Platform Notes: The SunHSI/S™ Device Driver

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto,CA 94303-4900
U.S.A. 650-960-1300

Part No. 806-3983-10
February 2000, Revision 01

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Contents

CHAPTER 1

Installing the SunHSI/S Software

These Platform Notes include instructions for installing and configuring the software used by the SunHSI/S adapter and patch panel.

This chapter is organized as follows:

Software Overview

The SunHSI/S 3.0 software includes a network device driver and several utilities to diagnose the functionality of the SunHSI/S hardware.

The SunHSI/S 3.0 driver provides a streams-based interface to the Solaris operating environment. The streams interface works with other products such as system network architecture (SNA) 3270, SNA peer-to-peer, point-to-point protocol (PPP), and X.25 protocols. See Appendix B for a complete description of the software.

The SunHSI/S 3.0 software supports bundled Solaris synchronous interface diagnostic utilities (`syncstat`, `syncloop`, and `syncinit`) and also supports an enhanced set of synchronous utilities (`hsi_stat`, `hsi_loop`, and `hsi_init`). These synchronous utilities are described in Chapter 2.

Installing the Driver Software

The software required by the SunHSI/S adapter is on the Solaris CD that accompanies these Platform Notes.

▼ To Install the Driver Software

1. **Become superuser.**

2. **Use the** `prtconf -pv` **command to determine if the system contains a SunHSI/S device.**

```
# prtconf -pv | grep HSI
    alias: 'HSI'
    name: 'HSI'
```

   - If you see `HSI` in the output, your system contains a SunHSI/S device and you should continue with Step 3.
   - If your system does not contain a SunHSI/S device, shut down the system and install the adapter and patch panel as described in the *SunHSI/S 3.0 Installation and Administration Guide*.

3. **Use the** `pkginfo` **command to check the system for previous versions of the SunHSI/S software.**

```
# /usr/bin/pkginfo | grep SUNWhsis
system   SUNWhsis    HSI/S Driver/Utilities 2.0 v1.x
```

   - If no SunHSI/S packages are listed, skip to Step 4 to continue with the software installation.
   - If any SunHSI/S 2.0 or 3.0 packages are listed, you must remove them as described below.

     Removing the SunHSI/S 2.0 package:

```
# /usr/sbin/pkgrm SUNWhsis
```

Removing SunHSI/S 3.0 packages:

```
# /usr/sbin/pkgrm SUNWhsis SUNWhsism SUNWhsisu
```

Caution – Do not overwrite any existing SunHSI/S software packages.If you install the SunHSI/S software packages over existing SunHSI/S software packages, you will have two instances of the software packages. This may cause problems when installing or backing out of software patches.

4. **Install the SunHSI/S software as described in the** *Solaris Sun Hardware Platform Guide* **that shipped with these Platform Notes.**

   When you have completed the software installation, and the `pkgadd` utility has run the post-installation script, you will have created the devices illustrated FIGURE 1-1.

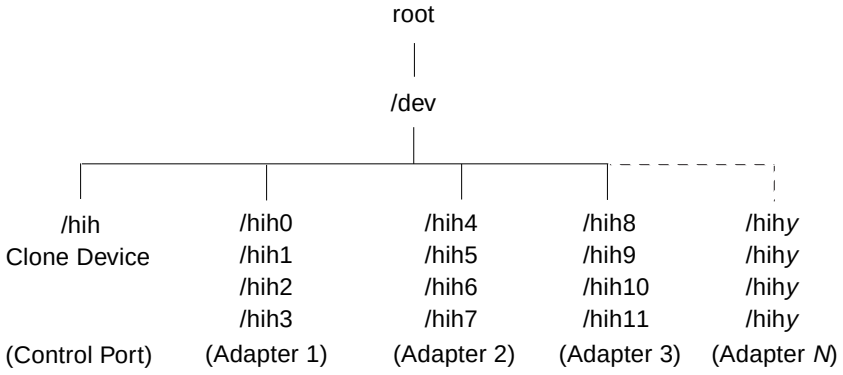

FIGURE 1-1 SunHSI/S Devices Created By the Post-Install Script

Post Installation Procedures

After installing the SunHSI/S software, use the `hsi_init` command to check links maximum transmission unit (MTU) and maximum receive unit (MRU) sizes, and, if needed, use the command to reset SunHSI/S hardware ports. The `hsi_init` command is described more completely in Chapter 2.

