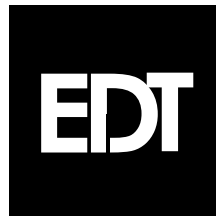


S53B1

SBus to MIL-STD 1553B Interface

USER'S GUIDE

008-00297-07



The information in this document is subject to change without notice and does not represent a commitment on the part of Engineering Design Team, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, without the express written agreement of Engineering Design Team, Inc.

© Copyright Engineering Design Team, Inc. 1992–1997. All rights reserved.

Refer questions or problems with this manual or the hardware or software described herein to:

Engineering Design Team, Inc.
1100 NW Compton Drive, Suite 306
Beaverton, Oregon 97006

Phone: (503) 690-1234
Fax: (503) 690-1243
E-mail: info@edt.com
Web: www.edt.com

Sun, SunOS, SBus, SPARC, and SPARCstation are trademarks of Sun Microsystems, Incorporated.

UNIX and OPEN LOOK are registered trademarks of UNIX System Laboratories, Inc.

X Window System is a product of the Massachusetts Institute of Technology.

EDT and Engineering Design Team are trademarks of Engineering Design Team, Inc.

Contents

Overview	1
Installation.....	2
Installing the Hardware	2
Installing the Software.....	2
Using SunOS Version 4.1	3
Using System V Release 4 (Solaris 2.0)	4
Included Files	4
About the MIL-STD 1553 Bus	6
Bus Elements	6
Message Types	6
Command Word	8
Status Word	9
Mode Codes.....	10
Connecting to a 1553 Bus	11
Connectors.....	11
External Bus Coupling	11
Writing Applications.....	15
Example Applications	15
bctest	15
rttest.....	16
s53btest	16
bm	17
setdebug	17
xmt1553	18
rcv1553	18
testdriver	18
mem_s53bi.....	19
Opening The S53B1 Driver.....	19
Reading and Writing Data To and From the S53B1	19
In Bus Controller Mode	19
In Remote Terminal Mode.....	20
In Bus Monitor Mode.....	20
Read and Write Data Structures.....	22
Using the Data Structures as a Bus Controller.....	23
Using the Data Structures as a Remote Terminal	25
Sending Mode Codes	27
Receiving Mode Codes	28
Specifying Error Insertion and Intermessage Gap for System Tests	29
Scheduling bc_auto Structures	33
Shared Memory Management.....	34
Using the Commander	37
Registers.....	42
Data Registers.....	43
Status Registers	44
Reset Registers	44

Command Registers.....45
Board Control Registers45
RAM.....46
EEPROM.....46
Specifications47
 MIL-STD 155347
 Software47
 SBus Compliance.....47
 Power47
 Environmental.....47
 Physical47
Glossary48
References50

Figures

Direct Coupling	12
Transformer Coupling	13
S53B1 Memory	35
The Commander as Bus Controller	38
The Commander as Remote Terminal	38
Setting the Masks	39
The Commander as Bus Monitor, Summarizing	40
The Commander as Bus Monitor, Showing Data	41
SBus Addresses of S53B1 Hardware Registers	42
SPARC Addresses of S53B1 Hardware Registers	43

Tables

Mode Codes	10
Data Bus and Coupling Requirements	14
Mode Code Responses	28
Error Codes	30
Status Register Bit Definitions.....	44
Reset Register Bit Definitions.....	44
Command Register Bit Definitions	45
Board Control Interrupt Register Bit Definitions	46
Board Control SBus Interrupt Register Bit Definitions	46
Board Control SPARC Run Register Bit Definitions	46
Board Control Reset Register Bit Definitions.....	46

Overview

MIL-STD-1553B is a 1 Mb per second serial bus interface used where reliability in extreme environments is essential, such as aircraft or satellites. It is typically used to configure a variety of sensors or subsystems and to report their status to a central controller.

The S53B1 SBus to MIL-STD-1553B interface allows you to connect a Sun workstation to a 1553B bus, or to use your Sun workstation to emulate an entire 1553B bus system of 32 devices or fewer, including a bus controller, several remote terminals, and a bus monitor. The S53B1 has two channels for dual redundancy and an embedded SPARC microprocessor and 128 KB of memory, which applications can access, to ensure optimum performance. It supports the full 1553B set of standard commands and subcommands.

NOTE: Engineering Design Team can customize an S53B1 to detect any standard command or subcommand as illegal if your application requires it.

The S53B1 includes a SunOS-loadable, configurable device driver and a wide variety of example applications that can be customized for many different purposes. A graphical OPEN LOOK-based bus analyzer program is also included, making the S53B1 an ideal development and testing tool for 1553B systems.

This document describes how to install the S53B1 bus interface and write applications for it. It is divided into the following sections:

Installation	describes how to install the S53B1 module and its related software.
About the MIL-STD-1553 Bus	provides an overview of the 1553B bus functionality.
Connecting to a 1553 Bus	describes the connectors and coupling required for various configurations.
Writing Applications	explains how the example programs provided can get you started programming for the S53B1 driver.
Using the Commander	explains how to use the graphical interface provided to control the S53B1.
Registers	describes the S53B1 hardware registers.
Glossary	defines the terms and acronyms used in this document.
Specifications	lists the specifications.
References	refers to other documents that may prove useful to you when writing applications for the S53B1.

Installation

Installing the S53B1 differential serial bus interface is a two-step process. First you must physically install the board inside the host computer. Then you must install the software driver so that applications can access the S53B1. Hardware installation is described in the following section. Software installation is described in the section after.

Installing the Hardware

The S53B1 is a single-slot SBus board. To install it, refer to your SBus host computer documentation for complete information on installing an SBus board. For example, for the SUN SPARCstation models 4/60 and 4/65, see the *SPARCstation Installation Guide*, Sun part number 800-4036-10, Appendix A: *Installing SBus Boards and Cards*.

Use the following procedure to install the S53B1:

CAUTION

Both the S53B1 and your SBus host computer contain static-sensitive components. Install the S53B1 at a static-free work area. If a static-free work area is not available, take the following precautions to reduce the risk of component damage:

1. Remove from the immediate area all materials that can generate or hold a static charge.
 2. Discharge yourself by touching both hands to a metal portion of the host computer's chassis before you open the host computer or open the S53B1 static-shielded bag.
-
-

1. Unpack the S53B1 from the shipping packaging. Do not remove the S53B1 from the static shielding bag until all you remove all other packaging materials from the area and established a static-free work area
2. Install the S53B1 in the SBus host, following the directions provided with the SBus host. The S53B1 can be installed in any slot.
3. The small switch located between the two 1553 connectors switches the S53B1 from a direct-coupled bus connection, labeled DIRECT to a transformer-coupled bus connection, labeled XFMR. Set the bus coupling switch as required for your application: the difference is explained thoroughly on page 11. Most systems use transformer coupling, which is more reliable. Transformer coupling is required if the stub is longer than one foot.
4. Connect the board to the bus as your application requires. See page 11 for complete instructions.

To remove the S53B1, reverse the installation procedure.

Installing the Software

The S53B1 can run on a Sun workstation using either SunOS Version 4.1 or Solaris 2.0 (System V Version 4, or SVR 4). The installation procedures differ. Both are given below.

Using SunOS Version 4.1

If you are using SunOS Version 4.1, use the following procedure to install the S53B1 driver:

1. Become root or superuser.
2. Change to the directory in which you wish to install the S53B1 driver.
3. Place the diskette that came with the S53B1 into the diskette drive.
4. The S53B1 driver and related files are included on a diskette in *tar* format. To copy them to your hard disk, enter:

```
tar xvf /dev/rfd0
```

5. The *tar* program extracts a number of files. (The list of files distributed is provided in the section entitled **Included Files**.) The S53B1 diskette contains versions of the S53B1 driver for a variety of Sun platforms and versions of the Sun operating system. The installation program installs the correct driver based on the host platform and operating system version.

```
make install
```

The makefile provided installs and loads the S53B1 driver.

6. During the installation, the following question appears on the display:

```
Automatically load the s53b1 driver during each reboot? [y|n] (y):
```

7. Entering *y* (or simply typing <Return>) causes the S53B1 driver to be loaded whenever you reboot your host computer. If you respond with *n*, you must manually reload the driver after rebooting. To do so, enter:

```
make load
```

During the installation, the following question appears on the display:

```
How many s53b1 devices do you want? (1):
```

You can install up to eight S53B1 boards in your system (numbered 0 -7). Enter the number corresponding to the number of S53B1 boards you have installed in your system. If you simply type <Return>, one driver is installed.

NOTE: If you anticipate installing more than one S53B1 board into your system, install as many S53B1 drivers as you will ultimately require. The extra drivers will do no harm and will be there when you need them, saving you a step.

8. If the S53B1 module has not been installed inside the host computer, or has been installed incorrectly, the following message appears on the display:

```
Can't load this module
```

If you see this message, go back to the section entitled **Installing the Hardware** and reinstall the board.

To unload the S53B1 driver:

1. Change to the directory in which you placed the S53B1 files, if necessary.
2. Become root or superuser.

3. Enter:

```
make unload
```

Using System V Release 4 (Solaris 2.0)

If you are using Sun System V Release 4 (Solaris 2.0), use the following procedure to install the S53B1 driver:

1. Become root or superuser.
2. Change to the directory in which you wish to install the S53B1 driver.
3. Place the diskette that came with the S53B1 into the diskette drive.
4. Invoke the File Manager by entering:

```
filemgr
```

or switch to the File Manager window if it is already running.

5. In the File Manager, pull down the File menu and execute the menu item Check For Floppy.
6. When the Unlabeled Floppy window appears, select Cancel.
7. At the shell prompt, enter:

```
pkgadd -d /vol/dev/aliases/floppy0 EDTs53bi
```

For further details, consult your Solaris 2.0 documentation, or call the Engineering Design Team.

To remove the S53B1 driver:

1. Become root or superuser.
2. Enter:

```
pkgrm EDTs53bi
```

For further details, consult your Solaris 2.0 documentation, or call the Engineering Design Team.

Included Files

The following files are included in your purchase of the S53B1 bus interface for SunOS Version 4.1 (see the *readme* file for a complete, up-to-date listing):

s53b.o.sun4c

The executable S53B1 driver for SunOS 4.1.3 on a Sun 4C architecture such as a SPARCStation 1, 1+, 2, or IPC.

s53b.o.sun4m

The executable S53B1 driver for SunOS 4.1.3 on a Sun 4M architecture such as a SPARCStation 5, 10, 20, LX, Classic, or an Ultra 1 or 2.

s53b

The executable S53B1 driver for Solaris Version 2.x.

s53bi.h

S53B1 driver header file with *ioctl* and register definitions.

s53bi.INSTALL

The installation script used by the makefile.

<code>makefile</code>	The makefile for installing, loading, and unloading the driver, and making the example programs. Used with the SunOS <i>make</i> command to automatically install the driver, load the driver and compile the example applications.
<code>s53bload</code>	An executable used by the makefile, and when the driver boots, to load object code to the embedded SPARC microprocessor.
<code>embedded.image</code>	The object code downloaded to the embedded SPARC microprocessor.
<code>README</code>	An ASCII file containing last-minute information on the S53B1 software.
<code>checks53b.c</code>	A diagnostic program to check the time stamp on the board, the current state of the embedded SPARC, and other possible problems.
<code>cmdword.c</code>	A program to disassemble and assemble command words. This can be useful when monitoring the bus.
<code>commander</code>	An OPEN LOOK graphical user interface for controlling the S53B1.
<code>s53btest.c</code>	Example program showing all functions of the S53B driver, including error insertion and specifying the intermessage gap.
<code>bctest.c</code>	Example program showing bus controller operations.
<code>rttest.c</code>	Example program showing remote terminal operations.
<code>bm.c</code>	Example program showing bus monitor operations.
<code>setdebug.c</code>	Example program setting the S53B1 debug level.
<code>xmt1553.c</code>	Example program to place <i>stdin</i> into the data words of RT receive commands and output them to the 1553B bus.
<code>rcv1553.c</code>	Example program to receive commands from the 1553B bus and copy the data words contained therein to <i>stdout</i> .
<code>s53b_mon.c</code>	Example program showing user-defined bus monitoring.
<code>userbm.c</code>	Example program that invokes <code>s53b_mon.c</code> and shows an example of using user-defined bus monitoring.
<code>rtmodeq.c</code>	Example program showing how to examine the mode codes queued for a remote terminal.
<code>bc_auto_sched.c</code>	Example program illustrating scheduling modes for <code>bc_auto</code> structures.
<code>testdriver.c</code>	Example program illustrating continuous double-buffered <code>bc_auto</code> structure execution. The executable is also included.
<code>mems53bi.c</code>	Example and utility program showing memory management options for <code>bc_auto</code> structures and remote terminals. The executable is also included.
<code>libs53bi.a</code>	General-purpose application support routines.
<code>libs53bi.c</code>	Source code for general-purpose application support routines.

Many of the example programs are described in more detail starting on page 15.

About the MIL-STD 1553 Bus

The MIL-STD 1553 bus, revision B, is a differential serial bus interface used in military equipment. The 1 Mb-per-second bus usually has redundant channels. It has been used in research and development, as well as production systems, to integrate target, weapons and system status. For the complete specification and implementation handbook, see the section entitled **References**.

Bus Elements

Each 1553 bus has at least three elements:

- a bus controller (BC),
- one or more remote terminals (RT), and
- the bus itself (cable, couplers and connectors).

The bus can also have a bus monitor (BM).

A bus can have only one active bus controller at a time. The BC explicitly manages all data transfers on the bus using a command/response protocol. Bus controller responsibility can be transferred from unit to unit using mode codes—a capability referred to as *dynamic bus control*—although some systems explicitly disallow this. The BC initiates a transfer by sending a command word followed by data, if required. The selected RT responds with status and data, if required.

Each remote terminal on the bus has a unique address. The bus controller selects a remote terminal using five bits of the command word. Of the 32 RT addresses, address 31 is reserved for broadcasting a transmission to all RTs. Consequently, no more than 31 RTs are allowed on a 1553 bus. Within the command word, five more bits are reserved for selecting one of 32 subaddresses. Two subaddresses (0 and 31) are reserved to flag a mode code transmission. Each of the remaining 30 subaddresses can contain up to thirty-two 16-bit data words. A remote terminal needs to implement only the subaddresses and data words required for its function.

