PCI53B

PCI Bus to MIL-STD 1553B Interface

USER'S GUIDE

008-00967-02

EDT
INCORPORATED

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.

Engineering Design Team, Inc. ("EDT"), makes no warranties, express or implied, including without limitation the implied warranties of merchantibility and fitness for a particular purpose, regarding the software described in this document ("the software"). EDT does not warrant, guarantee, or make any representations regarding the use or the results of the use of the software in terms of its correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of the software is assumed by you. The exclusion of implied warranties is not permitted by some jurisdictions. The above exclusion may not apply to you.

In no event will EDT, its directors, officers, employees, or agents be liable to you for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, and the like) arising out of the use or inability to use the software even if EDT has been advised of the possibility of such damages. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. EDT's liability to you for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, tort [including negligence], product liability or otherwise), will be limited to $50.

Contents

Tables

Figures

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 PCI53B PCI Bus to MIL-STD 1553B Interface allows you to connect a Sun workstation or Windows NT / 2000 computer to a 1553B bus, or to use your host computer to emulate an entire 1553B bus system of 32 devices or fewer, including a bus controller, a bus monitor, and up to 31 remote terminals. The PCI53B has two channels for redundancy, an embedded SPARC microprocessor and 4 MB of onboard memory, and hardware support for timestamps of 1553B bus events. It includes a PCI Bus DMA master to transfer data between onboard memory and the host processor's main memory, under the control of the embedded SPARC microprocessor. It supports the full 1553B set of standard commands and subcommands.

NOTE *Engineering Design Team can customize a PCI53B to detect any standard command or subcommand as illegal if your application requires it.*

The PCI53B includes a loadable, configurable device driver and a wide variety of example applications that can be customized for many different purposes.

This document describes how to install the PCI53B bus interface, and write applications for it. It is divided into the following sections:

Installation	provides instructions about manually checking and reconfiguring software installation.
After Installing	gives instructions for verifying the installation, configuring the PCI53B, building the sample programs, uninstalling the software, and upgrading the firmware when necessary.
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 PCI53B driver.
Connector Pinout	provides a pinout diagram describing the connection from the PCI53B board to the interface (bus) cable.
Registers	describes the PCI53B hardware registers.
Specifications	lists the specifications.
Glossary	defines the terms and acronyms used in this document.
References	refers to other documents that may prove useful to you when writing applications for the PCI53B.
ioctl Parameters	This appendix describes parameters to the `p53b_ioctl` library call that applications can use to perform I / O, other than reading or writing.

Installation

Refer to the separate document, Installation Instructions, for specific instructions on how to install the PCI53B hardware and software.

During software installation, the install program will check to see whether the current configuration of the PCI 53B is compatible with the system architecture. PCI53B boards are configured at the factory to run on either Sun Sparc or Intel X/86 system architecture, and if there is a conflict, you will be instructed to reconfigure the board.

You can also do the check and the reconfiguration manually after the software installation. To check the board's compatibility with the host architecture, go to /opt/EDTpdv (Unix/Linux), or bring up the P53b utilities command shell (Windows NT/2000), and run:

```
pdb checkver
```

To update the firmware for an X/86 host, run:

```
pdb update86
```

To update the firmware for a Sun host, run:

```
pdb updatesun
```

After running the update, you must shut down the system and cycle power in order for the changes to take effect.

After Installing

After you've installed the PCI53B board and software, follow the instructions below to verify the installation. Then configure the board as a 1553A device, if necessary (the default is a 1553B), and build the sample programs.

Configuring the PCI53B

You can configure the PCI Bus to MIL-STD 1553B Interface to perform as a general-purpose 1553B interface with or without the broadcast address, or as a 1553A interface. The 1553B standard normally reserves bus address 31 to indicate a broadcast transaction, leaving addresses 0-30 as remote terminal (RT) addresses. The no-broadcast option does not reserve address 31 (all 32 bus addresses are available for remote terminals) but sacrifices the ability to broadcast to all RTs at once.

The default is a general-purpose 1553B interface with broadcast enabled. If this is acceptable, you need not configure the device.

To configure the device:

1. If you're on a Windows NT/2000 platform, run *P53b Utilities*.
2. At the DOS prompt (Windows NT/2000) or the shell prompt (Solaris or Linux), enter the command:

   ```
   embselect
   ```

3. Enter **1** for the firmware for the general-purpose 1553B interface; **2** for the 1553A; or **3** for the no-broadcast option.

Other selections may be added for custom firmware options.

Verifying the Installation

To verify that installation was successful and that the PCI53B is operating correctly, run the *p53btest* board test as follows.

For Windows NT/2000:

1. Double-click on the *P53b Utilities* shortcut to open the PCI53B utility window.
2. Run the test:

```
p53btest -l 1
```

For Solaris or Linux:

1. Open a command-line window and `cd` to `/opt/EDTp53b`.
2. Run the test:

```
./p53btest -l 1
```

The -l 1 option specifies the number of loops of the test to run; here we specify 1 iteration.

Building the Sample Programs

Solaris or Linux Systems

To build any of the example programs on Solaris or Linux systems, enter the command:

```
make file
```

where *file* is the name of the example program you wish to install.

To build and install all the example programs, enter the command:

```
make
```

All example programs display a message that explains their usage when you enter their names without parameters.

Windows NT/2000 Systems

To build any of the example programs on Windows NT/2000 systems:

1. Double-click the P53b Utilities icon to open the utility window.
2. Enter the command:

   `make file`

where `file` is the name of the example program you wish to build.

To build and install all the example programs, simply enter the command:

```
make
```

All example programs display a message that explains their usage when you enter their names without parameters.

You can also build the sample programs by setting up your own projects in Windows Visual C++.

Uninstalling

Solaris or Linux Systems

To remove the PCI53B driver on Solaris or Linux systems:

1. Become root or superuser.
2. Enter:

   ```
   pkgrm EDTp53b
   ```

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

Windows NT/2000 Systems

To remove the PCI53B toolkit on Windows NT/2000 systems, use the Windows NT/2000 Add/Remove utility. For further details, consult your Windows NT/2000 documentation.

Obtaining Software Updates

You can always get the most recent update of the software from our web site, *http://www.edt.com*. See the document titled *Contact Us*.

Upgrading the Firmware

After upgrading to a new device driver, it may sometimes also be necessary to upgrade the PCI interface PROM. If so, the *readme* file will say so.

To use the following commands, first get a command-line prompt in the EDT directory:

- If you're using Windows NT/2000, double-click on the *P53B Utilities* shortcut.
- If you're using Solaris or Linux, `cd` to `/opt/EDTp53b`.

Checking the Firmware Level

To check the firmware level, use the pdb utility:

1. At the command-line prompt, enter:

   ```
   pdb
   ```

2. At the ":" prompt, enter:

   ```
   fv
   ```

You should see the following. If not, contact EDT for a firmware update.

```
Checking xilinx rev
Version 3 pci xilinx
sector 6: p03z.lca b  4013EPQ240 c  99/03/09 d  16:40:03 e
sector 7: p03z.lca b  4013EPQ240 c  99/03/09 d  16:40:03 e
```

3. To exit from `pdb`, enter

   ```
   q
   ```

Applying an Update

To update the firmware use enter:

```
pdb updateall
```

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 comprises the following elements:

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

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 normally reserved for broadcasting a transmission to all RTs. Consequently, no more than 31 RTs are allowed on a 1553 bus, unless the bus is configured in "no-broadcast" mode. 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 under the 1553B specification, only one subaddress (0) is reserved under the 1553A specification. Each of the remaining 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 (redundant) pair of controlled-impedance differential cables. These cables are 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.