▼ To Check the MTU and MRU Sizes

Before operating an SunHSI/S link, make sure that the MTU and MRU sizes specified on each side of the link are the same on both sides.

- **Use the `hsi_init` command to check the MTU and MRU sizes (replace *N* with the port number of the link you are testing):**

```
# hsi_init hihN
```

Note – Checking the MTU and MRU sizes is especially important if you use the SunHSI/S software with a type of hardware other than the SunHSI/S card (such as the on-board serial port or third-party equipment). See “The hsi_init Command” on page 9 for instructions on how to set these sizes to match your hardware.

▼ To Change the Cabling or Equipment

1. **If you make any cabling or equipment changes on a port (for example, changing modems), reset the port with the `hsi_init reset` command.**

   N represents the SunHSI/S port number.

```
# hsi_init hihN reset
```

2. **After all the changes have been made, re-initialize the port with the `hsi_init` command. See “The hsi_init Command” on page 9 for more `hsi_init` command options.**

Viewing the Man Pages

The following man pages are included with the SunHSI/S software:

- `hsi(7d)`
- `hsi_init(1m)`
- `hsi_loop(1m)`
- `hsi_stat(1m)`
- `hsi_trace(1m)`

If you cannot view these man pages, you need to add the `/opt/SUNWconn/man` directory to your `MANPATH` environment variable.

Console Messages

This section lists line error console messages that may be displayed in your console window and a brief description of the error messages.

Note – *N* represents the port number.

Informational Messages

`hih`*N* `up and running at` *baud rate*`, mode=`*mode* `txc=`*txc* `rxc=`*rxc*

The SunHSI/S driver just brought up port `hih`*N* with the parameters shown. The baud rate shown in this message may be different from the externally set baud rate when external clocking is used.

`hih`*N*`: reset`

The SunHSI/S port *N* is reset.

Error Messages

`ERROR: hih_init: pll and !NRZI.`

Using the `hsi_init` command, the `txc` parameter was set to `pll`. However, the `nrzi` parameter was not set to `yes`. Setting the transmit clock source to `pll` requires NRZI data encoding.

`ERROR: hih_init: pll or baud and baud = 0`

The baud rate specified was 0 and internal clocking was set.

`hih`*N*`: Bad PPA =` *N*

SunHSI/S driver received a `DL_ATTACH_REQ`, which has an out-of-range PPA number *N*, from upper layers.

`hih`*N*`: port` *N* `not installed`

The SunHSI/S port *N*, which is referenced by the PPA number in a received `DL_ATTACH_REQ` message, is not installed to the system.

`hih`*N*`: out of STREAMS mblocks`

Running out of streams mblocks for SunHSI/S port *N*.

`hih`*N*`: xmit hung`

Transmission hung on SunHSI/S port *N*. This usually happens because of cabling problems or due to missing clocks from the CSU/DSU or modem.

`hih`*N*`: <hih_rxsoft> no buffers - rxbad`

Running out of streams mblocks for SunHSI/S port *N* in `hih_rxsoft()` routine.

Warning Messages

`WARNING: hih_init: changed baudrate from 100000 to 99512.`

The baud rate specified was rounded to a value the SunHSI/S hardware can support.

CHAPTER 2

SunHSI/S Utilities and SunVTS Diagnostic Testing

This chapter describes the utilities associated with SunHSI/S interface driver, and it introduces the SunVTS™ `sunlink` diagnostic software.

The SunHSI/S software includes its own serial port utilities. These utilities provide a superset of the features described in this chapter.

TABLE 2-1 SunHSI/S Utilities

SunHSI/S Command	Description
`hsi_init`	Initializes serial ports and enables you to modify and view driver-level parameters.
`hsi_loop`	Performs loopback testing to check the integrity of your data transmission path.
`hsi_stat`	Monitors serial port activity on a “snapshot” or repeating-interval basis.

Note – You must be superuser (root) in order to run the `hsi_init`, `hsi_stat` or `hsi_loop` utilities.

Software Port Names

The port naming conventions are used by initialization and serial port diagnostic commands. Software port names for SunHSI/S ports are of the form `hih`*N*, where *N* is a number in a range starting with 0 and ending at one fewer than the number of serial ports on your machine. For example, on a system with one SunHSI/S adapter installed, the serial ports are named `hih0`, `hih1`, `hih2`, and `hih3`.