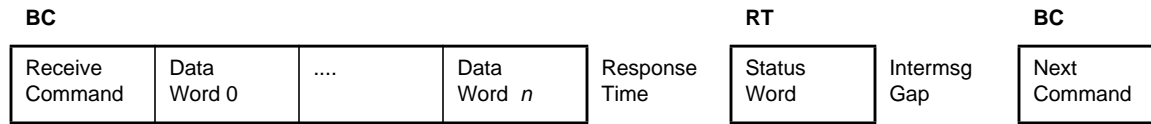
The bus itself is a controlled-impedance differential cable. This cable is terminated at both ends with resistors valued at the characteristic impedance. The remote terminals and bus controller are connected to the bus using either a direct connection or a transformer-coupled connection. The cable for the direct connection, if allowed, must not be longer than one foot. For longer distances, a transformer coupling must be used, and in most applications a transformer connection is required to enhance bus reliability.

A bus monitor can be used to monitor all bus traffic. This information can be used for development or diagnostics.

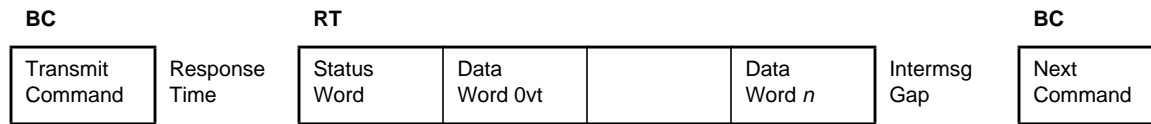
Message Types

The 1553 bus uses ten message types, whose format is shown below. Each box represents 20 s on the bus: 3 s for synchronization and word identification (whether the word represents data or a command), 16 s to transmit the command or data, and 1 s for parity. Response time is 4–12 s, and intermessage time is any amount of time longer than 4 s.

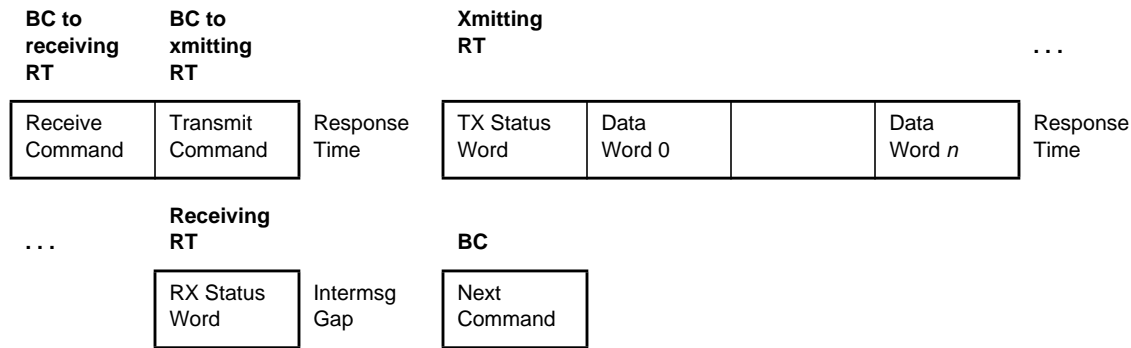
BC to RT Transfer



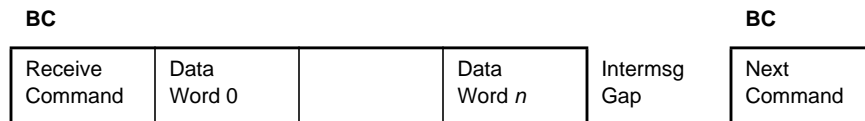
RT to BC Transfer



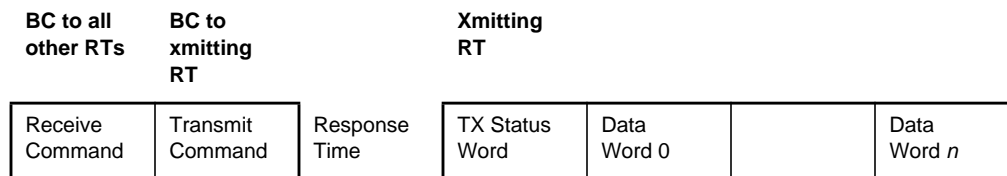
RT to RT Transfer



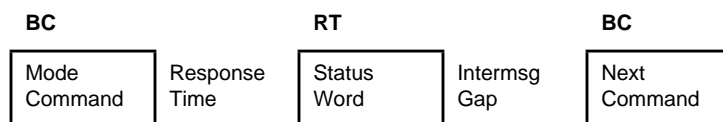
BC to All RTs Broadcast



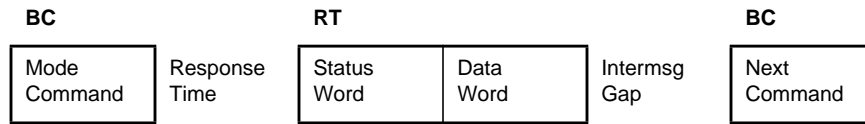
RT to All Other RTs Broadcast



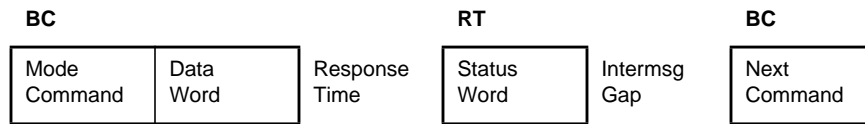
Mode Command—No Data Word



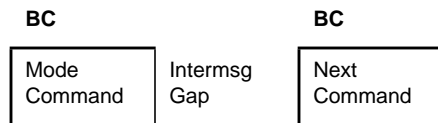
Mode Command—Data Word RT Transmit



Mode Command—Data Word RT Receive



Broadcast Mode Command—No Data Word



Broadcast Mode Command—Data Word RT Receive



Command Word

The command word contains 16 active bits, three sync bits and a parity bit. The sync bits tag the word as a command word. The bits are defined below in the reverse order transmitted.

D15	D14	D13	D12	D11	D10	D9	D8
RTADD4	RTADD3	RTADD2	RTADD1	RTADD0	T/R	SADD4	SADD3
D7	D6	D5	D4	D3	D2	D1	D0
SADD2	SADD1	SADD0	WCNT4R	WCNT3R	WCNT2R	WCNT1R	WCNT0R

RTADD[4-0] Remote Terminal Address. A value of 11111 (31) indicates a broadcast command

T/R Transmit/Receive bit. A value of 0 indicates addressed RT must receive.

SADD[4-0] Subaddress/Mode. Values 00001 through 11110 indicate which subaddress of the addressed RT must transmit or receive. 00000 or 11111 indicate that the command is a mode command and the WCNT field is the mode code.

WCNT[4-0] Word Count/ Mode Code. For subaddresses 00001 through 11110, WCNT indicates the word count—that is, the data transfer size. Values of 1 through 31 indicate 1 word through 31 words, respectively. A value of 0 indicates 32 words.

When the subaddress field is 0 or 31, the WCNT bits specify the mode code.

Status Word

The status word contains 16 active bits, three sync bits and a parity bit. The sync bits tag the word as a status word. The bits are defined below in the reverse order transmitted.

D15	D14	D13	D12	D11	D10	D9	D8
RTADD4T	RTADD3D	RTADD2S	RTADD1B	RTADD0B	MERR	INSTR	SRQ
D7	D6	D5	D4	D3	D2	D1	D0
RSV2	RSV1	RSV0	BCRCD	BUSY	SFLAG	DBACP	TF

RTADD[4-0]	Remote Terminal Address indicates address of RT returning status.
MERR	Message Error. Set to a one if the preceding command or data words fail validity tests.
INSTR	Instrumentation Bit. Some systems use this bit to distinguish a command word from a status word unambiguously. In such systems, the corresponding bit in the command word would always be set to zero, restricting such a system to 15 subaddresses per RT. The S53B1 does not support this feature.
SRQ	Service Request. When set to one, indicates the RT requires application-dependent service.
RSV[2-0]	Reserved. Set to zero.
BCRCD	Broadcast Command Received. Set to one when the previous command was a broadcast command.
BUSY	Busy Bit. Set to one when RT cannot move the data requested by the BC.
SFLAG	Subsystem Flag. Set to one when the RT has detected an internal fault.
DBACP	Dynamic Bus Control Acceptance. Set to one if the RT has received a Dynamic Bus Control mode code and is prepared to assume BC responsibilities.
TF	Terminal Flag. Set to one to indicate a fault in the RT.

Mode Codes

Mode codes allow the BC to control the mode and operation of the bus and obtain diagnostic information.

Mode Code	Function	T/R	Data Word	Broadcast Allowed	S53B1 Response (see page 28 for details)
00000	Dynamic Bus Control	1	None	No	Possible interrupt
00001	Synchronize	1	None	Yes	Possible interrupt
00010	Transmit Status Word	1	None	No	Transmit contents of status register
00011	Initiate Self Test	1	None	Yes	None: wrap-around test executed for every message
00100	Transmitter Shutdown	1	None	Yes	Possible interrupt
00101	Override Transmitter Shutdown	1	None	Yes	Possible interrupt
00110	Inhibit Terminal Flag (TF)Bit	1	None	Yes	Possible interrupt
00111	Override Inhibit TF Bit	1	None	Yes	Possible interrupt
01000	Reset Remote Terminal	1	None	Yes	Possible interrupt
01001– 01111	Reserved				
10000	Transmit Vector Word	1	1	No	Transmit contents of vector word register
10001	Synchronize with Data Word	0	1	Yes	Returns data word
10010	Transmit Last Command	1	1	No	Transmit contents of command/status word register
10011	Transmit Built-in Test Word	1	1	No	Transmit contents of built-in test error register
10100	Selected Transmitter Shutdown	0	1	Yes	Possible interrupt
10101	Override Selected Transmitter Shutdown	0	1	Yes	Possible interrupt
10110– 11111	Reserved				

Table 1. Mode Codes

Connecting to a 1553 Bus

The MIL-STD 1553 specification (see **References**, page 50) covers the physical design of the bus in detail. This document discusses a typical bus, a 78- twinax cable—a 100% shielded cable with two signal wires—terminated at both ends with 78- resistors. At each tap point for a subsystem, a transformer and another twinax cable—the stub—is connected to the subsystem..

Resistance can vary between 70–85 , but the resistors terminating both ends must match the resistance of the cable.

The transformer is chosen so that the subsystem presents very little load on the bus at the frequency of bus operation.

Connectors

The primary and secondary bus connectors are three-lug concentric triaxial type, part number AMP 222153-3. A good source for small quantities of mating connectors is Trompeter Electronics of Westlake Village, CA, (818) 707-2020. The exact mating part number depends on your cable. Typical cable assemblies are Trompeter PTWY series.

To determine the connector required to connect the stub to the bus, consult your system configuration.

If you are using the S53B1 to emulate an entire 1553 system, including the bus controller, remote terminals, and bus monitors, place a 35–43 termination resistor on both outputs. For example, a 40 resistor is available from Trompeter Electronics. The part number is TNG-1-1-40. Other suitable parts can be found in the TNG-1-1-*n* series, where *n* is the resistance.

If you wish merely to connect the S53B1 to one other 1553 device, use two Trompeter TNG-2-*n* resistors, where *n* is the resistance. Make sure that the resistance of the terminations matches the resistance of the cable. For example, use Trompeter part number is TNG-2-78 and a 78 cable.

External Bus Coupling

The small switch located between the two 1553 connectors switches the S53B1 from a direct-coupled bus connection, labeled DIRECT to a transformer-coupled bus connection, labeled XFMR. A good source of couplers and terminators is Technitrol of Philadelphia, PA, (215) 426-9105.

Use a direct-coupled connection if the length of the stub (the length from the wire to the board) is one foot or less. Direct-coupled connections can be smaller, lighter, cheaper, and perhaps simpler than transformer-coupled connections, although this is not always the case. However, they are significantly less robust, as a short circuit in the cable or device can cause the bus to fail. For this reason, most applications require transformer coupling.

Direct coupling is shown in the figure below.

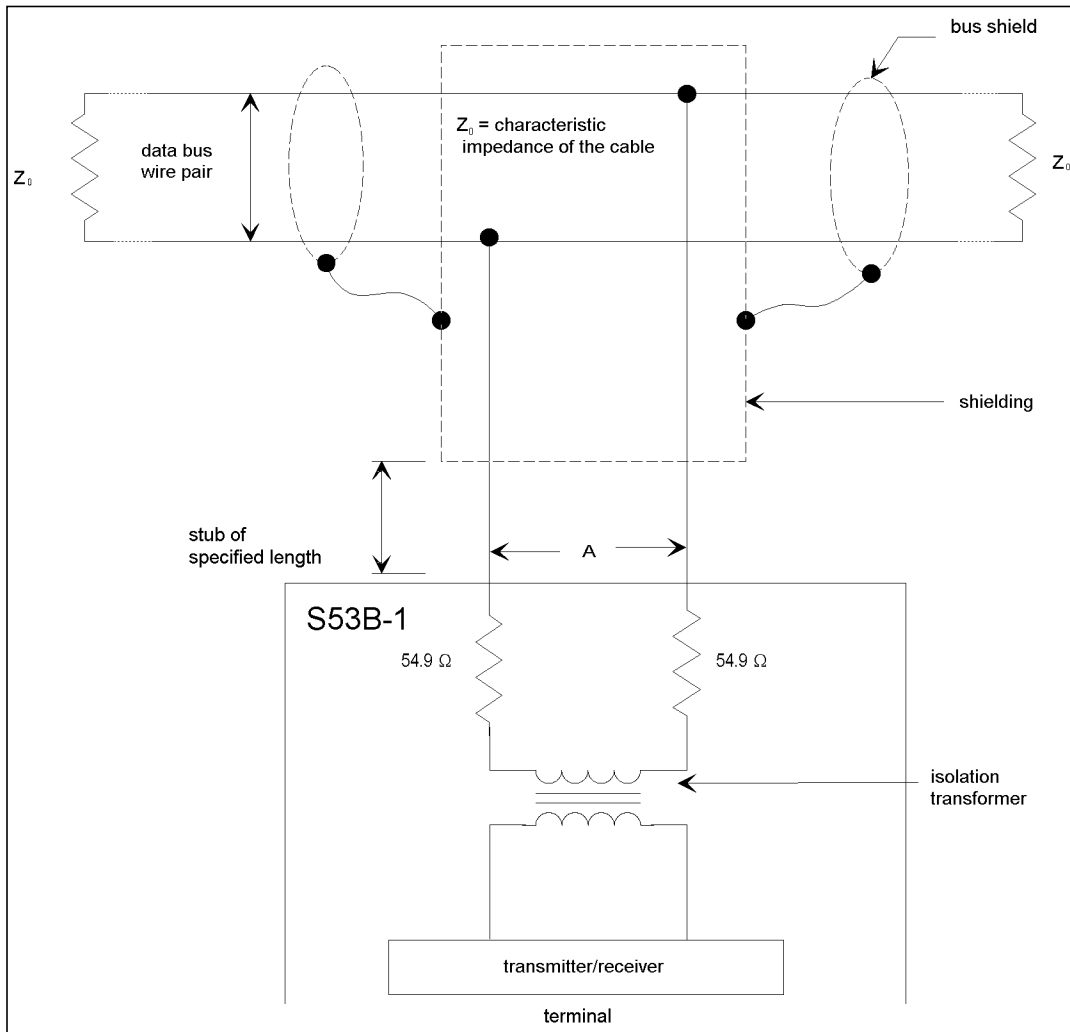


Figure 1. Direct Coupling

A transformer-coupled connection increases the possible length of the stub to 20 feet, as well as increasing robustness by protecting the bus from short circuits in the device or cable.