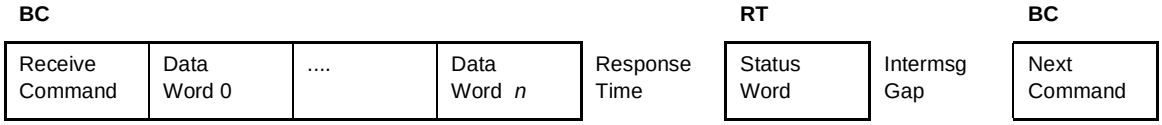
The PCI53B can be configured to operate as any or all of the 31 possible remote terminals (or 32 in no-broadcast mode), the bus controller, and a bus monitor, all independently and concurrently.

Message Types

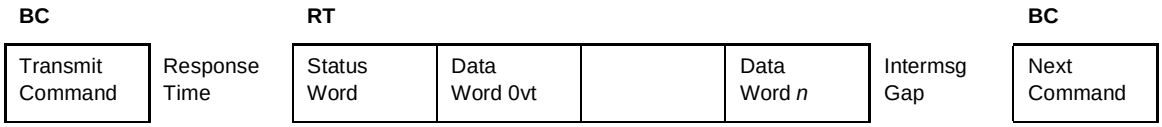
The 1553 bus uses ten message types, whose format is shown below. Each box represents 20 µs on the bus: 3 µs for synchronization and word identification (whether the word represents data or a command), 16 µs

to transmit the command or data, and 1 µs for parity. Response time is 4–12 µs, and intermessage time is any amount of time longer than 4 µs.

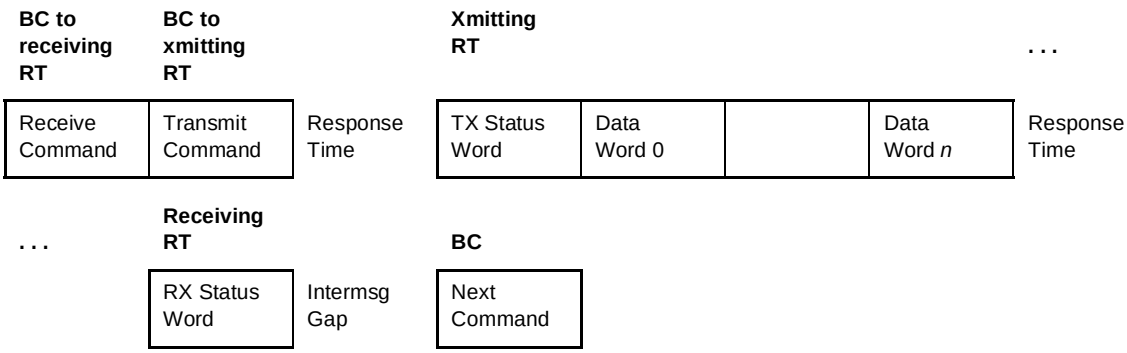
BC to RT Transfer



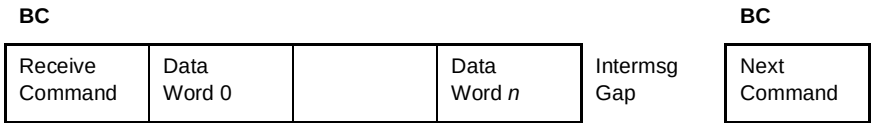
RT to BC Transfer



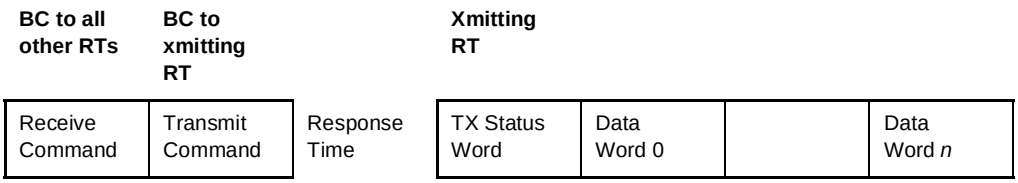
RT to RT Transfer



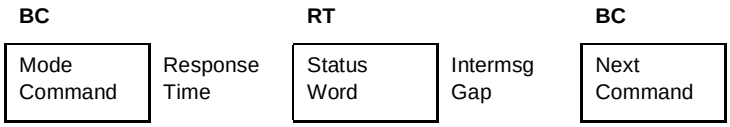
BC to All RTs Broadcast



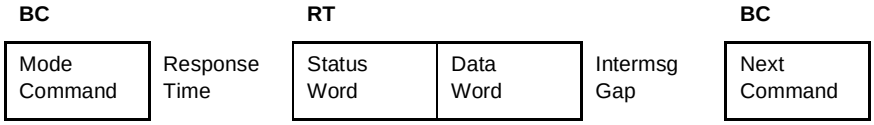
RT to All Other RTs Broadcast



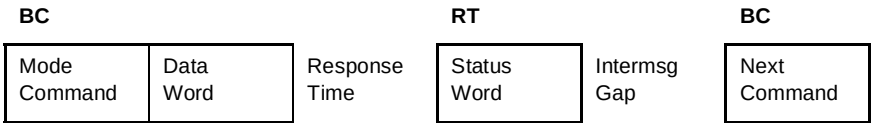
Mode Command—No Data Word



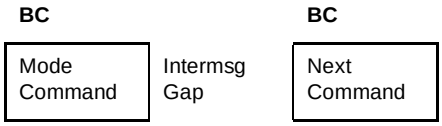
Mode Command—Data Word RT Transmit



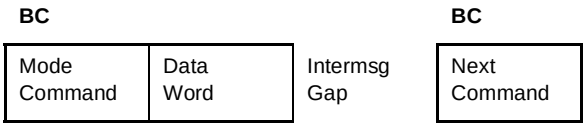
Mode Command—Data Word RT Receive



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**
RTADD4	RTADD3	RTADD2	RTADD1	RTADD0	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 PCI53B 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	PCI53B Response (see page 35 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-Ω twinaxial cable—a 100% shielded cable with two signal wires—terminated at both ends with 78-Ω resistors. At each tap point for a subsystem, a transformer and another twinaxial cable—the stub—is connected to the subsystem.

Resistance can vary between 70–85 Ω, 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 Trompeter BJ76. A good source for small quantities of mating connectors is Trompeter Electronics of Westlake Village, CA, (818) 707-2020, *http://www.trompeter.com*. 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 PCI53B to emulate an entire 1553 system, including the bus controller, remote terminals, and bus monitors, place a 35–43 Ω termination resistor on both outputs. For example, a 40 Ω 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 PCI53B 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 Ω cable.

External Bus Coupling

A good source of couplers and terminators is Technitrol of Philadelphia, PA, (215) 426-9105, *http://www.technitrol.com*.

Have the software select 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.

Use the supplied *setdebug* utility to change between a direct-coupled connection and a transformer-coupled connection. On Windows, double-click the *P53b Utilities* icon to open the command window.

```
setdebug -C direct
```

where *direct* is 1 for direct-coupled or 0 for transformer-coupled.

Direct coupling is shown in the figure below.