SunHSI/S port names have the format: `hih`*y*, where *y* represents the port number. Initially, the port numbers `0`, `1`, `2`, or `3` are reserved for the first SunHSI/S adapter, and `4`, `5`, `6`, or `7` are reserved for the second adapter, and so on. For example, `hih1` would be the name for port 1 (the second port) on the first SunHSI/S adapter. TABLE 2-2 displays the relationship between the software port numbers (as used in SunHSI/S port names) and the hardware port numbers used on each adapter.

Note – The relationship between software and hardware port numbers will initially be set up as shown in TABLE 2-2. However, if you remove and replace a SunHSI/S adapter on the system, the software port numbers will be incremented to the next four numbers.

For example, when you initially install two SunHSI/S adapters on a system, the port numbers will range from 0 to 7. If you remove an adapter and install a new adapter, the new adapter's software port numbers will range from 8 to 11, and the new software port names will be: `hih8`, `hih9`, `hih10`, and `hih11`. If you continue to remove and replace SunHSI/S adapters, the software port numbers and port names will continue to be incremented up to a maximum of 159.

Use the `hsi_stat -a` command to display all the valid SunHSI/S ports on the system. See “The hsi_stat Command” on page 20 for more information about this command.

TABLE 2-2 SunHSI/S Hardware and Software Port Numbers

SunHSI/S Adapter Number	Hardware Port Number	Software Port Number
1	0	0
	1	1
	2	2
	3	3
2	0	4
	1	5
	2	6
	3	7
3	0	8
	1	9
	2	10
	3	11
N	0	(*N* - 1) x 4 + 0
	1	(*N* - 1) x 4 + 1
	2	(*N* - 1) x 4 + 2
	3	(*N* - 1) x 4 + 3

The `hsi_init` Command

The `hsi_init` command enables you to display and modify some of the hardware operating modes common to high-speed serial lines. These features make the `hsi_init` command valuable when troubleshooting and repairing problematic serial link lines.

Other applications also use the `hsi_init` command. For example, some applications use the `hsi_init` command to initialize serial ports, and the `hsi_loop` command (described in “The hsi_loop Command” on page 15) uses the command in a number of loopback tests.

When using `hsi_init` at the command line, the first argument required by the command is always the port name of the link being displayed or modified (for example, `hih0`). With no further arguments, `hsi_init` displays the parameter values as presently set on the selected link:

```
# hsi_init hih0
port=hih0 speed=1536000, mode=fdx, loopback=no, nrzi=no, mtu=1600, mru=1600,
    txc=txc, rxc=rxc, txd=txd, rxd=rxd, signal=no.
```

Note – You can use the `hsi_init` command to display the parameter values associated with a serial line, even if the serial device has been initialized through another command. However, you should not use `hsi_init` to modify any parameters on lines that were not initialized by `hsi_init`.

You can set all of the `hsi_init` parameters shown in the `Usage` statement below.

```
# hsi_init
Usage: hsi_init ifname \
    [baudrate] [loopback=[no|yes]] [nrzi=[yes|no]] \
    [txc=[txc|-txc|baud|rxc]] [rxc=[rxc|-rxc|baud]] \
    [mode=[fdx|ibm-fdx|ibm-hdx|ibm-mpt]] [signal=[yes|no]] \
    [external|sender|stop|reset] \
    [mtu=<mtu_size>] [mru=<mru_size>] \
    [txd=[txd|-txd]] [rxd=[rxd|-rxd]]
```

To set these parameters, use the syntax `hsi_init` *portname keyword=value*. For example, to set the maximum transmission unit (`mtu`) parameter of port `hih2` to 1000 bytes, you would type:

```
# hsi_init hih2 mtu=1000
```

TABLE 2-3 displays the possible values for each of these `hsi_init` parameters and lists the default values as initialized by the SunHSI/S driver for each port. The parameter values are described in greater detail after this table.