Transformer coupling is shown in the figure below.

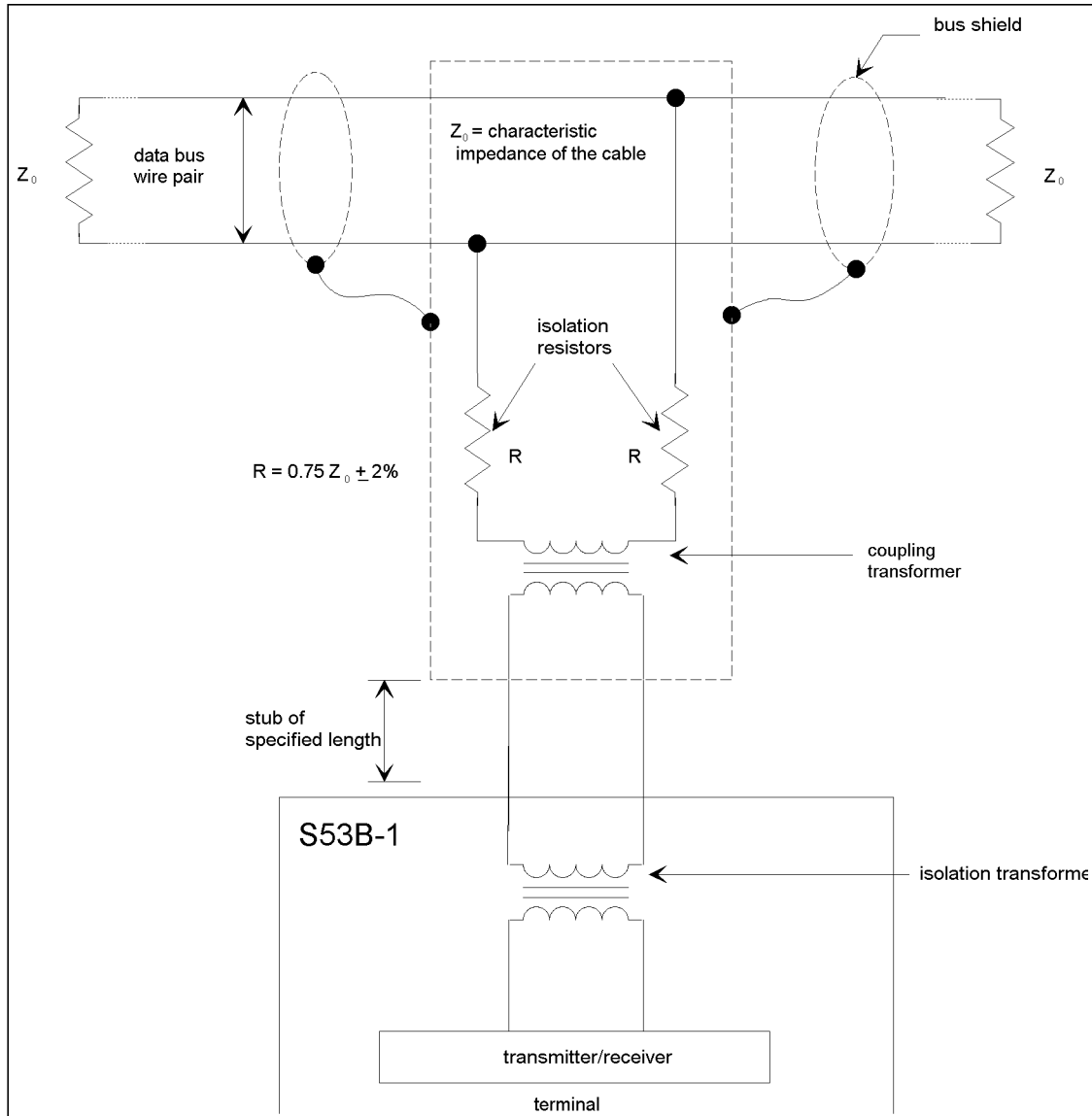


Figure 2. Transformer Coupling

The table below summarizes the physical requirements of the data bus and coupling.

Parameter	MIL-STD-1553B Requirement
<i>Transmission line</i>	
cable type	twisted-shielded pair
capacitance (wire-to-wire)	30 pF/ft, maximum
twist	4/ft (0.33 in), minimum
characteristic impedance (Z_0)	70 to 85 at 1.0 MHz
attenuation	1.5 dB/100 ft at 1.0 MHz, maximum
length of main bus	unspecified
termination	both ends terminated in resistors = Z_0 (2%)
shielding	75% coverage, minimum
<i>Cable Coupling</i>	
stub definition	short stub 1 ft long stub > 1 to 20 ft (may be exceeded)
coupler requirement	short stub direct-coupled long stub transformer-coupled
coupler transformer:	
turns ratio	1 to 1.41
input impedance	3000 minimum (75.0 KHz to 1.0 MHz)
droop	20% maximum (250 KHz)
overshoot and ringing	1 V peak (250 KHz square wave with 100 ns maximum rise and fall time)
common mode rejection	45.0 dB at 1.0 MHz
fault protection	Resistor in series with each connection equal to $(0.75 Z_0)$ 2%

Table 2. Data Bus and Coupling Requirements

Writing Applications

A basic S53B1 application has the following elements:

- a `#include "s53bi.h"` statement
- an `open()` system call with the device name (typically `"/dev/s53bi0"`)
- an `ioctl` to select the desired mode (BC, RT or BM) with calls to `ioctl()`.
- `ioctls` to configure the device (for example, the RT address when in RT mode).
- `read()` and `write()` system calls to send or receive buffers of data to or from the S53B1.
- a `close()` system call to close the device before exiting.

Example Applications

To help you get started, several example applications have been provided. Many are explained below.

bctest

Demonstrates bus controller transmitting and receiving data.

Usage `bctest -x | -r [-l n] [-w n] [-s n] [-t n] [-b n] [-u n] [-R n]`

Arguments

- x Bus controller transmits and remote terminal receives.
- r Bus controller receives and remote terminal transmits.
- l *n* Set loop count to *n*—number of times program repeats the command. The default is 1; 0 repeats the command until you press <Control-C>.
- w *n* Set word count to *n*. The maximum is 32. The default is 1.
- s *n* Set number of subaddresses to *n*. The default is 1.
- t *n* Set remote terminal address to *n*. The default is 1; 31 broadcasts to all remote terminals.
- b *n* Specify the bus channel to use. 0 = primary; 1 = secondary. The default is 0.
- u *n* Set which S53B1 board to use, in case more than one board is installed. The default is 0.
- R *n* Retry *n* times in case of error, switching to the other channel with each retry. The default is 0.

Notes You must invoke this program with at least one of the arguments `-x` or `-r`.

rttest

Demonstrates remote terminal transmitting and receiving data.

Usage `rttest -t n [-r|-x] [-u n] [-s n] [-w] [-l n]`

Arguments

- `-t n` Set remote terminal address to *n*. The default is 1.
- `-r` Wait for bus controller to specify that remote terminal is to receive data.
- `-x` Wait for bus controller to specify that remote terminal is to transmit data.
- `-u n` Set which S53B1 board to use, in case more than one board is installed. The default is 0.
- `-s n` Wait for bus controller to specify that remote terminal is to transmit data to or receive data from subaddress *n*. You can enter more than one subaddress; if you do, retype `-s` each time. For example:
`rttest -t 3 -x -s 1 -s 2`
- `-w` Wait for the user to type a <Return>, then check what the bus controller has requested.
- `-l n` Specify a loop count. A value of 0 loops forever. While looping, the device keeps track of how many subaddresses it has seen. The default is 1.

Notes You must invoke this program with the argument `-t`, and, if using `-s`, also one of `-x` or `-r`.

After initializing, the board responds to commands either to receive or to transmit as long as the remote terminal is active.

`rttest` always initializes the data to be sent when the bus controller requests it, regardless of whether it is waiting for a transmit or receive command.

s53btest

Demonstrates all functions of the S53B device driver, including error insertion and specifying the intermessage gap.

Usage `s53btest [-l n] -u n [-b n] [-n n] [-p]`

Arguments

- `-l n` Set loop count to *n*—number of times program repeats. The default is 1; 0 repeats the program until you press <Control-C>.
- `-u n` Set which S53B1 board to monitor, in case more than one board is installed.
- `-b n` Specify the bus channel to use. 0 = primary; 1 = secondary. The default is 0.
- `-n n` Set the timeout in sec for no response.
- `-p` Pause after an error.

Notes You must specify which board to use when you invoke this program.

bm

Demonstrates monitoring of all data on the 1553 bus or data transmitted by the specified S53B1 only.

Usage `bm [-c] [-h] [-v] [-s] [-f] [-m n] [-w n] -u n`

Arguments

- `-c` Count words.
- `-h` Show history only, then exit.
- `-u n` Set which S53B1 board to monitor, in case more than one board is installed. The default is 0.
- `-v` Monitor in verbose mode—show the time stamp of each word on the bus, and state of the board and bus as specified in `s53bi.h`.
- `-s` Monitor in symbolic mode—disassemble command words. Also report error status and which bus experienced the error.
- `-f` Clears all history from the bus monitor.
- `-m n` Specifies the RT monitor mask—a 32-bit hexadecimal number allowing you to specify which subaddresses to monitor. Each bit can mask the corresponding subaddress—a value of 0 ignores that subaddress, a value of 1 monitors it. The default is FFFF FFFF, which monitors all subaddresses.
- `-w n` Waits *n* words before the device returns, which can cut down on system load when appropriate. The default is 1.

Notes The bus monitor program shows the history of the bus (unless invoked with `-f`), then waits for new activity. Pressing `<Ctrl-\>` while running this program produces a summary of how many times each subaddress has been seen so far. You must invoke this program with the argument `-u`.

setdebug

Sets the debugging level and the interrupt debug level.

Usage `setdebug [-u n] [-d n] [-i n] [-r]`

Arguments

- `-u n` Set which S53B1 board to use, in case more than one board is installed. The default is 0.
- `-d n` Set the debug level. Valid values are:
 0no debugging
 1enables verbose mode when you invoke `modstat(8)`
 2traces start and end of routines
- `-i n` Set the interrupt debug level. Valid values are:
 0no debugging
 1show interrupts as bus operates
- `-r` Reports current debug level.

xmt1553

Places *stdin* into the data words of RT receive commands and outputs them to the 1553B bus. Use this program as a companion to *rcv1553*.

Usage `xmt1553 [-u n] [-b n] -t n`

Arguments

- `-u n` Set which S53B1 board to use, in case more than one board is installed. The default is 0.
- `-t n` Set which remote terminal to receive the commands.
- `-b n` Specify the bus channel to use. 0 = primary; 1 = secondary. The default is 0.

Notes You must specify the remote terminal when you invoke this program.

rcv1553

As a remote terminal, receives commands from the 1553B bus and copies the data words contained therein to *stdout*. Use this program as a companion to *xmt1553*.

Usage `rcv1553 [-u n] -t n`

Arguments

- `-u n` Set which S53B1 board to use, in case more than one board is installed. The default is 0.
- `-t n` Set which remote terminal receives the commands.

Notes You must specify the remote terminal when you invoke this program.

testdriver

Demonstrates continuous double-buffered `bc_auto` structure execution.

Usage `testdriver [-u n] [-v] [-n n] [-l n] [-w n] [-s n] [-S]`

Arguments

- `-u n` Set which S53B1 board to use, in case more than one board is installed. The default is 0.
- `-v` Turns on verbose mode. The default is off.
- `-n n` Set the size of the `bc_auto` structure array. The default is 128.
- `-l n` Set the number of times to loop through the array. The default is 30.
- `-w n` Set the intermessage gap, in microseconds. The default is 10,000 (10 ms).
- `-s n` Set the number of seconds to wait before continuing execution after a halt. The default is 0.
- `-S` Enables mapping of embedded driver memory to application memory, allowing the application to access the `bc_auto` structure directly instead of loading a separate copy. The default is off.

Notes See page 31 for further details about the `bc_auto` programming interface.

mem_s53bi

Sets the memory management options for the `bc_auto` array and remote terminal data structures; also enables or disables separate transmit and receive buffers per remote terminal.

Usage `mem_s53bi [-u n] [-r] [-f] [-s n] [-b n]`

Arguments

- `-u n` Set which S53B1 board to use, in case more than one board is installed. The default is 0.
- `-r` Generates a report to *stdout* showing current memory usage. This option can be used with other options.
- `-f` Release all memory allocated to remote terminals.
- `-s n` If *n* is set to 1, enables separate transmit and receive buffers for remote terminals. If *n* is set to 0, disables separate transmit and receive buffers for remote terminals. The default is 0. Memory usage is 2048 bytes when disabled, 4096 bytes when enabled.
- `-b n` Specify the number of elements in `bc_auto` array to allocate in the driver.

Notes Shared memory management is discussed more fully starting on page 34.

Opening The S53B1 Driver

For all 1553B modes, the initial access to the S53B1 driver is done with the *open* system call, specifying the name of the device you wish to open.