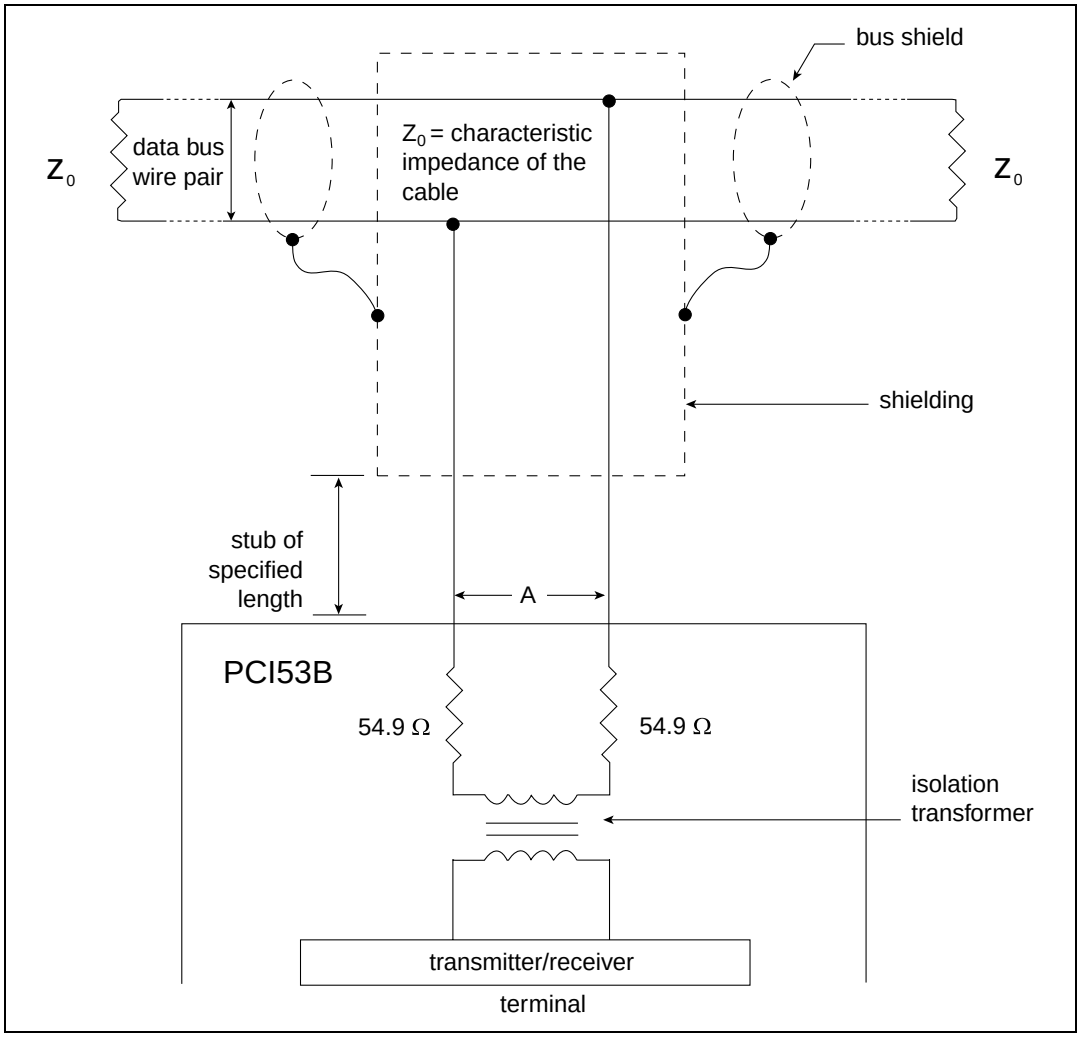


Figure 1. Direct Coupling

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.

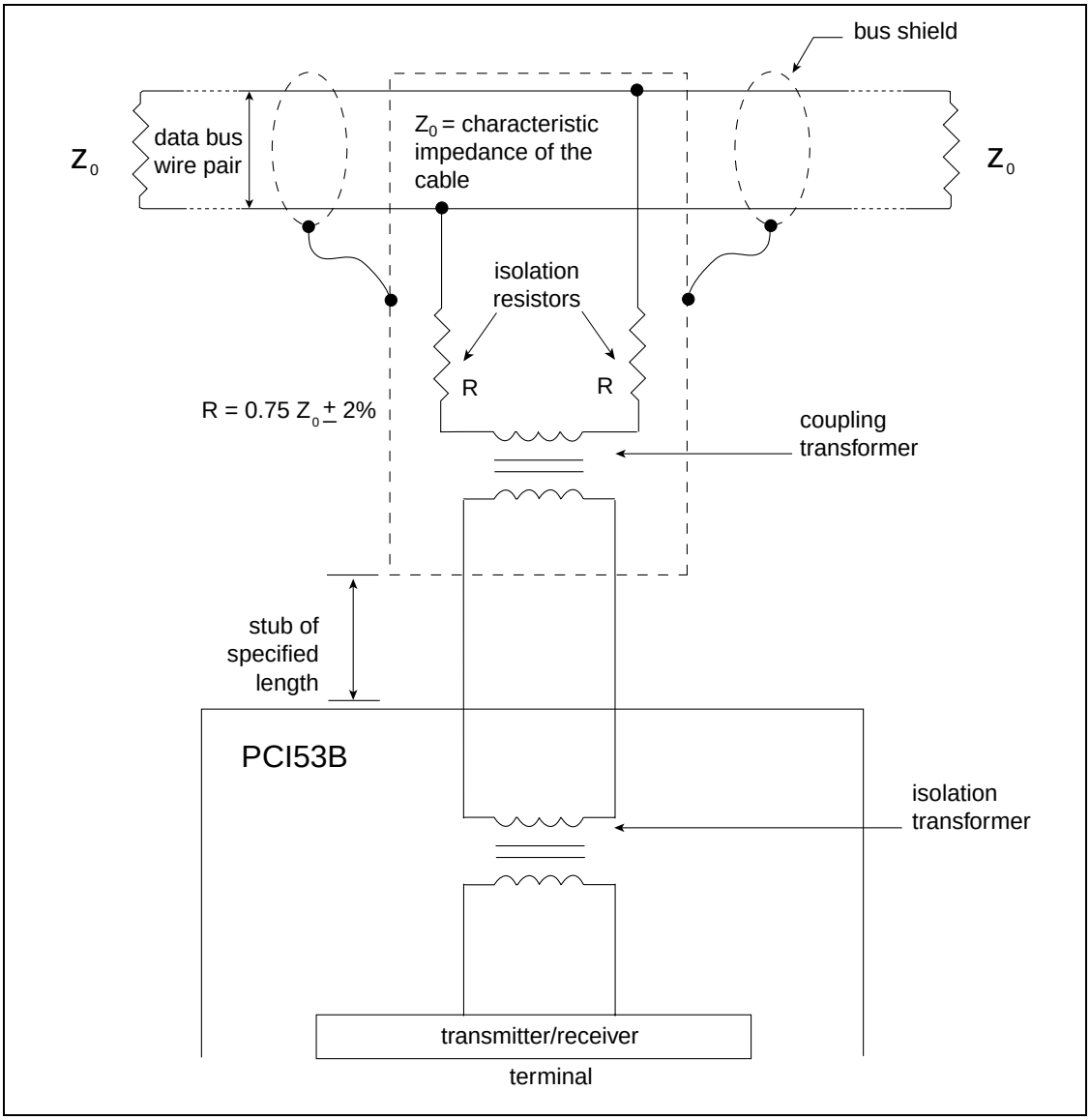


Figure 2. Transformer Coupling

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

Parameter	MIL-STD-1553B Requirement
Transmission line	
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 Ω 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
Cable Coupling	
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 Ω 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% Ω

Table 2. Data Bus and Coupling Requirements

Writing Applications

The PCI53B includes a library of routines to use in your applications, and various example applications. These are described below.

A basic PCI53B application has the following elements:

- a `#include "edtinc.h"` statement
- a `p53b_open( )` library call with the unit number and bus element; for example:

  `p53b_open(0, RT_7)`

  Bus elements can be one of `BUS_CONTROLLER`, `BUS_MONITOR`, or `RT_0`–`RT_30`.

NOTE *If you've configured the PCI53B to run in no-broadcast mode, as described in "Building the Sample Programs" on page 3,* `RT31` *is also a legal bus element.*

- `p53b_read( )` and `p53b_write( )` library calls to transfer data between bus elements.
- `p53b_ioctl( )` library call to configure, query, or control the device.

  On Solaris or Linux platforms, `p53b_ioctl( )` makes *ioctl* system calls. On Windows NT/2000 platforms, it performs comparable functions.
- a `p53b_close( )` library call to close the device during execution, if necessary. The device is automatically closed when the program exits.

PCI53B Library Routines

The following library routines are included with the PCI53B software.