TABLE 2-3 `hsi_init` Parameter Values

Parameter	Default Value	Possible Values
`speed`	1536000 bps	The line speed can be set from 0 to 2048000 bps.
`loopback`	`no`	Can be set to `yes` or `no`. Useful when used with the `hsi_loop` command.
`nrzi`	`no`	Can be set to `yes` or `no`, depending on whether the port uses NRZI data encoding.
`txc`	`txc`	Sets the port transmit clocking signal to `txc`, `baud`, `rxc`, or `-txc`.
`rxc`	`rxc`	Sets the port receive clocking signal to `baud`, `rxc`, or `-rxc`.
`mode`	`fdx`	Sets the network mode to `fdx`, `ibm-fdx`, `ibm-hdx`, or `ibm-mpt`.
`signal`	`no`	Can be set to `yes` or `no`. When set to `yes`, the modem signal (RTS and CTS) state changes are reported by the driver to the application.
`mtu`	1600 bytes	The maximum transmission unit can be set from 1 to 1600 bytes.
`mru`	1600 bytes	The maximum receive unit can be set from 1 to 1600 bytes.
`txd`	`txd`	The transmit data signal can be inverted (`-txd`) to accommodate certain T1 or CEPT transmission equipment.
`rxd`	`rxd`	The receive data signal can be inverted (`-rxd`) to accommodate certain T1 or CEPT transmission equipment.

Note – See Appendix A “hsi_init Options” for more information about inverting the `txd`, `rxd`, `txc` and `rxc` options to accommodate the requirements of T1 or CEPT transmission equipment. Appendix A also contains more information about `mode` options.

`speed`

The `speed` parameter sets the line speed, or baud rate, of the serial line in bits per second. You can set this parameter to be from 0 to 2048000 bps.

In most situations, the actual line speed is determined by the modems in use, not by the Sun hardware, so the speed set by `hsi_init` is used only for compiling performance statistics for the `hsi_stat` command (see “The hsi_stat Command” on page 20). The `speed` parameter *is* significant when you are using the internal (workstation or server) baud generator to generate clocking. You invoke the internal baud generator when you use the `txc=baud` or `rxc=baud` settings with the `hsi_init` command (these parameters are described below).

Note – When you use `hsi_init` to specify a very high speed, and the `txc` or `rxc` parameters are set to `baud`, the actual speed (as reported by `hsi_stat` or another monitoring tool) can differ from the speed you specify, because the speed is rounded to the nearest integral multiple of the baud-rate generator clocking frequency. For example, after setting the speed parameter to 64000 bits per second, you may see `hsi_stat` report a line speed of, for example, 63750 bits per second.

`loopback`

This parameter sets and reports the internal loopback state of the serial chip. Setting a link to an internal loopback state (`loopback=yes`) is useful for testing serial ports that are not attached to external loopback equipment.

After testing a port, you can disable the internal loopback state on the port by setting the parameter to `no`:

```
# hsi_init portname loopback=no
```

This parameter is set to `loopback=yes` transparently by the `hsi_loop` command when you run the `hsi_loop -t 1` *portname* test. (See “The hsi_loop Command” on page 15 for more information.)

`nrzi`

This parameter sets the port to operate with NRZI (Non-Return to Zero, Inverted) data encoding. This parameter can be `yes` for NRZI encoding or `no` for NRZ encoding.

NRZ data encoding maintains a constant voltage level when data is present and does not return to a zero voltage until data is absent. The data is decoded as an absolute value based on the voltage level, which is 1 when data is present and 0 when data is absent.

NRZI data encoding does a voltage transition when data is absent (voltage level 0), and it does not do a voltage transition (no return to 0) when data is present (voltage level 0). With NRZI, the data is decoded using relational decoding.

`txc`

Sets the origin of the clocking for the transmitted data. For transmitted data, you can set the clock origin to:

- `txc` - Incoming transmit clock (TxCI signal)
- `-txc` - Inverted incoming transmit clock
- `rxc` - Incoming receive clock (RxC signal)
- `baud` - Internal (workstation) baud rate generator

The default for SunHSI/S ports is `txc=txc`. When `txc=baud`, the `speed` argument of the `hsi_init` command, not the modem clocking, controls the data rate. To accommodate the requirements of T1 or CEPT transmission equipment, this signal can be inverted (`txc=-txc`). See Appendix A “hsi_init Options” for more information.

`rxc`

Sets the origin of the clocking for the received data. You can set the clock origin to:

- `rxc` - Incoming receive clock (RxC signal)
- `-rxc` - Inverted incoming receive clock
- `baud` - Internal (workstation) baud rate generator

The default for SunHSI/S ports is `rxc=rxc`. To accommodate the requirements of T1 or CEPT transmission equipment, this signal can be inverted (`rxc=-rxc`). See Appendix A “hsi_init Options” for more information.