Device names are created when the driver is installed. They take the form `s53biUn`, where *U* specifies the physical board—0 in a system with only one board installed. (You can have up to eight boards installed in a system, numbered 0–7.)

n can be any integer from 0 through 31, allowing different applications to open the same board up to 32 times simultaneously. (Do not use a leading 0 for integers 0–9.) By specifying different numbers each time an application opens a device, you can have, for example, a bus controller and a bus monitor running at the same time, each accessing the sole S53B1 board installed in your system. If you establish and stick to a naming convention for your applications, you can avoid conflicts and confusion.

Reading and Writing Data To and From the S53B1

The driver provides access to bus controller, remote terminal, and bus monitor modes, using standard *read* and *write* system calls. Mode codes can be sent with an *ioctl*. As a remote terminal, many mode codes are handled by the hardware, but *ioctls* are available to set and inquire mode code information, and to allow an interrupt to the user application upon receipt of a mode code.

In Bus Controller Mode

The following example shows how to place the S53B1 in bus monitor mode:

```
u_short mode = S53B_BC;
ioctl (s53bfd, S53S_MODE, &mode);
```

In bus controller mode, you can set the S53B1 to use the desired channel. The following example sets the S53B1 to use the primary channel (channel 0):

```
u_short bus = 0; /* 0 = primary channel, 1 = secondary */
ioctl (s53bfd, S53S_BUS, &bus);
```

The *read* system call causes the bus controller to send an RT Transmit command. The remote terminal sends data and status to the bus controller. This is referred to as BC Receive. The *write* system call causes the bus controller to send an RT Receive command. The remote terminal will then receive data from the bus controller. This is referred to as BC Transmit. Examples and discussion are provided below.

See `bctest.c` for more example code showing reading and writing in bus controller mode.

In Remote Terminal Mode

The following example shows how to place the S53B1 in remote terminal mode. It also sets the remote terminal address to 1.

```
u_short mode = S53B_RT;
u_short addr = 1;
ioctl (s53bfd, S53S_MODE, &mode);
ioctl (s53bfd, S53S_MYRTADDR, &addr);
```

The *read* system call returns data that has arrived at the S53B1 board in response to a bus controller sending data with a BC Transmit command. This is referred to as RT Receive. The *write* system call loads data on the S53B1 board to be sent to the bus controller in response to a BC Receive command. This is referred to as RT Transmit. Examples and discussion are provided below.

See `rttest.c` for more example code showing reading and writing in remote terminal mode.

Queue Mask

The S53B1 also allows you to set a queue mask—a 32-bit number wherein each bit corresponds to an RT subaddress. Setting a bit to 1 queues the data for the corresponding subaddress, ensuring that no data for that subaddress will be lost. If you are only interested in the latest value for a given subaddress, set the corresponding bit in the queue mask to 0 instead. The default setting is all zeros. The following example sets the queue mask for subaddress 2:

```
u_int mask = 1 << 2;
ioctl (s53bfd, S53S_QUEUEMSK, &mask);
```

In Bus Monitor Mode

The following example shows how to place the S53B1 in bus monitor mode, in which it monitors all data on the bus:

```
u_short mode = S53B_BM;
ioctl (s53bfd, S53S_MODE, &mode);
```

The following example shows how to place the S53B1 in bus monitor transmit mode, in which it monitors all data the S53B1 has written to the bus:

```
u_short mode = S53B_BMTX;
ioctl (s53bfd, S53S_MODE, &mode);
```

The *read* system call returns the data that was monitored on the 1553B Bus. This data contains status words, command words, data, and time stamps. This is referred to as BM Receive. Examples and discussion are provided below.

Per-Word Error Facilities

Each command or status word on the 1553 bus has an associated error status. An application viewing data in bus monitor mode can detect the associated error status in the `q_elem` data structure, which is defined as follows:

```
struct q_elem {
  u_char  cmd; /* Bits 0x3: set to new message state if cmd or status */
           /* Bit 0x08: holds bus address (0 or 1) */
           /* Bits 0xf0: holds message type when received */
  u_char  error; /* Set to error number of word if any */
  u_short word; /* the word itself */
  u_int   time; /* timestamp */
};
```

Macros can set and access the bits in the `cmd` element. They are defined as follows:

```
/*
 * Defines to access fields of the cmd element of the q_elem struct.
 */
#define GET_QELEM_MSGTYPE(x)      (x >> 4)
#define SET_QELEM_MSGTYPE(x, y)  (x = (x & ~0xf0) | (y << 4))
#define GET_QELEM_CMD(x)         (x & 0x03)
#define SET_QELEM_CMD(x, y)      (x = (x & ~0x03) | y)
#define GET_QELEM_BUSADDR(x)     ((u_char) (((u_char) (x & 0x08)) >> 3))
#define SET_QELEM_BUSADDR(x, y)  (x = (x & ~0x08) | (y << 3))
```

See `bm.c` for an example of the use of these macros.

The error types are defined in `s53bi.h` and are as follows:

<code>S53B_PARITY</code>	bad parity
<code>S53B_HIWORD</code>	too many words received
<code>S53B_LOWORD</code>	too few words received, or timeout
<code>S53B_NONCONT</code>	noncontinuous error detected
<code>S53B_BADCV</code>	manchester code violation (see 1553B specification)
<code>S53B_DATAMATCH</code>	mismatch between sent or received
<code>S53B_SYNCERR</code>	unexpected sync bits

Remote Terminal Monitor Mask

The S53B1 also allows you to set a remote terminal monitor mask—a 32-bit number wherein each bit corresponds to a remote terminal. Setting a bit to 1 monitors the corresponding remote terminal; setting a bit to 0 ignores data for that remote terminal instead. The default setting is all ones. The following example sets the remote terminal monitor mask to monitor remote terminals 2 and 4:

```
u_int mask = (1 << 2) | (1 << 4);
ioctl (s53bfd, S53S_RTMONMSK, &mask);
```

See `bm.c` for more example code showing reading in bus monitor mode.

User-Defined Monitor

If you wish to be able to monitor only certain subaddresses within a remote terminal, you can do so using a user-defined monitor. The remote terminal monitor mask described above takes priority over this user-defined monitor, but if you have set the corresponding bits to 1 in the remote terminal monitor mask, you can monitor only certain subaddresses within a remote terminal, or a range of subaddresses, or change the monitoring dynamically depending on other events. This facility is useful for debugging; it also allows

complex and flexible operations such as enabling storage of certain data starting only when one subaddress is transmitting, and only until another is receiving, for example.

User-defined monitoring comprises pieces that execute in your application and pieces that execute in the driver. The application portion is defined by the example file `userbm.c`. The driver portions is defined by the example file `s53b_mon.c`, which is linked with the driver. It defines three routines: `s53b_mon`, `s53b_userset`, and `s53b_userget`.

The example file `userbm.c` makes use of this facility to enable monitoring based on the value of a variable. It shows the use of the `ioctl`s **S53S_USER** and **S53G_USER[0-9]**.

The driver calls the routine `s53b_mon` whenever it receives a command word; it passes the command word to the routine. Change this routine as required for your application. It must return 1 to enable bus monitoring and 0 to disable it.

The driver calls the routine `s53b_userset` in response to the user-defined `ioctl` **S53S_USER**, which sets values for variables used by the routine `s53b_mon`. **S53S_USER** takes a pointer to the structure `user_def`, defined in the file `s53bi.h` to contain a command and a value.

The driver calls the routine `s53b_userget` in response to any of the user-defined `ioctl`s **S53G_USER[0-9]**. The driver passes the particular `ioctl` to the routine `s53b_userget`, which gets values of variables used by the routine `s53b_mon`. Define the variables in your application to make appropriate use of any or all of these ten `ioctl`s.

See `userbm.c` and `s53b_mon.c` for an example of using a user-defined monitor.

Read and Write Data Structures

In bus controller and remote terminal modes, the *read* and *write* system calls are performed passing the address of the structures defined below. These structures are needed because the MIL-STD-1553B interface can deal with more than one remote terminal, each of which can have more than one subaddress. These structures allow the application to communicate to the driver the word counts and status of each subaddress. The structures also use a mask to allow a remote terminal to specify that it must wait for a transmission to a specific subaddress, or a request for data from a specific subaddress.

The following structures, defined in `s53bi.h`, are used to collect data received and sent to the S53B1 using `read()` and `write()` system calls.

NOTE: The UNIX operating system uses a signed integer as a file offset. Each time you read or write a file, this offset is incremented by the amount read or written. When the file offset reaches 2^{31} , it becomes negative, after which further attempts to reads or write will fail. To avoid this problem, reset the file offset pointer to 0 using `lseek (fd, 0, 0)` periodically (or before it becomes negative).

```
/*
 * buffer for read/write system call while in bus controller mode
 * BC receive/transmit
 */
```

```

struct bc_buf
{
    short    rt_addr ;                /* Remote terminal address */
    short    count[NUMSUBS] ;        /* Word count per subaddress */
    u_short  data[NUMSUBS][32] ;    /* Actual data */
    short    status[NUMSUBS] ;      /* Status received per subaddress */
}

/*
 * buffer for read/write system call while in remote terminal mode
 * RT receive/transmit
 */

struct rt_buf
{
    u_int    mask ;                  /* SA mask showing SAs desired */
    u_int    actualmask ;           /* Mask showing act SAs TXferred */
    u_short  type ;                 /* Type of return */
    short    count[NUMSUBS] ;      /* Word count per SA */
    u_short  data[NUMSUBS][32];    /* Actual data */
}

```

Using the Data Structures as a Bus Controller

For a BC Transmit, the application fills in all of the `bc_buf` structure except the `status` field. The driver fills in the `status` field after the transmit returns with the status word from the remote terminal. The `count` field is an array that contains, for each subaddress, the number of words to write from the data array. Data arrays corresponding to counts of 0 are not written. The `rt_addr` field specifies the RT address of the destination remote terminal. An address of 31 broadcasts the transmission to all remote terminals.

If you have selected bus controller mode (see **IOCTLs**, page 29), the following code transmits two words (0xa5a5, 0x5a5a) as bus controller to subaddresses 1 and 2 on remote terminal 4:

```

struct bc_buf buf;
    buf.rt_addr = 4;
    for (i = 0; i < 32; i++)
        buf.count[i] = 0;
    buf.count[1] = 2;
    buf.data[1][0] = 0xa5a5;
    buf.data[1][1] = 0x5a5a;

    buf.count[2] = 2;
    buf.data[2][0] = 0xa5a5;
    buf.data[2][1] = 0x5a5a;

    write(fd, &buf, sizeof(buf));

```

For a BC Receive, the application fills in all but `data` and `status`. The driver fills in the `status` after the transmit returns. The `count` field is the number of words to read for each subaddress. Data arrays corresponding to counts of 0 are undefined after the read.

For example, the following code performs an RT to BC transfer from remote terminal address 1:

```

/* Receive 32 words from subaddress 0. After the write, bcbuf.status[0]
 * contains status of the transmitting RT */

struct bc_buf bcbuf;

bcbuf.rt_addr = 1;
bcbuf.count[0] = 32;
read (s53bfd, bcbuf, sizeof(struct bc_buf));

```

Here's another example showing a BC to all RTs broadcast (the 31 specifies broadcasting because it is the RT broadcast address):

```

/* Send 32 words. No status available after a broadcast */

struct bc_buf bcbuf;

bcbuf.rt_addr = 31;
bcbuf.count = 32;
write (s53bfd, bcbuf, sizeof(struct bc_buf));

```

The following code transfers data from RT address 1 to RT address 2:

```

/* After the ioctl, rtrt.xmtstat and rtrt.rcvstst contain the status
 * of the transmitting RT and the receiving RT, respectively */

struct rt_rt rtrt;

rtrt.xmtrt = 1; /* RT 1 transmit */
rtrt.xmtsa = 0; /* start at subaddress 0 */
rtrt.xmcnt = 32; /* transmit 32 words */
rtrt.rcvrt = 1; /* RT 2 receive */
rtrt.rcvsa = 0; /* start at subaddress 0 */
rtrt.rcvcnt = 32; /* receive 32 words (usually the same as xmtcnt) */
ioctl (fd, S53S_RTRT, &rtrt);

```

Here's an example showing an RT to all other RTs broadcast:

```

/* Send 32 words. After the ioctl, rtrt.xmtstat contains the status
 * of the transmitting RT. No receive status after a broadcast. */

struct rt_rt rtrt;

rtrt.xmtrt = 1; /* RT 1 transmit */
rtrt.xmtsa = 0; /* start at subaddress 0 */
rtrt.xmcnt = 32; /* transmit 32 words */
rtrt.rcvrt = 31; /* All RTs receive */
rtrt.rcvsa = 0; /* start at subaddress 0 */
rtrt.rcvcnt = 32; /* receive 32 words (usually the same as xmtcnt) */
ioctl (fd, S53S_RTRT, &rtrt);

```

The structure `rt_rt` is defined as follows:

```

/* ioctl for bc issuing rt to rt */
struct rt_rt {
    u_short xmtrt ;
    u_short xmtsa ;
    u_short xmtcnt ;
    u_short xmtstat ;
    u_short rcvrt ;
    u_short rcvsa ;
    u_short rcvcnt ;
    u_short rcvstat ;
};

```

Using the Data Structures as a Remote Terminal

For an RT Transmit, the application must fill in the `count`, `mask`, and `data` fields. The driver fills in the `status`, `type`, and `actualmask` fields. It also updates the `count` fields with the word count requested by the bus controller. The `count` fields are only valid for those subaddresses that were requested by the bus controller (those having a bit set in the `actualmask` field). As with the `bc_buf` structure, the `count` is an array that holds the number of words to transfer for each subaddress. All subaddresses with a nonzero count are initially transferred to the S53B1 board. The `mask` field contains bits for subaddresses that the application expects the bus controller to request before the `write` returns.

If the application requires updating the data to be sent to the bus controller without waiting for the bus controller to request the data, the application can specify a `mask` of 0.

The `write` returns when one of the following events occurs:

- All subaddresses specified by the `mask` were requested by the bus controller. This returns a type `NORMAL`.
- One of the subaddresses in the `mask` was requested a second time by the bus controller before all the other expected subaddresses were requested. This returns a type `UNEXPECTED`. (If your `mask` is all zeros, `UNEXPECTED` will not be returned.)

```

NORMAL          1                               /* all subaddresses satisfied on write */
UNEXPECTED      4                               /* second receipt of a subaddress before all other SAs satisfied */

```

The `actualmask` field contains bits showing which subaddresses the bus controller requested. After one of these events occurs, the driver updates the `actualmask`, `count`, `type`, and `data` fields, and returns from the `write`.