Routine	Description
p53b_open	Opens the PCI53B for application access.
p53b_rtopen_notactive	Opens a PCI53B RT for application access with the RT initially disabled.
p53b_close	Terminates access to the PCI53B and releases resources.
p53b_rtactive	Enables or disables a PCI53B RT.
p53b_read	Single, application-level buffer read from the PCI53B for BC and RTs.
p53_bm_read	Special read for BM.
p53b_write	Single, application-level buffer write to the PCI53B.
p53b_ioctl	Performs an *ioctl* operation on the PCI53B.
p53b_perror	Returns a system message in case of error.
p53b_msleep	Suspends execution of the application for the specified number of milliseconds.
P53G_RT_WAIT	Suspends program activity until an RT gets a send or receive command on a specific subaddress.

Table 3. PCI53B Library Routines

p53b_open

Description

Opens the specified PCI53B bus element device, and sets up the device handle.

Syntax

```
#include "edtinc.h"

EdtDev p53b_open(int unit, int BUS_EL);
```

Arguments

unit	specifies the device unit number
BUS_EL	The bus element to open. One of: BUS_CONTROLLER BUS_MONITOR RT_0–RT_30

NOTE *If you've configured the PCI53B to run as a 1553A (in no-broadcast mode), as described in "Building the Sample Programs" on page 3,* `RT_31` *is also a legal bus element.*

Return

A handle of type `(EdtDev *)`, or `NULL` if error. (The structure definition for `(EdtDev *)` is included in *edtinc.h*.) If an error occurs, call `p53b_perror()` for the system error message. The device name for the PCI53B is "p53b". Once opened, use the device handle to perform I/O using `p53b_read()`, `p53b_write()`, `p53b_ioctl()`, and other input-output library calls.

p53b_rtopen_notactive

Description

Opens the specified PCI53B remote terminal device initially disabled, and sets up the device handle. To enable the RT after configuring it, use `p53b_rtactive()`.

Syntax

```
#include "edtinc.h"

EdtDev p53b_open(int unit, int BUS_EL);
```

Arguments

unit	specifies the device unit number
BUS_EL	The bus element to open. One of: RT_0–RT_30

NOTE *If you've configured the PCI53B to run as a 1553A (in no-broadcast mode), as described in "Building the Sample Programs" on page 3, `RT_31` is also a legal bus element.*

Return

A handle of type `(EdtDev *)`, or `NULL` if error. (The structure definition for `(EdtDev *)` is included in *edtinc.h*.) If an error occurs, call `p53b_perror()` for the system error message. The device name for the PCI53B is "p53b". Once opened, use the device handle to perform I/O using `p53b_read()`, `p53b_write()`, `p53b_ioctl()`, `p53b_rtactive()`, and other input-output library calls.

p53b_close

Description

Shuts down all pending I/O operations, closes the device and frees all driver resources.

Syntax

```
#include "edtinc.h"

int p53b_close(EdtDev *p53b_p);
```

Arguments

p53b_p	PCI53B device handle returned from `edt_open`.

Return

0 on success; –1 on error. If an error occurs, call `p53b_perror()` to get the system error message.

p53b_rtactive

Description

Activates (enables) or deactivates (disables) an RT. This routine is useful to activate an RT after calling `p53b_rtopen_notactive()` to open the RT without activating it initially.

Syntax

```
#include "edtinc.h"
int p53b_rtactive(EdtDev *p53b_p, int active);
```

Arguments

p53b_p	PCI53B device handle returned from `edt_open` or `p53b_rtopen_notactive`.
active	1 = activate the device; 0 = deactivate it.

Return

0 on success; –1 on error. If an error occurs, call `p53b_perror()` to get the system error message.

p53b_read

Description

Performs a read on the PCI53B. For those on UNIX systems, the UNIX 2 GB file offset bug is avoided during large amounts of input or output, that is, reading past 2^{31} bytes does not fail. This call is not multibuffering, and no transfer is active when it completes.

Syntax

```
#include "edtinc.h"
int p53b_read(EdtDev *p53b_p, void *buf, int size);
```

Arguments

p53b_p	PCI53B device handle returned from `p53b_open`
buf	address of buffer to read into
size	size of read in bytes

Return

If successful, it always returns the size in bytes that was passed in; –1 is returned in case of error. Call `edt_perror()` to get the system error message.

p53b_bm_read

Description

Performs a read on the bus monitor. Always use this option if you use a bus monitor. p53b_read uses a constant size when returning information; p53b_bm_read uses varying sizes when returning information, and only returns what is available from the bus monitor.

Syntax

```
#include "edtinc.h"
int p53b_bm_read(EdtDev *p53b_p, void *buf, int size);
```

Arguments

p53b_p	PCI53B device handle returned from `p53b_open`
buf	address of buffer to read into
size	size of read in bytes

Return

The number of bytes available from the Bus Monitor*; –1 is returned in case of error. Call `edt_perror()` to get the system error message.

* The Bus Monitor fills the buffer with q_elem structures. The return value is the number of q_elem structures multiplied by the number of bytes per q_elem structure.

p53b_write

Description

Perform a write on the PCI53B. For those on UNIX systems, the UNIX 2 GB file offset bug is avoided during large amounts of input or output; that is, writing past 2^{31} does not fail. This call is not multibuffering, and no transfer is active when it completes.

Syntax

```
#include "edtinc.h"
int p53b_write(EdtDev *p53b_p, void *buf, int size);
```

Arguments

p53b_p	PCI53B device handle returned from `p53b_open`
buf	address of buffer to write from
size	size of write in bytes

Return

The number of bytes successfully transferred; –1 is returned in case of error. Call `edt_perror()` to get the system error message.

p53b_ioctl

Description

Performs the specified input-output control operation on the open device.

Syntax

```
#include "edtinc.h"
int p53b_write(EdtDev *p53b_p, int ioctl_cmd, void* arg);
```

Arguments

p53b_p	PCI53B device handle returned from `p53b_open`
ioctl_cmd	a #define from *p53b.h*
arg	a command-dependent argument

Return

0 on success; –1 on error. If an error occurs, call `p53b_perror()` to get the system error message.

p53b_perror

Description

Formats and prints a system error.

Syntax

```
#include "edtinc.h"
void
p53b_perror(char *errstr)
```

Arguments

errstr	Error string to include in the printed error output.

Return

No return value.

p53b_msleep

Description

Suspends execution of the application for the specified number of milliseconds.

Syntax

```
#include "edtinc.h"
void
p53b_msleep(int msecs)
```

Arguments

msecs	The number of milliseconds to suspend execution of the application.

Return

No return value.

P53G_RT_WAIT

Description

Suspends program activity until a remote terminal gets a send or receive command on a specific subaddress. The P53G_RT_WAIT command accepts a pointer to a 'struct rt_wait.' This structure contains three receive and three transmit masks to indicate which subaddresses you want to target. Masks are declared as unsigned 32-bit integers.

Syntax

```
u_int   rcvmask;   /* subaddresses to target */
u_int   xmtmask;
u_int   rcvwait;   /* subaddresses you want to wait for to complete as a group */
u_int   xmtwait;
u_int   actualrcv; /* what occurred; populated by the P53B driver */
u_int   actualxmt;
```

Bits in these masks are arranged so that the least significant bit corresponds to subadddress 0, and the most significant bit corresponds to subaddress 31. If you want to target a subaddress 23, for example, set the bit in the rcvmask as follows:

```
struct rt_wait rtw = {0, 0, 0, 0, 0, 0};
int sa = 23;
rtw.rcvmask|=(1<<(sa - 1));
```

Or turn off the bit:

```
rtw.rcvmask &= ~(1<<(sa - 1));
```

Set the bits in rcvmask and xmtmask for the subaddresses you want to target. P53G_RT_WAIT will block until one of these subaddresses gets a receive (rcvmask) or transmit (smtmask) command from the bus controller.

If you want to wait for a group of subaddresses to change, set the bits in rcvwait and xmtwait. P53G_RT_WAIT will blcok until all these subaddresses get transmit or receive commands.