Note – While `rxc=baud` is supported on most Sun serial port options, it is useful only in conjunction with internal loopback mode (`loopback=yes`) or where the Sun machine is supplying clocking for one or both sides of a link.

`mode`

The `mode` parameter sets the operating mode of the serial link. The two main operating modes used by the SunHSI/S software are high-level data link control (HDLC) mode and IBM (SDLC) mode.

The values `mode` are:

- `fdx` - HDLC compatible full-duplex
- `ibm-fdx` - IBM compatible full-duplex
- `ibm-hdx` - IBM compatible half-duplex
- `ibm-mpt` - IBM compatible multi-point multi-drop

The default mode value is `fdx`. See Appendix A “hsi_init Options” for more information about operating modes.

`signal`

Controls whether modem signal (RTS and CTS) changes are reported back by the driver to the application. When this parameter is set to `yes`, these changes are reported.

`mtu/mru`

The `mtu` parameter sets the packet size of the maximum transmission unit, and the `mru` parameter sets the packet size of the maximum receive unit. By adjusting these parameters, you may achieve better performance out of the link. Both of these parameters can be set between 1 and 1600 bytes.

`txd/rxd`

These flags are used for inverting transmit (`txd`) and receive (`rxd`) data on serial lines. You can switch the polarity of a link by setting these flags to be negative (for example, `-txd` and `-rxd`). When `txd=txd`, the transmit data is not inverted, and when `txd=-txd`, the transmit data is inverted. Likewise, when `rxd=rxd`, the receive data is not inverted, and when `rxd=-rxd`, the data is inverted. These flags are useful when you run SunHSI/S over T1 and CEPT lines (see Appendix A “hsi_init Options” for more information).

You can also use the following set of one-word commands to specify useful combinations of `hsi_init` parameters.

TABLE 2-4 `hsi_init` One-Word Commands

Command	**Equivalent** `hsi_init` **Parameters**
`external`	`txc=txc rxc=rxc loopback=no`
`sender`	`txc=baud rxc=rxc loopback=no`
`reset`	(Resets the port and stops its operation.)
`stop`	`speed=0` (Stops the port.)

One clocking arrangement, called sender clocking, is useful for testing because it requires cabling only between the two systems under test, without intervening modems or modem eliminators. To configure sender clocking, use the `sender` command (`txc=baud`, `rxc=rxc`, `loopback=no`). This causes the transmitting side to generate a clock signal, which can then be routed to the receive clock on the receiving side. In fact, since each direction of data flow has independent clocking, the two directions can have different speeds, each determined by the `speed` option on the transmitting system.

Configuring Internal or External Clocking

To configure an RS-449 port to provide transmit clocking for itself as well as receive clocking for the other end of the link, set the `txc` (transmit clock) and `rxc` (receive clock) parameters in `hsi_init` to `baud` and `rxc`, respectively. For example, the following `hsi_init` command sets the data rate of the first CPU serial port to 9600 bps and sets the clocking as just described:

```
# hsi_init hih0 9600 txc=baud rxc=rxc
```

You enter such a command at both ends of a link if both sides are supplying clocking.

If you have Sun systems at both ends of a link and one machine supplies clocking for both sides, you would type the following on the machine that is not supplying clocking:

```
# hsi_init hih0 9600 txc=txc rxc=rxc
```

The `hsi_loop` Command

The `hsi_loop` command performs a loopback test that checks the following components of your communications link:

- Port-driver software layering
- CPU-to-port communication
- Correct operation of the serial port
- Cable from port to modem (or modem equivalent)
- Local and remote modems (or modem equivalents)
- Transmission line

When you invoke `hsi_loop`, it runs the `hsi_init` command to initialize the serial port and send out packets. `hsi_loop` then reads the incoming packets to verify that they were received. It also verifies the packet length and checks that the data is correct.

Note – Do not run `hsi_loop` on a port that is in use. Because certain `hsi_loop` options put a port in loopback mode, its use can prevent communication with a remote host during the time that `hsi_loop` is sending and receiving packets.

