	T	1.2	C					
ccs_modsens-1			1.1	T	1.2	C					
cdb__1-	1	1.0	1.1	T	1.2	C					
cdb__1-	2	1.0	1.1	T							
cdb__2-	1	1.0	1.1	T	1.2	C					
cdb__2-	2				1.2	C					
cdb__3-	1	1.0	1.1	T	1.2	C					
cdb__3-	2				1.2	C					
chk_user_limi-1		1.0			1.2	C					
chk_user_stri-1		1.0			1.2	C					
cmd_tail_bol-1		1.0	1.1	T	1.2	C					
cmd_tail_stri-1		1.0	1.1	T	1.2	C					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type	Revision Level _._	_._	_._	_._	_._
cntlbyte-	1	1.0	1.1	TP	1.2	C					
comp-	1	1.0			1.2	C					
comp-	2	1.0			1.2	C					
compwr-	1	1.0			1.2	C					
copy-	1	1.0			1.2	C					
copyver-	1	1.0			1.2	C					
copyver-	2	1.0			1.2	C					
copy_user_str-	1	1.0			1.2	C					
datain_-	1	1.0	1.1	T	1.2	C					
datain_-	2	1.0	1.1	C	1.2	C					
datain4-	1	1.0	1.1	T	1.2	C					
datain4-	2	1.0	1.1	C	1.2	C					
dataout_-	1	1.0	1.1	T	1.2	C					
dataout_-	2	1.0	1.1	C	1.2	C					
debug-	1	1.0			1.2	C					
delayms-	1	1.0			1.2	C					
delays-	1	1.0			1.2	C					
delta_time-	1		1.1	T	1.2	T					
delta_time-	2		1.1	T	1.2	T					
dispbuf-	1		1.1	T	1.2	C					
dmarst-	1	1.0	1.1	T	1.2	C					
dmaset-	1	1.0	1.1	C	1.2	C					
dmaset_va-	1	1.0	1.1	T	1.2	C					
dmaset_vblk-	1	1.0	1.1	TP	1.2	C					
eea-	1	1.0			1.2	T					
erase-	1	1.0			1.2	C					
errdelay-	1		1.1	T							
error_ok-	1				1.2	T					
eseom-	1	1.0			1.2	C					
esfm-	1	1.0			1.2	TP					
esili-	1	1.0			1.2	C					
esinfob-	1	1.0			1.2	C					
eskey-	1	1.0			1.2	C					
eskeynot-	1	1.0			1.2	C					
esvalid-	1	1.0			1.2	C					
exp_status-	1	1.0			1.2	T					
fail-	1	1.0	1.1	T	1.2	C					
fillbcb-	1	1.0	1.1	T	1.2	C					
fillbcb-	2	1.0	omitted								
fillbcw-	1	1.0	1.1	T	1.2	C					
fillbcw-	2	1.0	omitted								
fillbyte-	1	1.0	1.1	T	1.2	C					
filld-	1	1.0	1.1	T	1.2	C					
filli-	1	1.0	1.1	T	1.2	C					
fillk-	1	1.0	1.1	T	1.2	C					
fillk-	2	1.0			omitted						
fillpr-	1	1.0	1.1	T	1.2	C					
fixed-	1	1.0			1.2	C					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type	_._	_._	_._	_._	_._
					Revision Level						
forcbusy-	1	1.0			1.2	C					
forcbusy-	2	1.0	1.1	T							
forceattn-	1	1.0	1.1	T	1.2	C					
forcperr-	1	1.0	1.1	T							
format-	1	1.0			1.2	C					
get_byte-	1	1.0			1.2	C					
get_f_stats-	1	1.0			1.2	C					
get_f_status-	1	1.0			1.2	C					
get_g_stats-	1	1.0			1.2	TP					
get_infoin-	1	1.0			1.2	C					
get_phase-	1	1.0			1.2	C					