When P53G_RT_WAIT returns, actualrcv and actualxmt will contain bits set to those subaddresses that have changed. The following can be called to update the rt_buf structures or send new data to be transmitted:

```
p53b_ioctl(p53b_p, P53G_RT_RCVUPDT, &rt_buf);
p53b_ioctl(p53b_p, P53G_RT_XMTUPDT, &rt_buf);
```

As a special case, rcvmask and xmtmask can be set to zero and P53G_RT_WAIT can be called to check actualrcv and actualxmt for changes without waiting.

See rttest.c for example usage.

Return

The return value from *p53b_ioctl*; zero is returned if successful, -1 is returned in case of error. Call edt_msg_perror() to get the system error message.

Example Applications

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

p53btest

Demonstrates all functions of the PCI53B device driver, including error insertion and specifying the intermessage gap.

Usage `p53btest [-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 PCI53B 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 µsec for no response.
`-p`	Pause after an error.

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

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 PCI53B 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 PCI53B 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.
`-q`	Set the RTs not to queue data received. Incoming data then overwrites any previous values and the value returned by a read is simply the last received.
`-X m`	Sets the transmit-queue mask to *m*, a 32-bit hexadecimal number. The default is 0. For each RT whose corresponding bit is set, the RT will queue data to be transmitted. If the corresponding bit is clear, the RT will not internally queue data to be transmitted.

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.

bm

Demonstrates monitoring of all data on the 1553 bus or data transmitted by the specified PCI53B 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 PCI53B 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 *pci53bi.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.
`-p`	Generates output that is more suitable for automated parsing.

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

setdebug

Sets the debugging level and the interrupt debug level.

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

Arguments

`-u n` Set which PCI53B board to use, in case more than one board is installed. The default is 0.

`-d n` Set the debug level. Valid values are:
0 no debugging
1 enables verbose mode when you invoke *modstat(8)*
2 traces start and end of routines

`-i n` Set the interrupt debug level. Valid values are:
0 no debugging
1 show interrupts as bus operates

`-r` Reports current debug level.

-C *n* Configure in direct-coupled mode. Cleared by system reboot; must be rerun at boot time to initialize the PCI53B to direct-coupled mode. Valid values are:
0 transformer-coupled (This is the default.)
1 direct-coupled

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 PCI53B 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 PCI53B 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 PCI53B 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.

Notes See page 38 for further details about the `bc_auto` programming interface.

mem_pci53bi

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_pci53bi [-u n] [-r] [-f] [-s n] [-b n]`

Arguments

`-u n`	Set which PCI53B 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 thenumber of elements in `bc_auto` array to allocate in the driver.

Opening The PCI53B Driver

For all 1553B modes, the initial access to the PCI53B driver is done with the `p53b_open( )` library call, specifying the unit number and the type of bus element you want to open.

Valid bus element types are BUS_CONTROLLER, BUS_MONITOR, or RT_0–RT_30, allowing different applications to open the same board up to 32 times simultaneously. (RT_31 is also a valid bus element type if you've configured your board using the 1553A firmware, as described on page 3.) By specifying different bus element types 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 PCI53B 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 PCI53B

The driver provides access to bus controller, remote terminal, and bus monitor modes, using `p53b_read` and `p53b_write` library calls. Mode codes can be sent using `p53b_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 PCI53B in bus controller mode:

```
EdtDev *p53b_bc = p53b_open(0, BUS_CONTROLLER);
```

In bus controller mode, you can set the PCI53B to use the desired channel. The following example sets the PCI53B to use the primary channel (channel 0):

```
u_short bus = 0; /* 0 = primary channel, 1 = secondary */
p53b_ioctl(p53b_bc, P53S_BUS, &bus);
```

The `p53b_read` library 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 `p53b_write` library 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.

If you want to use, dynamic bus control, see Using Dynamic Bus Control on page 36.

In Remote Terminal Mode

The following example shows how to place the PCI53B in remote terminal mode. It also sets the remote terminal address to 1.

```
EdtDev *p53b_rt1 = p53b_open(0, RT_1);
```

The `p53b_read` library call returns data that has arrived at the PCI53B board in response to a bus controller sending data with a BC Transmit command. This is referred to as RT Receive. The `p53b_write` library call loads data on the PCI53B 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 PCI53B 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;
p53b_ioctl(p53b_p, P53S_QUEUEMSK, &mask);
```

Transmit Queue Mask

The P53S_XMTQUEUEMSK ioctl command takes the address of an unsigned integer, and causes the P53B driver to queue up to 1024 RT write operations (like the rcvmask queue). The RT data is then automatically updated from the transmit queue immediately following a BC transmit command to RT.

Subaddress word count must be 16 or greater for reliable operation.

The P53S_XMTQUEUESRQ ioctl command also takes the address of an unsigned integer and causes the RT's service request (srq) bit to be inserted automatically when p53b_write() places data in the subaddress specified by the srq mask. The srq bit will be cleared when the BC asks for a transmit of the last p53b_write() queued.

See P53S_XMTQUEUEMSK and P53S_XMTQUEUESRQ in the sample program rttest.c.

In Bus Monitor Mode

The following example shows how to place the PCI53B in bus monitor mode, in which it monitors all data on the bus:

```
EdtDev *p53b_bm = p53b_open(0, BUS_MONITOR);
```

The `p53b_read` library 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; /* shows errors detected by board; if no errors, shows
0*/
      u_short word ; /* the word itself */
      u_int   time ; /* timestamp (see Timestamp on page 30 for details)*/
    } ;
```

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 *pci53bi.h* and are as follows:

P53B_PARITY	bad parity
P53B_HIWORD	too many words received
P53B_LOWORD	too few words received, or timeout
P53B_NONCONT	noncontinuous error detected
P53B_BADCV	manchester code violation (see 1553B specification)
P53B_DATAMATCH	mismatch between sent or received
P53B_SYNCERR	unexpected sync bits

Timestamp

The timestamp of the q_elem structure is set to the lower 32 bits of the 64-bit embedded controller timestamp. The 32-bit setting should be sufficient unless you need to use absolute time, which needs 64 bits. To use absolute time:

1. Call `p53b_ioctl (p53b_p,P53G_TIME_HI,&time_hi);` where time_hi is a u_int.
2. Shift time_hi 32 bits to the left.
3. Combine the two 32-bit sections with the bitwise OR operator.

See checkp53b.c for an example.

Clock Synchronization

To synchronize your controller clock with your system clock, run checkp53b -s, or use that code to run in your program. You can set your system clock to an atomic clock at *http://www.boulder.nist.gov/timefreq/service/its.htm*.

After you synchronize the p53b to your system or host clock, you can modify bm.c to print the actual time as follows:

Set zerotime to 0 instead of reltime on line 617. This change alone will provide the lower 32 bits of microseconds relative to 01/01/1970. To get the high 32 bits, use the p53b_ioctl() above. Using code in checkp53b.c, you can then display the 64-bit actual time in any of these formats.

Read and Write Data Structures

In bus controller and remote terminal modes, the `p53b_read` and `p53b_write` library 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 *pci53bi.h*, are used to collect data received and sent to the PCI53B using `p53b_read` and `p53b_write` library calls.

```
/*
 * buffer for read/write library 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 library 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 "IOCTL Parameters" on page 35), 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;

   p53b_write(p53b_p, &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;
p53b_read (p53b_p, 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;
p53b_write (p53b_p, 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) */
p53b_ioctl (p53b_p, PCI53S_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) */
p53b_ioctl (p53b_p, PCI53S_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 PCI53B board. The `mask` field contains bits for subaddresses that the application expects the bus controller to request before `p53b_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 `p53b_write` library call 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.)