For an RT Receive, the application must fill in the `mask` field. The `mask` field contains bits for subaddresses that the application expects the bus controller to send before the `read` returns. A value of 0 in this field returns immediately with updated data. The driver fills in the rest of the `rt_buf` structure, including the `actualmask` field, which shows which bits the remote terminal received.

The following example shows how to use the `mask` to make the RT wait for the BC to request a transmission of the data in subaddress 3.

```

samask = 1 << 3;                               /* bit corresponding to bit 3 */
rt_buf.mask = samask;
write (fd, &rt_buf, sizeof(rt_buf));

```

The following example shows how to use the `mask` to make the RT wait for the BC to request that the RT receive the data in subaddress 3.

```

samask = 1 << 3;                               /* bit corresponding to bit 3 */
rt_buf.mask = samask;
read (fd, &rt_buf, sizeof(rt_buf));

```

The following examples show the same general functionality using the more versatile SunOS system call *select*. This mechanism allows the application to block, poll, block with a time-out, and wait for any of several file descriptors. These examples simply block.

The following example waits for the BC to request the RT to receive data to any subaddress.

```

/* Tell S53B1 the meaning of input available. */
FD_ZERO(&inmask);
FD_SET(fd, &inmask);      /*set input mask to select input from S53B1 fd */
mask = 0;                  /*receive from any subaddress */
ioctl(fd, S53S_RT_RCVMSK, &mask) ;
select(32, &inmask, 0, 0, 0) ;                /* wait for any subaddress */
ioctl(fd, S53G_RT_RCVUPDT,&rt_buf);          /* update rt_buf with word counts,*/
                                              /* actualmask, and data */

```

The following example waits for the BC to request the RT to receive data to subaddress 3.

```

FD_ZERO(&inmask);
FD_SET(fd, &inmask);      /*set input mask to select input from S53B1 fd */
mask = 1 <<3;             /*receive from subaddress 3 */
ioctl(fd, S53S_RT_RCVMSK,&mask) ;
select(32,&inmask,0,0,0) ;                /*wait for subaddress 3 */
ioctl(fd, S53G_RT_RCVUPDT,&rt_buf);      /* update rt_buf with word counts,*/
                                              /* actualmask, and data */

```

The following example waits for the BC to request the RT to transmit data from any subaddress.

```

/* Tell S53B1 the meaning of input available. */
FD_ZERO(&outmask);
FD_SET(fd, &outmask);     /*set output mask to select input from S53B1 fd */
mask = 0;                 /*transmit to any subaddress */
ioctl(fd, S53S_RT_XMTMSK, &mask) ;
select(32, 0, &outmask, 0, 0) ;          /*wait for any subaddress */
ioctl(fd, S53G_RT_XMTUPDT,&rt_buf);     /* update rt_buf with word counts */
                                              /* and actualmask */

```

The following example waits for the BC to request the RT to either transmit or receive, with no subaddress specified.

```

FD_ZERO(&outmask);
FD_SET(fd, &outmask);
FD_ZERO(&inmask);
FD_SET(fd, &inmask);

mask = 0;
ioctl(fd, S53S_RT_RCVMSK, &mask) ;
ioctl(fd, S53S_RT_XMTMSK, &mask) ;

select(32, &inmask, &outmask, 0, 0) ;
if (FD_ISSET(fd, &inmask))
{
    ioctl(fd, S53G_RT_RCVUPDT, &rt_buf);
    printf("received command to receive\n") ;
    /* look in actualmask to determine which subaddress */
}
else
{
    ioctl(fd, S53G_RT_XMTUPDT, &rt_buf);
    printf("received command to transmit\n") ;
    /* look in actualmask to determine which subaddress */
}

```

Sending Mode Codes

A bus controller can send a mode code with the `S53S_MODECODE` *ioctl*. This *ioctl* takes a pointer to the following structure:

```

struct mc_buf
{
    u_short    rtaddr;           /* RT address to send */
    u_short    code;            /* mode code */
    u_short    data;            /* optional data */
    u_short    status;          /* status returned */
}

```

The application always fills in `rtaddr` and the mode code field itself. The driver always returns from the *ioctl* with the status set. The application fills in the data field for mode codes that sends a data word to the remote terminal—Synchronize With Data Word, Selected Transmitter Shutdown, and Override Selected Transmitter Shutdown. The driver fills in the data field for mode codes receiving data from the remote terminal—Transmit Vector Word, Transmit Last Command, and Transmit Built-in Test Word.

The following code shows an example of a bus controller sending the Transmit Vector Word mode code using the *ioctl*. Mode code constants are defined in `s53bi.h`—substitute TVW as necessary.

```

mc.code = MC_TVW;                                     /* transmit vector word */
/* set mc.data if sending a data word */
mc.rtaddr = 1;                                       /* Send to RT 1; set to 31 for broadcast */
ioctl (fd, S53S_MODECODE, &mc);
/* status returned in mc.status */
/* mc.data contains data word, if any */
printf ("TVW status %x data %x\n", mc.status, mc.data) ;

```

Receiving Mode Codes

As a remote terminal, the S53B1 can respond to any mode code with a possible application interrupt using `S53B_MODE_SIG` *ioctl*s. In addition, the S53B1 responds to certain mode codes as follows:

Mode Code	S53B1 Action
Transmit status word	Transmit contents of status register from the message processor
Transmit vector word	Transmits the contents of the vector word register from the message processor. Set with <code>S53B_LOAD_TVW</code> .
Synchronize with data word	To be returned by the <code>S53B_GET_MODECODE</code> <i>ioctl</i>
Transmit last command	Transmits the contents of the command/status word register from the message processor
Transmit built-in test word	Transmits the contents of the built-in test error register from the message processor

Table 3. Mode Code Responses

In order to respond to these mode codes, the application can ask for a signal upon receipt of a mode code with the `S53B_MODE_SIG` *ioctl*. This *ioctl* takes a pointer to a `mode_sig` structure.

```

struct mode_sig
{
  u_intmask ; /* mask of the requested mode codes */
  u_shortsignal ; /* signal to be sent on mode code */
}

```

After receiving the signal (or at any time), the application can execute an `S53BG_MODECODE` *ioctl*. This *ioctl* takes a pointer to a `mode_data` struct to return the last mode code received and associated data.

```

struct mode_data
{
  u_shortdata ; /* optional data */
  u_shortcode ; /* mode code received */
}

```

The *ioctl* `S53G_MODECOUNT` allows you to get the number of mode codes queued for a remote terminal. It takes as its third argument an address of an unsigned integer in which to store the result. The *ioctl* `S53G_MODEEQ` allows you to get the entire queue of mode codes for a remote terminal. It takes a pointer to the `mode_data` struct. The example file `rtmodeq.c` shows how to use these *ioctl*s.

Status Bits

*ioctl*s are provided so that a remote terminal can set each of the status bits individually. Other than the service request bit, these are straightforward and can be set or cleared by sending the *ioctl*s defined in `s53bi.h`. The service request bit is unique, however; specific *ioctl*s cause the bit to be cleared pending a specific event. The `S53B_SRQ_X` clears the service request bit on the next receipt of a bus controller transmit request. The `S53B_SRQ_V` clears the service request bit on the next receipt of a Transmit Vector Word mode code.

The S53B1 can use the SRQ bit to send a software signal to an application. To do so, the application must first send the TVW mode code. If the SRQ bit is set, the driver will send a signal to the application. The following code shows an example of this:

```
{
    extern int your_signal_handler() ;
    u_short mode ;

    /* register signal to be sent */
    mode = SIGUSR1 ;    /* any signal can be used */
    ioctl(bcfd, S53S_SRQSIG, &mode) ;
    signal(SIGUSR1, your_signal_handler);
}
```

IOCTLs

*ioctl*s are defined in `s53bi.h`. Applications can use them to access the device driver. See example programs for details on their uses.

Specifying Error Insertion and Intermessage Gap for System Tests

In order to allow you to conduct realistic tests of your system, the S53B1 allows you to insert errors and specify intermessage gap times. The intermessage gap is the number of `s` between commands. In bus controller mode, use the `bc_auto` structure (described below) to insert errors or set the intermessage gap.

In order to ensure that the end of the previous command is received before the next command is sent, commands cannot be issued infinitely fast. Therefore, specifying an intermessage gap time of less than 20 `s` does not change the driver behavior.

In remote terminal mode, use the `S53S_RT_ERR` *ioctl* to insert the specified error. Set it to zero (the default) to specify that no errors are to be inserted.

The following errors can be inserted.

S53_ Error Code	Generated by	Description
PRTY_ERROR	BC or RT	A word is transmitted with a parity error.
GAP_ERROR	BC or RT	A word that should be transmitted immediately after another word is transmitted after a gap instead.
HIWORD_ERROR	RT only	The RT sends more words than requested.
LOWORD_ERRO R	RT only	The RT sends fewer words than requested.
NORESP_ERROR	RT only	The RT does not respond.
SLOW_ERROR	RT only	The RT responds with status more slowly than the required 12 s.
SYNC_ERROR	RT only	The RT sends a data SYNC pattern with the status word.

Table 4. Error Codes

***bc_auto* Structure**

The *bc_auto* structure specified in *s53b.h* is used in bus controller mode to insert errors or to specify an intermessage gap for the purposes of testing your system. Each *bc_auto* structure defines one command to send on the bus. You can define a list of commands to send consecutively by defining an array of *bc_auto* structures. After sending the command specified by each structure, the embedded SPARC waits a specified number of seconds and sends the next command without the host computer's participation.

The *bc_auto* structure contains the following fields:

<code>cmd = CMDWORD(rt,sa,wc,tr)</code>	The command to send. The macro <code>CMDWORD</code> builds the command using the specified remote terminal, subaddress, word count, and transmit bit.
<code>cmd2</code>	Set this optional field to initiate a remote terminal-to-remote terminal transfer. Otherwise set to zero.
<code>status</code>	The status returned by the remote terminal.
<code>status2</code>	Optional second status word returned from a remote terminal-to-remote terminal transfer. Otherwise set to zero.
<code>waittime</code>	The number of seconds to wait before issuing the next command. (Specifying a waittime of less than 20 s does not change driver behavior.)
<code>error</code>	Specifies the error to insert. A zero issues no error. A 32-word array specifying the associated data. When the bus controller is sending data to a remote terminal, initialize this field with the required data. When the bus controller is receiving data from a remote terminal, the array contains the data when the command is completed. After you initialize the array of <i>bc_auto</i> structures, you must load the array and start it executing.

Use the following parameters to *ioctl*:

1. Specify the number of elements the array contains with **S53S_AUTO_SIZE**.
2. Specify the number of commands to execute with **S53S_AUTO_TODO**. Ordinarily, if you wish each command in the array to be sent once, specify a number equal to the number of elements in the array

specified in **S53S_AUTO_SIZE**, above. However, you can specify that the commands in the array be issued more than once. To do so, specify a larger number than **S53S_AUTO_SIZE**. Your application loops through the array until the number of commands sent equals the number you specified.

If you pass 0 as an argument to **S53S_AUTO_TODO**, execution will continue until you send the ioctl parameter **S53S_AUTO_STOP**.

3. Specify the address of the array with **S53S_LOAD**. **S53S_LOAD** copies the data from the application to the start of the array in the S53B1.
4. Start execution with **S53S_GO**.

Other parameters specify optional behavior:

- If the application must wait until certain commands have all been issued before resuming, specify that with **S53G_AUTO_WAITCNT**. The third parameter is the address of an unsigned integer specifying an absolute offset into the `bc_auto` array. The driver counts the number of commands executed, starting at 0 and incrementing once for each command executed; it wraps at 2^{32} . Your application will block until the specified number of commands have executed.
- Check the number of commands that have been executed with **S53S_CNT**.
- Specify the number of commands to continue executing (after the previous set specified by **S53S_AUTO_TODO** or **S53S_AUTO_CONT**, with **S53S_AUTO_CONT**. The third parameter is the address of an unsigned integer specifying an absolute offset into the `bc_auto` array. The driver counts the number of commands executed, starting at 0 and incrementing once for each command executed; it wraps at 2^{32} . If processing of `bc_auto` commands has stopped, it will resume and the specified number of commands will be executed. If processing is ongoing, it will continue until the specified number of commands have been executed, overriding any previously specified stopping points.
- Check the number of errors that have been encountered with **S53S_ERR**.
- Specify an offset into the `bc_auto` array using **S53S_AUTO_OFFSET**. The argument to this parameter is the offset into the array at which to start loading data; run this before **S53S_LOAD** to modify where the load occurs (or the unload using **S53S-DUMP**). This is useful for double-buffering—you can load half of an array and cause the S53B1 to begin execution of that half while you specify the offset and load new information into the second half. This enables continuous `bc_auto` execution. This feature allows you to output or input data continuously, treating the `bc_auto` structure in hardware as a circular buffer. For example, if you know that the S53B1 has finished with commands in structure locations 50–99, you can start reloading at location 50 while the S53B1 processes commands in locations 0–49. See the example program `testdriver.c` (documented on page 18) for an example of this use.