get_user_int-	1	1.0			1.2	C					
get_user_long-	1		1.1	T	1.2	TP					
group-	1	1.0	1.1	T	1.2	C					
iea-	1	1.0			1.2	T					
iid-	1	1.0	1.1	T	1.2	C					
inc_blk-	1	1.0			1.2	C					
inc_len-	1	1.0			1.2	C					
inquiry-	1	1.0			1.2	C					
ioto-	1	1.0	1.1	T	1.2	C					
io6-	1	1.0			1.2	C					
io10-	1	1.0			1.2	C					
io10-	2	1.0			1.2	C					
io12-	1	1.0			1.2	C					
io12-	2	1.0			1.2	C					
ldunlds-	1	1.0			1.2	C					
line_mode-	1		1.1	T	1.2	C					
loadbuf-	1	1.0	1.1	T	1.2	TP					
loadbuf-	2	1.0									
logc-	1	1.0	1.1	T	1.2	C					
logp-	1	1.0			1.2	C					
lun-	1	1.0			1.2	T					
modesen-	1	1.0			1.2	C					
mode_sel-	1	1.0			1.2	C					
mode_sel-	2		1.1	T	1.2	C					
modsels-	1	1.0	1.1	T	1.2	C					
modsens-	1	1.0			1.2	C					
msgin-	1	1.0			1.2	C					
msgout-	1	1.0	1.1	T	1.2	C					
msgout_atnf-	1		1.1	T	1.2	C					
opcnt-	1	1.0			1.2	C					
overbcb-	1	1.0	1.1	T	1.2	C					
overbcb-	2	1.0	omitted								
overbcdw-	1				1.2	T					
overbcdw-	2				1.2	T					
overbcw-	1	1.0	1.1	T	1.2	C					
overbcw-	2	1.0									

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type	_._	_._	_._	_._	_._
					Revision Level						
page-	1	1.0			1.2	C					
paragph-	1	1.0	1.1	T	1.2	C					
parity-	1	1.0			1.2	T					
pass-	1	1.0			1.2	C					
pause-	1	1.0			1.2	C					
prevmedr-	1	1.0			1.2	C					
prevmeds-	1	1.0			1.2	C					
put_byte-	1	1.0			1.2	C					
random_blk-	1	1.0			1.2	C					
random_len-	1	1.0			1.2	C					
rbufbyte-	1	1.0			1.2	C					
rbufword-	1	1.0			1.2	C					
rdblklts-	1	1.0			1.2	C					
rd_buffer-	1		1.1	T	1.2	C					
rd_buffer-	2		1.1	T	1.2	C					
rd_defect-	1		1.1	T	1.2	C					
rd_defect-	2		1.1	T	1.2	C					
readcap-	1	1.0			1.2	C					
readcap-	2	1.0			1.2	C					
readr-	1	1.0			1.2	C					
readrev-	1	1.0			1.2	C					
readrl-	1	1.0			1.2	C					
readrl0-	1	1.0			1.2	C					
readrl0-	2	1.0			1.2	C					
readrl0_blk-	1	1.0			1.2	C					
readrl0_blk-	2	1.0			1.2	C					
readr_blk-	1	1.0			1.2	C					
reads-	1	1.0			1.2	C					
readsl-	1	1.0			1.2	C					
reasgnb-	1	1.0			1.2	C					
recbufds-	1	1.0			1.2	C					
recvdiag-	1	1.0			1.2	C					
releaser-	1	1.0			1.2	C					
releases-	1	1.0			1.2	C					
resel-	1	1.0	1.1	T	1.2	C					
resel_wt-	1		1.1	T	1.2	C					
reserves-	1	1.0			1.2	C					
reservr-	1	1.0			1.2	C					
reservr-	2	1.0			omitted						
reset-	1	1.0			1.2	C					