```
NORMAL1  /* all subaddresses satisfied on write */
UNEXPECTED4 /*2nd 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 `p53b_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;
p53b_write (p53b_p, &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;
p53b_read (p53b_p, &rt_buf, sizeof(rt_buf));
```

Sending Mode Codes

A bus controller can send a mode code with the PCI53S_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 send 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 *pci53bi.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 */
p53b_ioctl (p53b_p, P53S_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 PCI53B can respond to any mode code with a possible application interrupt using P53B_MODE_SIG *ioctls*. In addition, the PCI53B responds to certain mode codes as follows:

Mode Code	PCI53B 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 P53B_LOAD_TVW.
Synchronize with data word	To be returned by the P53B_GET_MODECODE *ioctl*
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 4. 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 P53B_MODE_SIG *ioctl*. This *ioctl* takes a pointer to a `mode_sig` structure.

```
struct mode_sig
{
        u_int    mask ; /* mask of the requested mode codes */
        u_short  signal ; /* signal to be sent on mode code */
}
```

After receiving the signal (or at any time), the application can execute an p53g_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_short  data ;            /* optional data */
        u_short  code ;            /* mode code received */
}
```

The *ioctl* **p53g_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* **p53g_MODEQ** allows you to get the entire queue of mode codes for a remote terminal. It takes a pointer to the `mode_data` struct.

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 *p53bi.h*. The service request bit is unique, however; specific *ioctl*s cause the bit to be cleared pending a specific event. The **P53B_SRQ_X** clears the service request bit on the next receipt of a bus controller transmit request. The **P53B_SRQ_V** clears the service request bit on the next receipt of a Transmit Vector Word mode code.

IOCTL Parameters

ioctl parameters are defined in *p53bi.h*. Relevant parameters are documented in "Appendix A ioctl( ) Parameters" on page 51. Applications can use them to access the device driver. See example programs for details on their uses. Use `p53b_ioctl` to make *ioctl* calls.

Using Dynamic Bus Control

Use dynamic bus control to change which bus device is your controller. Once a new controller is established, the old controller should become a remote terminal. To send a dynamic bus control mode code to to a remote terminal, you must first ensure the remote terminal can accept controller status. To set a remote terminal that is attached to this PCI53B board to accept controller status, perform the following:

```
u_short save_status;
p53b_ioctl(p53b_p, P53G_STATUS, &save_status);
save_status |= P53B_BUS_ACPT;
p53b_ioctl(p53b_p, P53S_STATUS, &save_status);
```

To send a dynamic bus control mode code to a selected remote terminal, perform the following:

```
struct mc_buf mc;
mc.code = MC_DBC;
mc.rtaddr = selected_rt;
mc.status = mc.data = 0;
p53b_ioctl(p53b_p, P53S_MODECODE, &mc);
```

See sample program p53b_dbc.c for more details.

Specifying Error Insertion and Intermessage Gap for System Tests

In order to allow you to conduct realistic tests of your system, the PCI53B allows you to insert errors and specify intermessage gap times. The intermessage gap is the number of µs between commands. In bus controller mode, use the bc_auto structure (described on page36) 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 µs does not change the driver behavior.

In remote terminal mode, use the **P53S_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.

P53B_ 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_ERROR	RT only	The RT sends fewer words than requested.
NORESP_ERROR	RT only	The RT does not respond.
NORESP0_ERROR	RT only	The RT does not respond on bus 0.
NORESP1_ERROR	RT only	The RT does not respond on bus 1.
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 5. Error Codes

bc_auto Structure

The `bc_auto` structure specified in *p53b.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 µseconds and sends the next command without the host computer's participation.

The `bc_auto` structure contains the following fields:

`cmd = CMDWORD(rt,sa,wc,tr)`
: The command to send. The macro `CMDWORD` builds the command using the specified remote terminal, subaddress, word count, and transmit bit.

`cmd2`
: Set this optional field to initiatea remote terminal-to-remote terminal transfer. Otherwise set to zero.

`status`
: The status returned by the remote terminal.

`status2`
: Optional second status word returned from a remote terminal-to-remote terminal transfer. Otherwise set to zero.

`waittime`
: The number of µseconds to wait before issuing the next command. (Specifying a waittime of less than 20 µs does not change driver behavior.)

`error`
: Specifies the error to insert. A zero issues no error.`data`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 `bc_auto` structures, you must load the array and start it executing.

Use the following parameters to `p53b_ioctl`:

1. Specify the number of elements the array contains with **P53S_AUTO_SIZE**.
2. Specify the number of commands to execute with **P53S_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 **P53S_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 **P53S_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 **P53S_AUTO_TODO**, execution will continue until you send the ioctl parameter **P53S_AUTO_STOP**.

3. Specify the address of the array with **P53S_AUTO_LOAD**. **P53S_AUTO_LOAD** copies the data from the application to the start of the array in the PCI53B.
4. Start execution with **P53S_AUTO_GO**.

Other parameters specify optional behavior:

- If the application must wait until certain commands have all been issued before resuming, specify that with **P53G_AUTO_WAITCNT**. The third parameter is the address of an unsigned integer specifying an absolute count of bc_auto commands. 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 **P53S_AUTO_CNT**.
- Specify the number of commands to continue executing (after the previous set specified by **P53S_AUTO_TODO** or **P53S_AUTO_CONT**, with **P53S_AUTO_CONT**. The third parameter is the address of an unsigned integer specifying an absolute count of bc_auto commands. 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 **P53G_AUTO_ERR**.
- Specify an offset into the `bc_auto` array using **P53S_AUTO_OFFSET**. The argument to this parameter is the offset into the array at which to start loading data; run this before **P53S_AUTO_LOAD** to modify where the load occurs (or the unload using **P53S_AUTO_DUMP**). This is useful for double-buffering—you can load half of an array and cause the PCI53B 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 PCI53B has finished with commands in structure locations 50–99, you can start reloading at location 50 while the PCI53B processes commands in locations 0–49. See the example program *testdriver.c* (documented on page 27) for an example of this use.

The following example places the PCI53B in bus controller mode and loops through the list of commands three times, inserting an intermessage gap of 1000 µs (1 ms).

```
p53b_p = p53b_open(unit, BUS_CONTROLLER);
{ if (p53b_p == NULL)
p53b_perror (“p53b_open”);
exit(1);
```

```
}

   /*
    * 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 ;
   p53b_ioctl(p53b_p,P53S_AUTO_SIZE,&size) ;
   /* set number of autotest items to execute */
   todo = 30 ;
   p53b_ioctl(p53b_p,P53S_AUTO_TODO,&todo) ;
   /* load it */
   addr = (u_int)testbuf ;
   p53b_ioctl(p53b_p,P53S_AUTO_LOAD,&addr) ;
   /* start executing */
   p53b_ioctl(p53b_p,P53S_AUTO_GO,0) ;
```

The following example places the PCI53B in bus controller mode and inserts a gap error in the command transmitted.

```
u_short mode = P53B_BC;
struct bc_auto testbuf[10];
p53b_ioctl (p53b_p, P53S_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 = P53B_GAP_ERROR ;
/* set size, load, and go */
size = 1 ;
p53b_ioctl(p53b_p,P53S_AUTO_SIZE,&size) ;
/* set number of autotest items to execute */
todo = 1 ;
p53b_ioctl(p53b_p,P53S_AUTO_TODO,&todo) ;
/* load it */
addr = (u_int)testbuf ;
p53b_ioctl(p53b_p,P53S_AUTO_LOAD,&addr) ;
/* start it */
p53b_ioctl(p53b_p,P53S_AUTO_GO,0) ;
```

The following example places the PCI53B 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 = P53B_RT;
u_short addr = 1;
u_short rt_err = P53B_PRTY_ERROR;
p53b_ioctl (p53b_p, P53S_MODE, &mode);
p53b_ioctl (p53b_p, P53S_MYRTADDR, &addr);
p53b_ioctl(p53b_p,P53S_RT_ERR,&rt_err);
```

See *p53btest.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 slect the scheduling type. When these bits are all 0, the structure is scheduled as usual. The bits are defined in *p53bi.h* as follows:

```
#define P53_SCHED_NRM           0x0000
#define P53_SCHED_ABS           0x1000
#define P53_SCHED_REL           0x2000
#define P53_SCHED_HW            0x3000
#define P53_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 PCI53B 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 PCI53B. 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 PCI53B waits for the time specified by the waittime element, and then executes the next command.