The following example places the S53B1 in bus controller mode and loops through the list of commands three times, inserting an intermessage gap of 1000 s (1 ms).

```
u_short mode = S53B_BC;
struct bc_auto testbuf[10];
ioctl (s53bfd, S53S_MODE, &mode);
```

```

/*
 * test transfer of 3 passes through autotest
 * with a delay of 1 ms at end of frame without errors
 * each command issues an RT receive command of ten words
 */
wc = 10 ; /* Set word count */
tr = 0 ; /* Set transmit bit to receive */
rt = 1 ; /* Set remote terminal address */
tmpval = 0x1111 ;
for(i = 0 ; i < 10 ; i++)
{
    a_p = &testbuf[i] ;
    sa = i + 1 ; /* Set subaddress */
    a_p->cmd = CMDWORD(rt,sa,wc,tr) ;
    a_p->cmd2 = 0 ;
    for(j = 0 ; j < 32 ; j++) /* Initialize test data */
        a_p->data[j] = tmpval ;
    tmpval += 0x1111 ;
    a_p->status = 0 ;
    a_p->status2 = 0 ;
    a_p->error = 0 ;
    a_p->waittime = 1000 ;
}
/* set size to load autotest */
size = 10 ;
ioctl(bcfid,S53S_AUTO_SIZE,&size) ;
/* set number of autotest items to execute */
todo = 30 ;
ioctl(bcfid,S53S_AUTO_TODO,&todo) ;
/* load it */
addr = (u_int)testbuf ;
ioctl(bcfid,S53S_AUTO_LOAD,&addr) ;
/* start executing */
ioctl(bcfid,S53S_AUTO_GO,0) ;

```

The following example places the S53B1 in bus controller mode and inserts a gap error in the command transmitted.

```

u_short mode = S53B_BC;
struct bc_auto testbuf[10];
ioctl (s53bfd, S53S_MODE, &mode);
wc = 5 ;
tr = 0 ;
sa = 1 ;
rt = 1 ;
a_p = testbuf ;
a_p->cmd = CMDWORD(rt,sa,wc,tr) ;
a_p->cmd2 = 0 ;
a_p->status = 0 ;
a_p->status2 = 0 ;
a_p->waittime = 0 ;
a_p->error = S53_GAP_ERROR ;
/* set size, load, and go */
size = 1 ;
ioctl(bcfd,S53S_AUTO_SIZE,&size) ;
/* set number of autotest items to execute */
todo = 1 ;
ioctl(bcfd,S53S_AUTO_TODO,&todo) ;
/* load it */
addr = (u_int)testbuf ;
ioctl(bcfd,S53S_AUTO_LOAD,&addr) ;
/* start it */
ioctl(bcfd,S53S_AUTO_GO,0) ;

```

The following example places the S53B1 in remote terminal mode, sets the remote terminal address to 1, and inserts a parity error whenever a command is received from the bus controller.

```

u_short mode = S53B_RT;
u_short addr = 1;
u_short rt_err = S53_PRTY_ERROR;
ioctl (s53bfd, S53S_MODE, &mode);
ioctl (s53bfd, S53S_MYRTADDR, &addr);
ioctl (s53bfd,S53S_RT_ERR,&rt_err);

```

See `s53btest.c` for more example code showing error insertion and specifying intermessage gap times.

Scheduling `bc_auto` Structures

The example program `bc_auto_sched.c` provides examples of various scheduling mechanisms for `bc_auto` structures. The four most significant bits of the error element of the `bc_auto` structure select the scheduling type. When these bits are all 0, the structure is scheduled as usual. The bits are defined in `s53bi.h` as follows:

```

#define S53_SCHED_NRM          0x0000
#define S53_SCHED_ABS          0x1000
#define S53_SCHED_REL          0x2000
#define S53_SCHED_HW           0x3000
#define S53_NOOP                0x8000

```

The NOOP bit is independent of the other scheduling bits, and therefore can be used in concert with them. The absolute, relative, and hardware scheduling operations share two bits and are therefore mutually exclusive. One bit is reserved for future use.

NOOP Bit

The NOOP bit can be used to create a `bc_auto` structure that is useful for scheduling the following `bc_auto` command word. When the NOOP bit is set, the S53B1 executes no commands; all that it does is to wait for the specified intermessage gap time. The intermessage gap can be controlled by setting the `waittime` element of the `bc_auto` structure, or by enabling one of the scheduling operations. The `bc_auto` operation is performed first, and then the intermessage gap is observed as specified by the requested scheduling operation. Therefore, you can schedule a `bc_auto` structure by preceding it with a NOOP combined with one of the scheduling operations.

Absolute Scheduling

The host computer has a clock, and a clock is also contained within the embedded microprocessor on the S53B1. Absolute scheduling is concerned with both clocks and two parameters: a global variable that indicates an absolute time—you can think of it as the time at which an alarm will go off—and an offset in microseconds in the `waittime` element of the `bc_auto` structure. In order to make use of absolute scheduling, you must first initialize the embedded clock using the time of day specified by the host computer's clock. You can then set the alarm. When the embedded clock reaches the time specified by the alarm, the S53B1 waits for the time specified by the `waittime` element, and then executes the next command.

The following example initializes the S53B1 clock and then sets the alarm for thirty seconds later.

```
{
    struct timeval tm;

    /* Initialize the absolute timer on the s53bi */
    gettimeofday(&tm);
    ioctl(bcfid, S53S_TIMEVAL, &tm);

    /* Set the absolute variable to current time + 30 seconds */
    tm.tv_sec += 30 ;
    ioctl (bcfd, S53S_TIMEABS, &tm);
}
```

Relative Scheduling

Relative scheduling affects the length of the intermessage gap. You can use relative scheduling by setting the SCHED bits to `S53_SCHED_REL` in a set of `bc_auto` structures. Set the `waittime` in the first structure to zero to initialize a time marker. Then set the `waittimes` in the following structures to increasing values; these will be interpreted as microsecond offsets from the time marker.

Hardware Scheduling

A `bc_auto` structure with the `S53_SCHED_HW` bit set observes an intermessage gap based not on time, but on a specified number of external hardware interrupts. These are specified by the `waittime` element. Values of 0 and 1 both wait for one interrupt. Values greater than 1 wait for the specified number of interrupts before ending the intermessage gap.

In order to make use of hardware scheduling, the S53B1 board must be modified to connect to an external interrupt line. Contact Engineering Design Team for details.

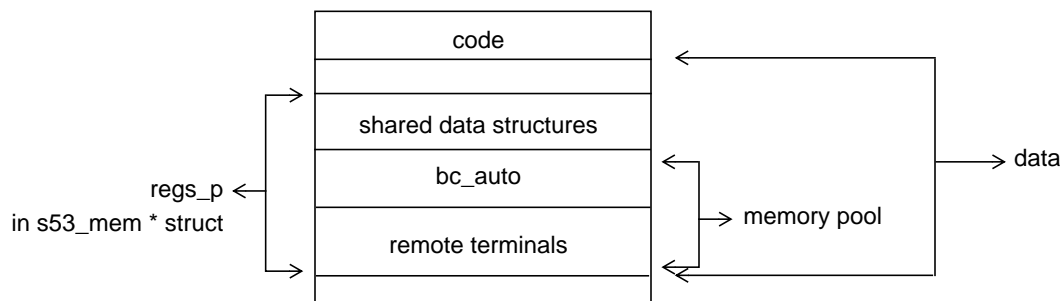
Shared Memory Management

Your application can directly access memory on the S53B1. Memory represented by the `bc_auto` structure arrays and remote terminal data structures can be mapped from the S53B1 to the memory addressed by your application, using interface routines provided in *libs53bi.a*. These facilities also permit you to dynamically resize the memory allocated to the `bc_auto` structure arrays and remote terminal data

structures; they also allow each remote terminal to have separate transmit and receive buffers. If you choose to use separate transmit and receive buffers, each remote terminal uses 4 KB instead of 2 KB of memory, which will limit the number of remote terminals available to approximately twenty, depending upon other details of memory usage.

Memory is allocated dynamically from a common pool. The memory pool is implemented as an array of unsigned shorts sized to use all available memory on the S53B1. `bc_auto` structures are allocated starting at the lowest array index and grow upwards in a heaplike fashion. Remote terminal subaddress buffers are allocated starting at the highest array index and grow downwards in a stacklike fashion. All elements contend for the same memory. Remote terminal subaddress buffers are allocated dynamically when each remote terminal is activated. This memory remains allocated until you explicitly free it, reboot the system, or reload the driver. Memory for RT 31 (broadcast) is allocated when the first remote terminal is activated. Figure 3 illustrates how memory is allocated. The example program `mem_s53bi` documented on page 19 allows you to set various options for memory management (and report on memory usage), using the memory management library routines discussed below.

Figure 3. S53B1 Memory



The routine `s53_shmget` maps S53B1 shared memory into the address space of the current process, returning a handle, `s53_mem *`, with which you can access the shared data structures. The routine is defined as follows:

```
s53_mem *
s53_shmget (fd)
int fd; /* file descriptor for S53B1 device */
```

The `s53_mem` structure is defined as follows:

```
typedef struct {
    struct s53bi_regs *emb; /* contains regs_p memory pointer */
    struct rt_monitor *rp; /* reserved */
} s53_mem;
```

The routine `s53_enable` enables the specified remote terminal and dynamically allocates memory for it. By default, the transmit and receive buffers are shared. To enable separate buffers for these purposes, use the `ioctl S53S_LOOPBACK` with an argument of 0. The routine is defined as follows:

```
int
s53_enable (regs_p, rtnum)
struct s53bi_regs *regs_p;
int rtnum; /* remote terminal number */
```

The routine `s53_enable` returns 1 if it is successful and 0 if it fails, due to insufficient memory.

You can determine how much memory is available by checking the `mempool_free` element of the `regs_p` structure. This element holds the number of unsigned shorts available. If you need more memory,

you can shrink the number of `bc_auto` structures, which, by default, is 512. To do so, call the *ioctl* **S53S_BC_AUTO_POOL_SIZE** with an argument of a smaller number, or free remote terminal memory using `s53_restmem`, discussed below.

The routine `s53_disable` disables the specified remote terminal. Subaddress memory remains allocated. The routine is defined as follows:

```
void
s53_disable (regs_p, rtnum)
struct s53bi_regs *regs_p;
int rtnum; /* remote terminal number */
```

The routine `s53_shmunget` unmaps S53B1 shared memory from the address space of the current process. The routine is defined as follows:

```
void
s53_shmunget (fd, ptr)
int fd; /* file descriptor for S53B1 device */
s53_mem *ptr;
```

The routine `s53_resetmem` frees all remote terminal subaddress and performs memory reset housekeeping functions. You can use this routine to switch between separate and shared transmit and receive buffers for remote terminals, or to allocate more memory for more `bc_auto` structures. Memory for remote terminal structures will be allocated as they are enabled. The routine is defined as follows:

```
void
s53_resetmem (regs_p)
struct s53bi_regs *regs_p;
```

The routine `s53_rt_change_trdata` allows you to update a separate subaddress transmit buffer while you are previously written data is being transmitted, without overwriting the values being transmitted. You must specify the subaddress buffer by the remote terminal number and specific subaddress number. This routine returns 1 if successful and 0 if it fails, due to insufficient RT data memory for the temporary copy. The routine is defined as follows:

```
int
s53_rt_change_trdata (regs_p, datap, rt, sa)
struct s53bi_regs *regs_p;
u_short *datap; /* pointer to an array of 32 unsigned shorts */
int rt, sa; /* remote terminal and subaddress */
```

Using the Commander

The S53B1 includes the program `commander`, an OPEN LOOK-based graphical interface to allow you to use your Sun workstation as a 1553B bus analysis tool. Using the Commander, you can access the S53B1 board, monitor commands and data transmitted and received, and edit data before it is transmitted.

To run the Commander, you must first be running OPEN LOOK. You can invoke the Commander by entering:

```
commander
```

By default, the Commander connects to the device named `/dev/s53bi0`. If you have more than one S53B1 board installed in your system, you can invoke the Commander on a different device by adding the specific device name. For example:

```
commander /dev/s53bi1
```

In the bottom right corner, the Commander shows the device address, using the convention described on page 37.

To exit the Commander, press the **Quit** button at the upper left.

The Commander can operate as a bus controller, remote terminal, or bus monitor of either all traffic or just that which the S53B1 writes to the bus (called bus monitor transmit). To select the desired mode, click the right mouse button on the button labeled **Mode**, and select from the menu that pops up. Each mode is shown and described below.

Acting as a bus controller, the Commander can send data to any remote terminal or broadcast to all of them. It can also receive data from any remote terminal.

Click on **Primary** or **Secondary** to select the channel you wish to use.

Choose the remote terminal(s) by specifying the address in the **Target RT** field. You can type the desired address or use the increment or decrement buttons.

Type the number of times you wish to repeat the operation in the **Repeat Count** field, or use the increment or decrement buttons. A value of 0 loops until you press the same button you used to start the operation.

In the **Transmit Data** or **Receive Data** pane, specify the subaddress in the **Subaddress** field. You can type the desired subaddress or use the increment or decrement buttons. Specify the number of words to transmit or receive in the **Count** field. You can type the desired count or use the increment or decrement buttons.

Each field in the **Transmit Data** pane is completely editable. The number of words specified in the **Count** field determines the number of fields filled in with default data. Data values appear in hexadecimal. To edit the data, select the data you wish to modify, and type the desired value. Then press <Return>. You can also use <Tab> to reach the next field, and <Control-U> or <Control-W> to empty the field in which the cursor appears.

When you are ready, click the **BC Transmit** or **BC Receive** button to begin the desired operation.

The screenshot shows the Commander software interface in 'Bus Controller' mode. At the top, there are buttons for 'Mode', 'RT Transmit', and 'RT Receive'. The 'Mode' section has 'Bus Controller' selected. Below it, there are fields for 'Target RT' and 'Repeat Count'. The main area is divided into two sections: 'Transmit Data' and 'Receive Data'. Each section has a 'Subaddress' field and a 'Count' field. The 'Transmit Data' section shows a grid of hexadecimal values for subaddresses 1 through 15. The 'Receive Data' section shows a similar grid of hexadecimal values.

Figure 4. The Commander as Bus Controller

Acting as a remote terminal, the Commander can emulate remote terminal devices under development. You can edit the data to be sent and examine the data that was received in response to commands from the bus controller.

Specify the remote terminal address in the **RT Address** field. You can type the desired address or use the increment or decrement buttons.

The screenshot shows the Commander software interface in 'Remote Terminal' mode. At the top, there are buttons for 'Mode', 'RT Transmit', and 'RT Receive'. The 'Mode' section has 'Remote Terminal' selected. Below it, there is a 'Completion Masks' button and an 'RT Address' field. The main area is divided into two sections: 'Transmit Data' and 'Receive Data'. Each section has a 'Subaddress' field and a 'Count' field. The 'Transmit Data' section shows a grid of hexadecimal values for subaddresses 1 through 15. The 'Receive Data' section shows a similar grid of hexadecimal values.

Figure 5. The Commander as Remote Terminal

Specify the subaddresses to which the application expects the bus controller to write, or from which the bus controller will read, using the **Completion Masks** button. When you press it, a window pops up as shown below.