rewind-	1	1.0			1.2	C					
rezero-	1	1.0			1.2	C					
rptbuf-	1	1.0	1.1	T	1.2	C					
rptsen-	1	1.0			1.2	C					
rptstats-	1	1.0			1.2	C					
rpttmr-	1	1.0			1.2	C					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type	Revision Level _._	_._	_._	_._	_._
savebuf-	1	1.0	1.1	T	1.2	C					
savebuf-	2	1.0	omitted								
sbb-	1	1.0			1.2	C					
sbw-	1	1.0			1.2	C					
searchde-	1	1.0			1.2	T					
searchde-	2	1.0			1.2	C					
searchdh-	1	1.0			1.2	T					
searchdh-	2	1.0			1.2	C					
searchdl-	1	1.0			1.2	T					
searchdl-	2	1.0			1.2	C					
seek-	1	1.0			1.2	C					
seekl-	1	1.0			1.2	T					
seekl0-	1	1.0			1.2	C					
seekl0-	2	1.0			1.2	C					
selmode-	1	1.0	1.1	T	1.2	T					
sel1-	1	1.0			1.2	C					
sel2-	1	1.0			1.2	TP					
sel3-	1	1.0	1.1	T							
sel4-	1	1.0			1.2	C					
senddiag-	1	1.0			1.2	C					
sense-	1	1.0			1.2	T					
serclass-	1	1.0			1.2	C					
serrcd-	1	1.0			1.2	C					
setbuf-	1	1.0			1.2	C					
setfill_buf-	1	1.0			1.2	C					
setlimts-	1	1.0			1.2	C					
setlimts-	2	1.0			1.2	C					
set_blk-	1	1.0			1.2	TP					
set_er_limits-	1	1.0			1.2	C					
set_len-	1	1.0			1.2	C					
sladdr-	1	1.0	1.1	T	1.2	C					
space-	1	1.0			1.2	C					
state_data-	1		1.1	T	1.2	C					
statin-	1	1.0			1.2	C					
statsen-	1	1.0			1.2	T					
stats_reset-	1	1.0	1.1	T	1.2	T					
stats_window-	1	1.0			1.2	T					
stat_mask-	1	1.0			1.2	T					
strstop-	1	1.0			1.2	C					
subpar-	1	1.0			1.2	C					
summary-	1	1.0			1.2	C					
svalid-	1	1.0			1.2	C					
svu-	1	1.0			1.2	C					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	1.2	Type	_._	_._	_._	_._	_._
					Revision Level						
test-	1	1.0			1.2	C					
testur-	1	1.0			1.2	C					
tid-	1	1.0			1.2	T					
tksel-	1	1.0			1.2	C					
tmrlmt-	1	1.0			1.2	C					
tmrset-	1	1.0			1.2	C					
tmrstart-	1	1.0			1.2	C					
tmrstop-	1	1.0			1.2	C					
tmrvalue-	1	1.0			1.2	C					
ucinc-	1	1.0			1.2	C					
ucname-	1	1.0			1.2	C					
ucrst-	1	1.0			1.2	C					
ureset-	1	1.0			1.2	C					
user_input-	1	1.0	1.1	T	1.2	T					
verifys-	1	1.0			1.2	C					
verify10-	1	1.0			1.2	C					
verify10-	2	1.0			1.2	C					
writer-	1	1.0			1.2	C					
writerl-	1	1.0			1.2	T					
writer10-	1	1.0			1.2	C					
writer10-	2	1.0			1.2	C					
writer10_blk-	1	1.0			1.2	C					
writer10_blk-	2	1.0			1.2	C					
writer_blk-	1	1.0			1.2	C					
writes-	1	1.0			1.2	C					