The following example initializes the PCI53B clock and then sets the alarm for thirty seconds later.

```
{
        struct timeval tm;

        /* Initialize the absolute timer on the p53bi */
        gettimeofday(&tm);
        p53b_ioctl(p53b_p, P53S_TIMEVAL, &tm);

        /* Set the absolute variable to current time + 30 seconds */
        tm.tv_sec += 30 ;
        p53b_ioctl(p53b_p, P53S_TIMEABS, &tm);
}
```

Relative Scheduling

Relative scheduling affects the length of the intermessage gap. You can use relative scheduling by setting the SCHED bits to P53_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 P53_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.

Connector Pinout

The PCI53B board uses a 9-pin male D shell connector, AMP part number 748875-1. Most applications will not require this connector.

The following pinout diagram describes the connection from the PCI53B board to the cable.

Pin	Signal	Description
1	VCC	output, fused 5 V for external transceivers, maximum 500 mA
2	RXD	serial debug port input from onboard SPARC, Channel A RS-232 receive
3	TXD	serial debug port output from onboard SPARC, Channel A RS-232 transmit
4	IRIG B	input, analog time signal from GPS, to synchronize the internal timebase counter (not implemented)
5	GND	logic ground
6 7	SYNC+ SYNC–	in/out, RS-422 differential pair to synchronize timebase counters among multiple PCI53B boards (independent of an IRIG B time source)
8	SPARE+	input, Rs-422 differential pair
9	SPARE–	

Table 6. Connector Pinout

Registers

The PCI53B has two memory spaces: the memory-mapped registers and the configuration space. Expansion ROM and I/O space are not implemented. Applications can access the PCI53B registers through the library routines provided.

The information in this section is provided for completeness. Most users will not need this level of detail.

Configuration Space

The configuration space is a 64-byte portion of memory required to configure the PCI Local Bus and to handle errors. Its structure is specified by the PCI Local Bus specification. The structure as implemented for the PCI53B is as shown in Figure 3 and described below.

Address	Bits 31 16	15 0
0x00	Device ID = 0x20	Vendor ID = 0x123D
0x04	Status *(see below)*	Command *(see below)*
0x08	Class Code = 0x088000	Revision ID = 0 *(will be updated)*
0x0C	BIST = 0x00 / Header Type= 0x00	Latency Timer *(set by OS)* / Cache Line Size *(set by OS)*
0x10	Base Address Register *(set by OS)*	
	not implemented	
0x3C	Max_Lat = 0x04 / Min_Gnt = 0x04	Interrupt Pin = 0x01 / Interrupt Line *(set by OS)*

Figure 3. Configuration Space Addresses

Values for the status and command fields are shown in Tables 7 and 8. For complete descriptions of the bits in the status and command fields, see the *PCI Local Bus Specification*, Revision 2.1. Complete reference information is given on page 50.

Bit	Name	Value	Bit	Name	Value
0–4	reserved	0	10	DEVSEL Timing	0
5	66 MHz Capable	0	11	Signaled Target Abort	implemented
6	UDF Supported	0	12	Received Target Abort	implemented
7	Fast Back-to-back Capable	0	13	Received Master Abort	implemented
8	Data Parity Error Detected	implemented	14	Signaled System Error	implemented
9	DEVSEL Timing	1	15	Detected Parity Error	implemented

Table 7. Configuration Space Status Field Values

Bit	Name	Value	Bit	Name	Value
0	IO Space	0	6	Parity Error Response	implemented
1	Memory Space	implemented	7	Wait Cycle Control	0
2	Bus Master	implemented	8	SERR# Enable	implemented
3	Special Cycles	0	9	Fast Back-to-back Enable	implemented
4	Memory Write and Invalidate Enable	0	10–15	reserved	0
5	VGA Palette Snoop	0			

Table 8. Configuration Space Command Field Values

PCI Local Bus Addresses

Figure 4 describes the PCI53B interface registers in detail. The addresses listed in Figure 4 are offsets from the gate array boot ROM base addresses. This base address is initialized by the PCI Local Bus host operating system at boot time.

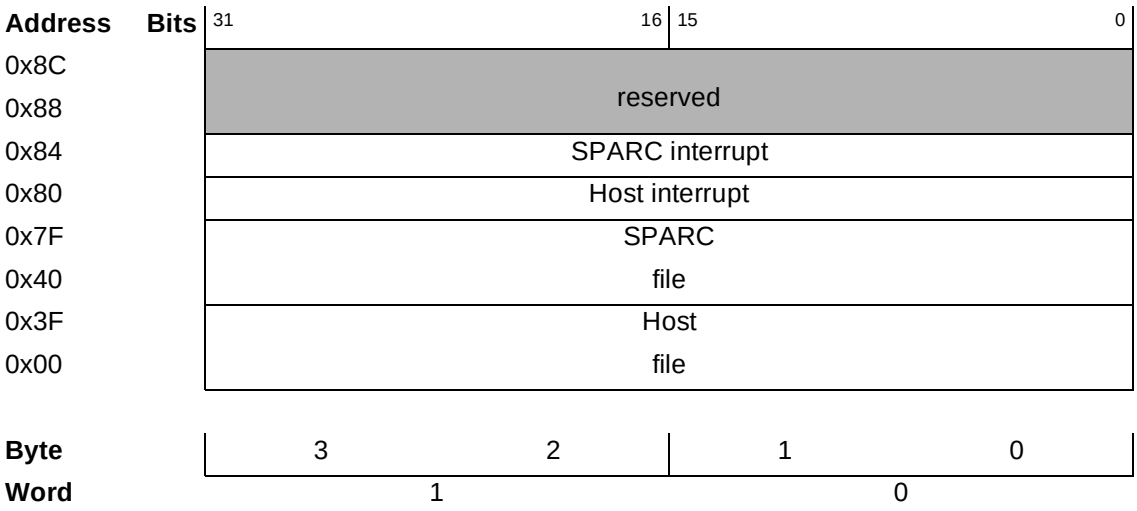


Figure 4. PCI Local Bus Addresses

Host File

The host file is a read-write register file of sixteen 32-bit words from address 0x00 to 0x3F. It is byte-addressable, but the PCI uses little-endian byte ordering (least significant bits start at 0), while the SPARC uses big-endian byte ordering.

The PCI host can read or write any location.

The SPARC can read these registers asynchronously, but cannot write them. Because the SPARC might read a register while the PCI host is writing it, thereby getting invalid data, software must coordinate host and SPARC accesses using semaphores and interrupts.

SPARC File

The host file is a read-only register file of sixteen 32-bit words from address 0x40 to 0x7F. It is byte-addressable, but the PCI uses little-endian byte ordering (least significant bits start at 0), while the SPARC uses big-endian byte ordering.

The onboard SPARC microprocessorcan read or write any location.

The PCI host can read these registers asynchronously, but cannot write them. Because the PCI host might read a register while the SPARC is writing it, thereby getting invalid data, software must coordinate host and SPARC accesses using semaphores and interrupts.

Host Interrupt Register

Size 32-bit

I/O read-write

Address 0x88

Bit	P53B_	Description
31	HOST_INT	The embedded SPARC sets this bit to interrupt the host over the PCI bus. The host then clears this bit by writing a 1. Also cleared by HALT (bit 29 of the SPARC interrupt register).
30	not used	
29	HINTEN	The host sets this bit to enable the host interrupt (bit 31 of this register).
23–0	not used	

SPARC Interrupt Register

Size 32-bit

I/O read-write

Address 0x84

Comments Used to allow the host computer to interrupt the embedded SPARC and to manage it.