Click on the subaddress(es) for which you wish to set a mask, either in the **Transmit Mask** pane to send data, or in the **Receive Mask** pane to receive data. Click on the **Set All** button to select all the subaddresses. Click on the **Clear All** button to clear all selected subaddresses.

When you are finished, click in the upper left corner to return to the main Commander window.

The screenshot shows a window titled "Completion Mask Bits". It is divided into two main sections: "Transmit Mask" and "Receive Mask". Each section has a "Set All" button and a "Clear All" button. Below these buttons is a grid of 32 bit positions, arranged in four rows and eight columns. The bit positions are labeled from 31 down to 0. The "Transmit Mask" section is currently active, and the "Receive Mask" section is inactive.

Figure 6. Setting the Masks

In the **Transmit Data** or **Receive Data** pane, specify the subaddress in the **Subaddress** field. You can type the desired subaddress or use the increment or decrement buttons. Specify the number of words to transmit or receive in the **Count** field. You can type the desired count or use the increment or decrement buttons.

Each field in the **Transmit Data** pane is completely editable. The number of words specified in the **Count** field determines the number of fields filled in with default data. Data values appear in hexadecimal. To edit the data, select the data you wish to modify, and type the desired value. Then press <Return>. You can also use <Tab> to reach the next field, and <Control-U> or <Control-W> to empty the field in which the cursor appears. When you are ready, click the **RT Transmit** or **RT Receive** button to begin the desired operation.

Acting as a bus monitor, the Commander passively records the information that travels across the bus, or specific statistics on it. The data is displayed in a scrollable window.

Click on the **Summary** button to show statistics. In this mode, the column labeled **RT** displays the remote terminal address. **ST** displays its status. **SA** displays the subaddress. **WC** displays the word count. **T/R** specifies whether the remote terminal transmitted or received. **Count** displays the number of times the operation was repeated.

The screenshot shows the EDT S53B Commander v2.1 interface. At the top, there is a 'Quit' button and a 'Mode' section with a 'Bus Monitor' button. To the right of the 'Mode' section is a 'Stop Monitor' button. Below this is a 'Display' section with 'Raw' and 'Summary' radio buttons, where 'Summary' is selected. The main area contains a table with the following data:

RT	ST	SA	WC	T/R	Count
0000	0111	0000	0000	Receive	0001
0000	0112	0000	0000	Receive	0001
0000	0113	0000	0000	Receive	0001
0000	0114	0000	0000	Receive	0001
0000	0115	0000	0000	Receive	0001
0000	0116	0000	0000	Receive	0001
0000	0117	0000	0000	Receive	0001
0000	0118	0000	0000	Receive	0001
0000	0119	0000	0000	Receive	0001
0000	011a	0000	0000	Receive	0001
0000	011b	0000	0000	Receive	0001
0000	011c	0000	0000	Receive	0001
0000	011d	0000	0000	Receive	0001
0000	011e	0000	0000	Receive	0001
0000	011f	0000	0000	Receive	0001

At the bottom right of the window, the text '/dev/s53b030' is visible.

Figure 7. The Commander as Bus Monitor, Summarizing

When you are ready, click the **Start Monitor** button to start. The bus monitor begins, and the button changes to read **Stop**. Click it again to stop the bus monitor.

Acting as a bus monitor in transmit mode, the Commander passively records statistics on only information that the S53B1 has written to the bus.

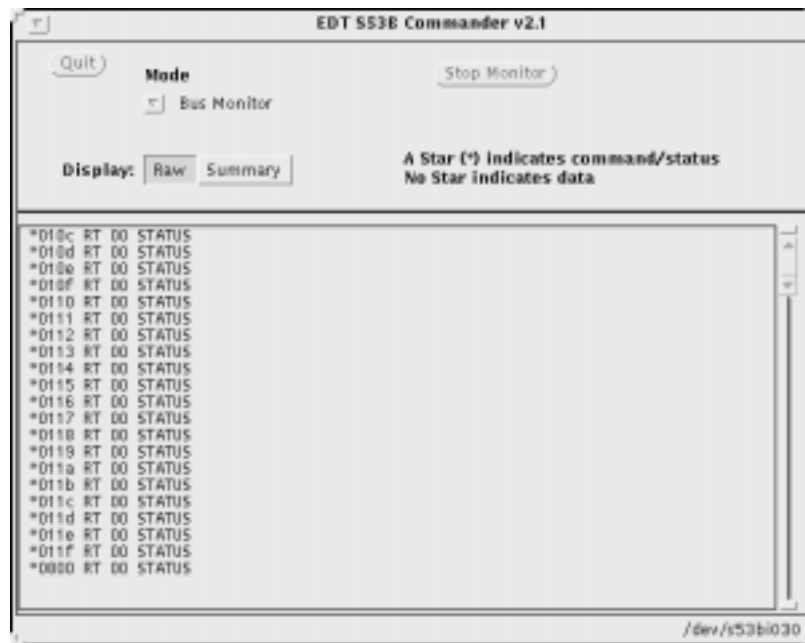


Figure 8. The Commander as Bus Monitor, Showing Data

You can use the Commander to show you the effects of the example programs. Start the Commander in bus monitor mode, and, in another window, type commands to the example programs. The bus monitor shows you the activity on the S53B1 occurring as a result of your commands.

NOTE: Mode codes are not implemented for this release of the Commander.

Registers

The S53B1 includes hardware registers that are usually accessed only by the embedded SPARC microprocessor. However, the SBus can access them for debugging or diagnostic purposes if required. If you need to use the SBus to access the S53B1 hardware registers, first halt the embedded SPARC microprocessor using the board control register as described below. In this way, you ensure against your application and the SPARC both trying to access the same register at the same time.

NOTE: The SunOS allows the application program direct access to the S53B1 hardware. Engineering Design Team, Inc. recommends using the driver, however. Future implementations will be compatible at the driver interface, but may not be hardware-compatible.

The figure below shows the SBus addresses of the S53B1 hardware registers.

0x0004.003C	primary channel, transmit command, read time stamp	not used		
0x0004.0038	primary channel, receive data or command, xmit data	not used		
0x0004.0034	primary channel, status and reset	not used		
0x0004.0030	primary channel, command	not used		
0x0004.002C	2ndary channel, transmit command, read time stamp	not used		
0x0004.0028	2ndary channel, receive data or command, xmit data	not used		
0x0004.0024	secondary channel, status and reset	not used		
0x0004.0020	secondary channel, command	not used		
not used				
0x0004.000C	board control, board reset	not used		
0x0004.0008	board control, SPARC halt/run	not used		
0x0004.0004	board control, SBus interrupt	not used		
0x0004.0000	board control, SPARC interrupt	not used		
0x0002.0000	RAM			
0x0000.0000	EPROM			
Byte	0	1	2	3
Word	0		1	

Figure 9. SBus Addresses of S53B1 Hardware Registers

The figure below shows the embedded SPARC addresses of the S53B1 hardware registers.

0x0006.003C	primary channel, transmit command, read time stamp	not used		
0x0006.0038	primary channel, receive data or command, xmit data	not used		
0x0006.0034	primary channel, status and reset	not used		
0x0006.0030	primary channel, command	not used		
0x0006.002C	2ndary channel, transmit command, read time stamp	not used		
0x0006.0028	2ndary channel, receive data or command, xmit data	not used		
0x0006.0024	secondary channel, status and reset	not used		
0x0006.0020	secondary channel, command	not used		
not used				
0x0006.000C	board control, board reset	not used		
0x0006.0008	board control, SPARC halt/run	not used		
0x0006.0004	board control, SBus interrupt	not used		
0x0006.0000	board control, SPARC interrupt	not used		
0x0002.0000	EPROM			
0x0000.0000	RAM			
Byte	0	1	2	3
Word	0		1	

Figure 10. SPARC Addresses of S53B1 Hardware Registers

Data Registers

These 16-bit readable and writable registers hold the data to be transmitted or received. Both the primary and secondary channels have two data registers. One register transmits and receives data. The other register transmits commands; reading this register allows you to determine the value of a s counter. This counter can provide a time stamp for an event on the bus; it is cleared when transmission is enabled using bit 0 of the command register.

The two are actually the same register; the S53B1 uses the different addresses to determine whether to transmit a data word or a command word on the bus.

Status Registers

Both the primary and secondary channels have a 16-bit read-only status register, used to check board status. The interrupts described in bits 7–2 cannot be asserted unless they are enabled using the corresponding bits of the command register.

Bit	S53B_	Description
15–8	UNUSED	unused
7	INT	1 = interrupt to embedded SPARC asserted. Level 1 = interrupt to primary channel. Level 2 = interrupt to secondary channel. This interrupt is asserted if any interrupt condition below is asserted.
6	TX_RDY	1 = transmit buffer is ready for next word
5	RX_RDY	1 = data in receive buffer
4	OVRRUN	1 = overrun occurred. A word was received from the bus before the previous word was read; the receive buffer was overwritten.
3	UNDRRUN	1 = underrun occurred. The next word was not written soon enough for back-to-back transmission. Always true at end of block.
2	RSPEXC	1 = response time exceeded. The time between the last transition of data received and the start of the transmission exceeded 12 sec.
1	RX_PRTY	The parity of the last received word. 1 = good, 0 = bad.
0	RX_SYNC	1 = the last word received is data. 0 = the last word received is command.

Table 5. Status Register Bit Definitions

Reset Registers

Both the primary and secondary channels have a 16-bit write-only reset register, used to reset interrupts. Write a 1 to the appropriate bit to reset. You need not clear these bits as the value is not stored.

Bit	S53B_	Description
15–5	UNUSED	unused. Set to zero.
4	OVER_RST	Resets the overrun interrupt (bit 4 of status register).
3	UNDER_RST	Resets the underrun interrupt (bit 3 of status register).
2	EXC_RST	Resets the response-time-exceeded interrupt (bit 2 of status register).
1	TX_RST	Resets the transmission-buffer-ready interrupt (bit 6 of status register).
0	RX_RST	Resets the data-buffer-ready interrupt (bit 5 of status register).

Table 6. Reset Register Bit Definitions

Command Registers

Both the primary and secondary channels have a 16-bit readable and writeable command register, used to enable interrupts, transmit parity, and enable the transmitter. In all cases, a value of one enables, and a value of zero disables.

Bit	S53B_	Description
15–8	UNUSED	unused
7	MNINT_ENB	Enables interrupt to embedded SPARC. It must be enabled if any interrupt is to be enabled.
6	XMT_INT_ENB	Enables transmit interrupt. The transmit interrupt is then set when the last word written to the data register is being transmitted, and the transmit buffer is ready for the next.
5	REC_INT_ENB	Enables receive interrupt. The receive interrupt is then set when a word received from the bus is in the receive buffer. Bit 0 of the status register tells whether the word is command or data. Bit 1 of the status register tells whether its parity was good or bad.
4	OVER_ENB	Enables overrun interrupt. Set if overrun occurs.
3	UNDER_ENB	Enables underrun interrupt. Set if underrun occurs.
2	RSPXCD_ENB	Enables interrupt when response time was exceeded.
1	TX_PRTY	Transmit parity. Useful for testing. 1 = good, 0 = bad.
0	TX_ENBL	Enables transmitter. Must be set to transmit.

Table 7. Command Register Bit Definitions

Board Control Registers

The following four 16-bit readable and writeable registers are used to reset the board or handle its communications with the SBus and the embedded SPARC microprocessor.

Bit	S53B_	Description
15	INT	The SBus writes a 1 to assert a level 0 interrupt to the embedded SPARC microprocessor. The SPARC writes a 0 to clear it.
14-0	UNUSED	unused

Table 8. Board Control Interrupt Register Bit Definitions

Bit	S53B_	Description
15	SBUS_INT	The embedded SPARC microprocessor writes a 1 to interrupt the SBus. The SBus writes a 0 to clear it.
14-0	UNUSED	unused

Table 9. Board Control SBus Interrupt Register Bit Definitions

The S53B1 is configured to use SBus interrupt level 2. Do not change the interrupt level without customizing the code in the EEPROM. Contact technical assistance at Engineering Design Team, Inc.

Bit	S53B_	Description
15	HALT	A value of 1 halts and resets the embedded SPARC microprocessor. Useful if the SBus must access the hardware registers for debugging or diagnostics. A value of 0 restarts the SPARC at the reset address.
14-0	UNUSED	unused

Table 10. Board Control SPARC Run Register Bit Definitions

Bit	S53B_	Description
15	RESET	A value of 1 resets and reinitializes the S53B1. A value of 0 has no effect. The value is not stored.
14-0	UNUSED	unused

Table 11. Board Control Reset Register Bit Definitions

RAM

The RAM contains data from the 1553 bus, memory shared by the S53B1 driver and the embedded SPARC microprocessor, and the 1553 program run by the embedded SPARC.

EEPROM

The electronically erasable programmable ROM is a 32 KB x 8 bit EEPROM mapped into the SBus slot address space at offset 0. It contains identification and initialization parameters.

In addition to its factory-set parameters, the EEPROM can also contain code for the user application. Contact Sun Microsystems for documentation and Engineering Design Team for assistance if custom PROM code programming is required.

Specifications

The S53B1 conforms to the following specifications.

MIL-STD 1553

Format	Redundant serial data bus
Modes	remote terminal (RT), bus controller (BC), bus monitor (BM), in any combination
Protocol	Command/response
Mode Codes	All
Coupling	Direct or transformer, selected by an external switch
Built-in Test	Yes
Connector	Amp 222153-3 Concentric Triax Type Three Lug

Software

Drivers for Sun OS Version 4.1 and System V Version 4 (Solaris 2.0)
10 registers, 128 KB buffer memory

SBus Compliance

Master/slave	Slave only
Transfer size	1, 2, and 4 bytes
Clock rate	25 MHz maximum
Interrupt	Level 2
Connector	Fujitsu FCN-2341PO96-GO or Honda PCS-96MD

Power

5 V at 1.2 A with no bus activity or 1.7 A with bus activity at 100%

Environmental

Temperature	Operating: 10 to 40 C Nonoperating: -20 to 60 C
Humidity	Operating: 20 to 80% noncondensing at 40 C Nonoperating: 95% noncondensing at 40 C

Physical

Occupies one standard SBus slot	
Dimensions	3.3" x 5.78" x 0.5"
Weight	6 oz.