writesl-	1	1.0			1.2	C					
wrtfilm-	1	1.0			1.2	C					
wrtvfy10-	1	1.0			1.2	C					
wrtvfy10-	2	1.0			1.2	C					
wrt_buffer-	1		1.1	T	1.2	C					
wrt_buffer-	2		1.1	T	1.2	C					
xfermode-	1	1.0	1.1	T	1.2	C					
xfermode-	2	1.0	1.1	T	1.2	C					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

					Revision Level						
Section	Page Number	1.0	1.1	Type	1.2	Type	_._	_._	_._	_._	_._
B-	1	1.0									
B-	2	1.0	1.1	T							
B-	3	1.0	1.1	T							
B-	4	1.0									
B-	5	1.0			1.2	C					
B-	6	1.0			1.2	C					
B-	7	1.0	1.1	C	1.2	T					
B-	8	1.0	1.1	T	1.2	T					
B-	9	1.0	1.1	T	1.2	T					
B-	10	1.0	1.1	T	1.2	T					
B-	11	1.0	1.1	T	1.2	T					
B-	12	1.0	1.1	T	1.2	T					
B-	13	1.0	1.1	T	1.2	T					
B-	14	1.0	1.1	T	1.2	T					
B-	15	1.0	1.1	T	1.2	T					
B-	16	1.0	1.1	T	1.2	T					
B-	17	1.0	1.1	T	1.2	T					
B-	18	1.0	1.1	T	1.2	T					
B-	19	1.0	1.1	T	1.2	T					
B-	20	1.0	1.1	T	1.2	T					
B-	21	1.0	1.1	T	1.2	T					
B-	22	1.0	1.1	T	1.2	T					
B-	23	1.0	1.1	T	1.2	T					
B-	24	1.0	1.1	T							
B-	25	1.0	1.1	T	1.2	T					
B-	26	1.0	1.1	T	1.2	T					
B-	27	1.0	1.1	T	1.2	T					
B-	28	1.0	1.1	T	1.2	T					
B-	29	1.0	1.1	T	1.2	T					
B-	30	1.0	1.1	T	1.2	T					
B-	31	1.0	1.1	T	1.2	T					
B-	32	1.0	1.1	T	1.2	T					
B-	33	1.0	1.1	T	1.2	T					
B-	34	1.0	1.1	T	1.2	T					
B-	35	1.0	1.1	T	1.2	T					
B-	36	1.0	1.1	T	1.2	T					
B-	37	1.0	1.1	T	1.2	T					
B-	38	1.0	1.1	T	1.2	T					
B-	39	1.0	1.1	T	1.2	T					
B-	40	1.0	1.1	T	1.2	T					
B-	41	1.0	1.1	T	1.2	T					
B-	42	1.0	1.1	T	1.2	T					
B-	43	1.0	1.1	T	1.2	T					
B-	44	1.0	1.1	T	1.2	T					
B-	45	1.0	1.1	T	1.2	T					
B-	46	1.0	1.1	T	1.2	T					
B-	47	1.0	1.1	T	1.2	T					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)

Section	Page Number	1.0	1.1	Type	Revision Level 1.2	Type	_._	_._	_._	_._	_._
B-	48	1.0	1.1	T	1.2	T					
B-	49	1.0	1.1	T	1.2	T					
B-	50	1.0	1.1	T	1.2	T					
B-	51	1.0	1.1	T	1.2	T					
B-	52	1.0	1.1	T	1.2	T					
B-	53	1.0	1.1	T	1.2	T					
C-	1	1.0			1.2	C					
C-	2	1.0	1.1	T	1.2	T					
C-	3	1.0	1.1	T	1.2	T					
C-	4	1.0	1.1	T	1.2	T					
C-	5	1.0	1.1	T	1.2	T					
C-	6	1.0	1.1	T	1.2	T					
C-	7	1.0	1.1	T	1.2	T					
C-	8	1.0	1.1	T	1.2	T					
C-	9	1.0	1.1	T	1.2	T					
C-	10	1.0	1.1	T	1.2	T					
C-	11	1.0	1.1	T	1.2	T					
C-	12	1.0	1.1	T	1.2	T					
C-	13		1.1		1.2	T					
C-	14				1.2	T					

Type (T=Technical, TP=Typing Errors, C=Cosmetic)