Bit	P53B_	Description
31	not used	
30	SPARC_INT	Write a 1 to interrupt the SPARC; the SPARC then clears this bit . Reading a value of 1 indicates that the SPARC hasn't yet cleared this bit since the last time it was set.
29	not used	
28	TRAPE	Read-only: 1 if embedded SPARC is hung on an illegal trap.
27	HALT	Set to 1 to halt the embedded SPARC through its reset line. Clear to reboot SPARC.
26	HANG	Set to 1 to hang the embedded SPARC through its bus request line. Clear to resume operation where it left off.
25	REBOOT (w)	Strobe 1 to reinitialize the board, as if it had just been powered on. PCI configuration space left uninitialized.
24		reserved for EDT internal use
23–0	not used	

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 software, using relays
Built-in test	Yes
Connector	Trompeter BJ76 Concentric Triax Type Three Lug
Software	Drivers for Solaris 2.6+ (Intel and SPARC platforms), Red Hat Linux 6 for x86, and Windows NT/2000 Version 4.0
Memory	4 MB onboard memory (16 MB optional)
PCI Bus Compliance	PCI 2.1S
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 PCI bus slot
Dimensions	3.9" x 6.2" x 0.5"
Weight	6 oz.

Table 9. PCI Bus to MIL-STD 1553B Interface Specifications

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 element	The bus controller, bus monitor, or one of the remote terminals.
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	Electronically erasable programmable read-only memory.
ioctl	An input-output control operation on the PCI53B board, other than reading or writing.
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 PCI53B.
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 PCI53B.

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 µs of a 20-µs 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.

PCI Local Bus Specification, Revision 2.1, 1995. Available from:

PCI Special Interest Group
5440 SW Westgate Drive
Suite 217
Portland, OR 97221
Phone: 800 / 433-5177 (United States) or 425 / 803-1191 (international)
Fax: 503 / 222-6190

www.pcisig.com

Appendix A *ioctl( )* Parameters

Engineering Design Team recommends that applications use the software library interface documented in "PCI53B Library Routines" on page 17. This library is designed to be used with the following *ioctl* parameters. Others may be defined, but are used for Engineering Design Team's internal purposes only.

In the list below, **x** can be replaced by **S** or **G**.

P53S_BUS Selects the primary or secondary bus. Set to zero for primary, one for secondary. The default is primary. Provide the address of an unsigned short as its third argument.

P53S_MODE Used by the library to configure a subdevice as a BUS_CONTROLLER, a BUS_MONITOR, or RT_0 through RT_30. Not normally used in applications. Provide the address of an unsigned short as its third argument.

P53S_MYRTADDR
Used by the library to configure the address of an RT subdevice. Not normally used in applications. Provide the address of an unsigned short as its third argument.

P53G_MYRTADDR
Gets the address of an RT subdevice previously set by P53S_MYRTADDR. Not normally used in applications. Provide the address of an unsigned short as its third argument.

P53S_MODESIG
Enables MODE CODE events to signal the application process and sets the signal type. Provide the address of a `struct mode_sig` as defined in *p53b.h* as its third argument.

P53G_MODECODE
Get a modecode and optional data as an RT. Blocks until a mode code command addressed to this RT occurs. Provide the address of a `struct mode_data` as defined in *p53b.h* as its third argument.

P53S_MODECODE
Send a modecode and associated data from a BC. Blocks until an RT responds. Accepts the address of a `struct mc_buf` as defined in *p53b.h* as its third argument.

P53S_SRQ_V Set service request bit in the RT status word and clear on the next transmit vector word. Provide the address of an unsigned short as its third argument.

P53S_SRQ_X Set service request bit in the RT status word and clear on next transmit request. Provide the address of an unsigned short as its third argument.

P53S_RTRT Initiate an RT to RT transfer from this BC. Provide the address of a `struct rt_rt` as defined in *p53b.h* as its third argument. See the example provided in "Using the Data Structures as a Bus Controller" on page 31.

P53G_AVAIL Get the number of 1553 datawords avail from the BM. See sample program *bm.c* for usage details. Provide the address of an unsigned int as its third argument.

P53x_LOOPBACK
Set or get the state of the driver RT loopback mode. If nonzero, this mode creates one shared buffer for RT transmits and receives; therefore, data can be looped back by transmitting, then receiving, on an RT. If zero, then separate data buffers are used for transmit and receive data. The default enables loopback mode. Provide the address of an unsigned short as its third argument.

P53x_STATUS Set or get the RT status word. Provide the address of an unsigned short as its third argument.

P53S_AUTO_OFFSET
Set the offset into the "struct bc_auto" array at which P53S_AUTO_LOAD starts when loading the array into the PCI53B board memory. For example, you might declare an array of a hundred bc_auto structs, initialize, load, and start processing half of them, then initialize and load the remaining half. Provide the address of an unsigned int as its third argument.

P53S_AUTO_SIZE
Set the size of `struct bc_auto` array in the PCI53B onboard memory. Execution of `bc_auto` commands loops back to the beginning of the array when this number has been reached. Also controls the number of `bc_auto` structs transferred in P53S_AUTO_LOAD and P53S_AUTO_DUMP (send and receive the contents of a `bc_auto` array). Provide the address of an unsigned int as its third argument.

P53S_AUTO_LOAD
Upload the contents of a struct `bc_auto` array to the PCI53B onboard memory. For more information, see "IOCTL Parameters" on page 35. The third argument must contain the array address within application memory. Provide the address of an unsigned short as its third argument.

P53G_AUTO_DUMP
Download the contents of a struct `bc_auto` array from the PCI53B onboard memory. For more information, see "IOCTL Parameters" on page 35. The third argument must contain the array address within application memory. Provide the address of an unsigned short as its third argument.

P53S_AUTO_TODO
Sets the absolute number of `bc_auto` array elements for the PCI53B board to process. Processing does not start until P53S_AUTO_GO is invoked. AUTO_TODO is invoked once before AUTO_GO is called, then AUTO_CONT is used to continue execution. Provide the address of an unsigned int as its third argument.

P53S_AUTO_GO
Causes the P53B board to start processing `bc_auto` array elements. P53S_AUTO_LOAD must have loaded these elements , and P53S_AUTO_TODO must have set the number of them to process, before making this call. Provide the address of an unsigned int as its third argument.

P53S_AUTO_CONT
Increments the number of `bc_auto` array elements to process, and then either continues processing if already started, or restarts processing if done. Provide the address of an unsigned int as its third argument.

P53G_AUTO_WAITCNT
Blocks until the absolute number of `bc_auto` array elements has been processed. Provide the address of an unsigned int as its third argument.

P53S_AUTO_STOP
Causes processing of `bc_auto` array elements to be suspended upon completion of the current element. The third argument is ignored.

P53G_AUTO_CNT
Get the absolute number of `bc_auto` array elements completed. Provide the address of an unsigned int as its third argument.

P53G_AUTO_ERR
Returns the number of `bc_auto` array elements that encountered an error during processing, since processing started. Provide the address of an unsigned int as its third argument.

P53G_AUTO_WAIT
Blocks until the absolute number of `bc_auto` array elements set by P53S_AUTO_TODO has been completed. The third argument is ignored.

P53x_IMG Set or get the default intermessage gap for BC commands. Provide the address of an unsigned int as its third argument.

P53S_VWORD Set the RT vector word for transmit vector word mode code. Provide the address of an unsigned short as its third argument.

The following *ioctl* parameters set and clear the specified RT status word bits. Provide the address of an unsigned short as the third argument to all of them.

P53S_INS Instrumentation

P53S_SRQ Service request

P53S_RSV2 Reserved 2

P53S_RSV1 Reserved 1

P53S_RSV0 Reserved 0

P53S_SUBSYSBSY
Subsystem busy

P53S_SUBSYSFL
Sub system fail

P53S_BUS_ACPT
Dynamic BC accept

Index

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X