Glossary

address	An integer from 0 to 31 specifying to which remote terminal the bus controller is sending a command or data. An address of 32 indicates a transmission broadcast to all remote terminals.
broadcast	Sending a transmission to all remote terminals on the bus, rather than the one specified by a specific address.
bus	A wire connecting a controller with one or more devices in order that commands, data, and status information can be transmitted and received among them.
bus controller	A device on a bus responsible for initiating all commands to send or receive data or status.
bus monitor	A device on a bus that can watch all the information that is transmitted or received, for diagnostic purposes.
channel	One wire that can transmit or receive information.
command word	A 16-bit word that instructs a device on the bus that it must perform some action.
coupling	A way of connecting the bus to the devices it controls.
data word	A 16-bit word that represents a value read from, or written to, a device.
device driver	The software that integrates an external device with the operating system of a host computer to which it is attached.
direct coupling	One method of coupling a device to a 1553 bus. Direct coupling is useful only if the stub is one foot long or shorter. It is unsuitable for applications requiring high reliability, as a short circuit in the device or stub can cause the bus to fail.
dual-redundant	A method of implementing redundancy using two of a specific component—in the case of the 1553 bus, using two channels.
dynamic bus control	A method of bus operation wherein responsibility for assuming the bus controller role can be passed from one device to another while the bus is operating.
EEPROM	Electrically erasable programmable read-only memory.
mode code	A command that causes devices on the bus to interpret the commands that follow in a different manner.
parity	A method of checking for errors to ensure that data is transmitted and received correctly.
primary channel	The main channel of a dual-redundant bus; channel 0 of the S53B1.
RAM	Random access memory.
redundancy	A method of ensuring robustness by including more than the required number of a specific component.
remote terminal	A device on the bus that can transmit and receive data or status only in response to a bus controller command.
secondary channel	The second channel of a dual-redundant bus; channel 1 on the S53B1.
status word	A 16-bit word specifying the status of a remote terminal.

stub	The cable between a 1553 bus and a device to which it is connected.
subaddress	An integer between 0 and 31 specifying the specific component or data location of a remote terminal.
sync	The transition within the first 3 s of a 20-s serial word, indicating the start of the word and its identifier—that is, whether it is command or data.
transformer coupling	One method of coupling a device to a 1553 bus. Transformer coupling is required if the stub is longer than one foot, or for applications requiring high reliability, as a short circuit in the device or stub cannot cause the bus to fail when transformer coupling is used.

References

MIL-STD 1553B *Military Standard Aircraft Internal Time Division Command/Response Multiplex Data Bus*, Sept. 21, 1978, available from the Department of Defense. Also available from Global Engineering Documents, (800) 854-7179.

Sun *SBus Specification B.0*, part number 800-5323-05, available from Sun Microsystems, Inc., (415) 960-1300.

Index

Numerics

S53G_USER 22

A

absolute scheduling
 for bc_auto structures 34
 address
 defined 48
 analyzing the S53B with the Commander 37
 application
 basic elements of 15
 application support routines 5

B

BC to all RTs broadcast
 example code 24
 bc_auto continuous execution
 example program 5
 bc_auto scheduling
 example program 5
 bc_auto structure 30
 absolute scheduling 34
 as circular buffer 31
 continuous double-buffered 18
 hardware scheduling 34
 memory management options for 19
 relative scheduling 34
 scheduling 33
 shared memory for 34
 bc_automemory management
 example program 5
 bc_buf structure 23
 bctest 15, 20
 bm 17, 21
 board control registers 45
 broadcast 6
 defined 48
 broadcast command received
 in status word 9
 bus
 coupling requirements 14
 defined 48
 performance 1, 6
 physical characteristics of 14
 physical components of 6, 11
 bus analysis tool 1, 37
 bus controller
 behavior of 6

 configuring device driver as 19
 defined 48
 example program 5, 15
 using Commander 37
 bus interface
 overview 1
 bus monitor 5, 6
 configuring device driver as 20
 defined 48
 error status, detecting 21
 example program 5, 17
 S53B1 transmissions only 20
 using Commander 39, 41
 busy bit
 in status word 9

C

cable 6
 length of 6, 11, 13
 resistance of 11
 tap points 11
 termination 11
 channels 1, 6
 defined 48
 interrupt in 44
 setting 19
 setting with the Commander 37
 clock rate
 specifications 47
 CMDWORD macro 30
 command registers 45
 command word 8
 defined 48
 mode code 8
 parity 8
 remote terminal address 8
 remote terminal subaddress 8
 synchronization 8
 transmit/receive 8
 word count 8
 Commander 5, 37
 as bus controller 37
 as bus monitor 39, 41
 S53B1 transmissions only 40
 as remote terminal 38
 editing data in 37
 invoking 37
 invoking with device driver name 37
 mode codes in 41

- setting the mask 39
- setting the mode 37
- commands from stdin
 - example program 5, 18
- commands to stdout
 - example program 5, 18
- connector
 - 1553B 47
 - source for 11
 - specifications 47
 - type 11
- coupling
 - and cable length 6, 11, 13
 - defined 48
 - direct 6, 11, 12
 - physical requirements for 14
 - setting 2
 - transformer 6, 11, 13

D

- data in receive buffer 44
- data registers 43
- data transfer size
 - specifications 47
- data word
 - defined 48
- debugging
 - example program 5, 17
- device driver 1, 4, 19
 - as bus controller 19
 - as bus monitor 20
 - as remote terminal 20
 - emulating an entire 1553 system 11
 - installing 3
 - loading 3
 - multiple 3
 - name of 19
 - unloading 3, 4
- diagnostics 5, 40
- DIRECT 2, 11
- direct coupling 2, 11, 12
 - defined 48
- distribution diskette 3, 4
 - contents 4
- dual-redundant
 - defined 48
- dynamic bus control 6
 - defined 48
 - in status word 9

E

- EEPROM 46
 - defined 48
- enable transmit 45
- enabling interrupts 45
- environmental
 - specifications 47
- error insertion 5, 16, 29
- error types 21
- examining mode codes
 - example program 5
- example programs 15
- EXC_RST 44

F

- file offset, UNIX 22

G

- GAP_ERROR 30

H

- hardware
 - installing 2
- hardware scheduling
 - for bc_auto structures 34
- HIWORD_ERROR 30
- humidity
 - specifications 47

I

- initializing S53B1 46
- inserting errors 5, 16, 29
- installing
 - device driver 3
 - hardware 2
 - software 3
- instrumentation bit
 - in status word 9
- INT 44
- intermessage time 5, 6, 16, 29
- interrupt
 - changing level 46
 - enabled to SPARC 45
 - level 47
 - overflow enabled 45
 - overflow reset 44
 - receive buffer ready reset 44
 - receive enabled 45
 - response time exceed enabled 45

- response time exceeded reset 44
- SBus 46
- SPARC 44, 46
- transmit buffer ready reset 44
- transmit enabled 45
- underrun enabled 45
- underrun reset 44
- ioctl 22
- ioctl's
 - for mode codes 19, 27
 - for status bits 29
 - responding to 28
 - S53S_AUTO_CONT 31
 - S53S_AUTO_OFFSET 31
 - S53S_AUTO_SIZE 30
 - S53S_AUTO_TODO 31
 - S53S_AUTO_WAITCNT 31
 - S53S_CNT 31
 - S53S_DUMP 31
 - S53S_ERR 31
 - S53S_GO 31
 - S53S_LOAD 31
 - S53S_RT_ERR 29

L

- loading
 - device driver 3
- LOWORD_ERROR 30

M

- makefile 3, 4
- masking
 - remote terminal subaddresses 20
 - remote terminals 21
- mc_buf structure 27
- memory 46
- mems53bi 19
- message
 - timing of 6
- message error
 - in status word 9
- message type
 - BC to all RTs broadcast 7
 - BC to RT transfer 7
 - broadcast mode command, data word RT receive 8
 - broadcast mode command, no data word 8
 - format 6
 - mode command, data word RT receive 8
 - mode command, data word RT transmit 8
 - mode command, no data word 7
 - RT to all other RTs broadcast 7

- RT to BC transfer 7
- RT to RT transfer 7
- MIL-STD 1553B 1
 - description of 6
 - specifications 47, 50
- MNINT_ENB 45
- mode codes 6, 10
 - constants 27
 - defined 48
 - dynamic bus control 10
 - in command word 8
 - inhibit terminal flag bit 10
 - initiate self test 10
 - override inhibit terminal flag bit 10
 - override transmitter shutdown 10
 - reset remote terminal 10
 - responses to 28
 - selected transmitter shutdown 10
 - sending with ioctl's 19, 27
 - synchronize 10
 - synchronize with data word 10, 28
 - transmit built-in test word 10, 28
 - transmit last command 10, 28
 - transmit status word 10, 28
 - transmit vector word 10, 28
 - transmitter shutdown 10
 - with Commander 41
- mode codes:override selected transmitter shutdown 10
- mode_data structure 28
- mode_sig structure 28
- monitor mask 21
- monitoring
 - bus 5
 - certain subaddresses only 21

N

- NOOP bit
 - in bc_auto structures 34
- NORESP_ERROR 30
- NORMAL
 - return type 25

O

- OPEN LOOK 37
- open system call 19
- OVER_ENB 45
- OVER_RST 44
- overrun interrupt reset 44
- overrun occurred 44
- OVERRUN 44

P

parity 44, 45
 command word 8
 error 30
 status word 9
 timing 6
 physical
 specifications 47
 pkgadd command 4
 pkgrm command 4
 power
 specifications 47
 primary channel
 defined 48
 PRTY_ERROR 30

Q

q_elem data structure 21
 macros for 21
 queue mask 20

R

RAM 46
 defined 48
 rcv1553 18
 read system call 20, 22
 failure of 22
 REC_INT_ENB 45
 receive buffer ready 44
 receive buffer ready interrupt reset 44
 redundancy 1, 6
 defined 48
 references
 MIL-STD 1553B specification 50
 SBus specification 50
 registers 42
 board control 45
 command 45
 data 43
 reset 44
 SBus addresses 42
 SPARC addresses 43
 status 44
 relative scheduling
 for bc_auto structures 34
 remote terminal
 address 6
 in command word 8
 in status word 9
 broadcasting to 6

 configuring device driver as 20
 defined 48
 example program 5, 16
 masking subaddresses 20
 monitor mask 21
 shared memory for 34
 subaddress 6
 in command word 8
 using Commander 38
 reset registers 44
 resetting S53B1 46
 response time 6
 response time exceeded 44
 response time exceeded interrupt reset 44
 return type 25
 RSPEXC 44
 RSPXCD_ENB 45
 RT to all other RTs broadcast
 example code 24
 RT to BC transfer
 example code 23
 rt_buf structure 23
 rt_rt structure 24
 rtttest 16
 RX_PRTY 44
 RX_RDY 44
 RX_RST 44
 RX_SYNC 44

S

S53B 21
 s53b_mon.c 22
 S53B1
 installing 2
 s53bi.h 29
 s53btest 16
 S53S_AUTO_CNT ioctl 31
 S53S_AUTO_OFFSET ioctl 31
 S53S_AUTO_SIZE ioctl 30
 S53S_AUTO_TODO ioctl 31
 S53S_AUTO_WAITCNT ioctl 31
 S53S_CNT ioctl 31
 S53S_DUMP ioctl 31
 S53S_ERR ioctl 31
 S53S_GO ioctl 31
 S53S_LOAD ioctl 31
 S53S_RT_ERR ioctl 29
 S53S_USER ioctl 22
 SBus 1, 2
 connector 47
 interrupting SPARC 46
 slot 2

- specifications 47
- scheduling bc_auto structures 33
 - absolute 34
 - hardware 34
 - NOOP 34
 - relative 34
- secondary channel
 - defined 48
- select system call 26
- service request
 - in status word 9
- setdebug 17
- setting the channel 19
- shared memory 34
- size
 - specifications 47
- SLOW_ERROR 30
- software
 - files included 4
 - installing 3
 - specifications 47
- SPARC 1, 5, 43
 - halt 46
 - interrupting SBus 46
 - reset 46
- specifications 47
 - clock rate 47
 - connector 47
 - data transfer size 47
 - environmental 47
 - humidity 47
 - interrupt level 47
 - MIL-STD 1553B 47, 50
 - physical 47
 - power 47
 - SBus 47, 50
 - size 47
 - software 47
 - temperature 47
 - weight 47
- specifying intermessage gap 5, 16, 29
- status bits
 - setting with ioctl's 29
- status registers 44
- status word 9
 - broadcast command received 9
 - busy bit 9
 - defined 48
 - dynamic bus control 9
 - instrumentation 9
 - message 9
 - parity 9

- remote terminal address 9
- service request 9
- subsystem flag 9
- synchronization 9
- terminal flag 9
- stdin 5, 18
- stdout 5, 18
- structures
 - bc_auto 30
 - bc_buf 23
 - mc_buf 27
 - mode_data 28
 - mode_sig 28
 - rt_buf 23
 - rt_rt 24
- stub 11
 - defined 49
 - length of 6, 13
- subaddress
 - defined 49
- subaddress mask 22, 25, 26, 27
- subsystem flag
 - in status word 9
- SunOS
 - Solaris 2.0 4
 - SVR 4 4
 - Version 4.1 3
- SYNC_ERROR 30
- synchronization 44
 - command word 8
 - defined 49
 - error 30
 - status word 9
 - timing 6
- system call
 - open 19
 - read 20, 22
 - select 26
 - write 20, 22, 25

T

- tar command 3
- temperature
 - specifications 47
- terminal flag
 - in status word 9
- testdriver 18
- testing 5, 40
- timing 6
- transferring data between remote terminals 24
- transformer 11
- transformer coupling 2, 11, 13

- defined 49
- transmit buffer ready 44
- transmit buffer ready interrupt reset 44
- transmit enable 45
- transmit/receive
 - in command word 8
- TX_ENBL 45
- TX_PRTY 45
- TX_RDY 44
- TX_RST 44

U

- UNDER_ENB 45
- UNDER_RST 44
- underrun interrupt reset 44
- underrun occurred 44
- UNDRRUN 44
- unloading
 - device driver 3, 4
- userbm.c 22
- user-defined bus monitoring
 - example program 5
- user-defined monitor 21

W

- waiting for a specific subaddress 22, 25, 26, 27
- weight
 - specifications 47
- word count
 - in command word 8
- write system call 20, 22, 25
 - failure of 22
- writing applications 19

X

- XFMR 2, 11
- XMT_INT_ENB 45
- xmt1553 18