To stop an active port (in this example, `hih0`), type the following `hsi_init` command:

```
# hsi_init hih0 0
port=hih0 speed=0, mode=fdx, loopback=no, nrzi=no, mtu=1600, mru=1600,
    txc=txc, rxc=rxc, txd=txd, rxd=rxd, signal=no
```

As an alternative to specifying a speed of `0`, you can use the `stop` or `reset` commands (see TABLE 2-4). After you finish with `hsi_loop` testing, you must restart your link to reinitialize your serial port.

An `hsi_loop` command takes the following general form:

```
# hsi_loop[options] portname
```

where *portname* is, for example, `hih2`.

The options for `hsi_loop` are described in TABLE 2-5.

TABLE 2-5 `hsi_loop` Options

Option	Parameter Name	Description
`-c`	packet count	The number of packets used for data transfer. The default is 100.
`-l`	packet length	The length of the packet in bytes. The default is 100; the maximum is 1600.
`-s`	line speed	The bit rate in bits/sec. Applies only if local machine supplies clocking. The default line speed is 9600 bps.
`-t`	test type	A value in the range 1 – 4 that specifies the type of test `hsi_loop` performs (see the following subsection).
`-d`	hex data byte	A hexadecimal number that is the byte content of each packet. The default is to use random data.
`-v`	verbose	The verbose mode. If data errors occur, the expected and received data are displayed.

Enter all numeric options except `-d` (hex data byte) as decimal numbers (for example, `-t 3`). If you do not provide the test type option, `hsi_loop` prompts for it, as in the following example:

```
# hsi_loop hih1
[ Using /dev/hih1 ]
Enter test type:
1: Internal Test
            (internal data loop, internal clocking)
2: Test using loopback plugs
            (external data loop, internal clocking)
3: Test using local or remote modem loopback
            (external data loop, external clocking)
4: Other, previously set, special mode
> 2
```

An alternative to the preceding command is to specify the test type on the command line:

```
# hsi_loop -t 2 hih1
```

In the preceding command line, note that `hsi_loop` requires a space between the option switch (`-t`) and the number.

Test Type Options

This section contains descriptions of the available test type options. You specify test type options with the `-t` option, as described in TABLE 2-5.

Test Option 1 – Internal Test

This option uses the internal clocking and internal loopback and runs the following `hsi_init` command:

```
hsi_init portname speed loopback=yes txc=baud rxc=baud
```

The test data packets (100 by default) are sent to the specified serial port and looped back internally. You do not need a loopback plug for this option.

Test Option 2 – Test Using Loopback Plugs

This option uses the internal clocking and requires a loopback plug. Option 2 runs the following `hsi_init` command:

```
hsi_init portname speed loopback=no txc=baud rxc=rxc
```

The test data packet will loop between the CPU and serial port through the loopback plug. Before using this option, install an RS-449 loopback plug (part number 530-1430-01) on the specified port or the 96-pin loopback plug (part number 370-1381-01) into the back of the SunHSI/S adapter.

Test Option 3 – Test Using Local or Remote Modem Loopback

This option uses the external clocking set by the modem and runs the following `hsi_init` command:

```
hsi_init portname speed loopback=no txc=txc rxc=rxc
```

Testing with a modem in local loopback mode verifies proper operation of the serial port, the external cable, and the local modem. Testing with the modem in remote loopback mode checks the components just mentioned, as well as verifying the operation of the communications link.

With test type option 3, `hsi_loop` treats both local and remote modem loopback testing the same, since clocking is provided by the modem in both cases. Whether the data is looped back through the local or remote modem depends on how the modems are set up.

If the test fails on remote modem loopback but succeeds on local modem loopback, carefully check the transmission line and the setup of both modems.

Test Option 4 – Use Previously Set Mode

There is no automatic `hsi_init` execution with this option. This enables you to use `hsi_init` before running `hsi_loop` to specify clocking and loopback options that are not possible with the other `hsi_loop` test options.

For example, you can make the local side supply transmit clock at a desired speed (for example, 19200 bps: `hsi_init hih`*N* `speed=19200 txc=baud rxc=rxc`), and run the `hsi_loop` command on the local side with test option 4.

To run the test for the hsi_loop command, enter:

```
# hsi_loop -t4 hihN
```

Note – You cannot use the hsi_loop command to test for the correct operation of a null-modem cable between two Sun systems.

hsi_loop Output

When the loopback test runs successfully using any of the test type options, hsi_loop reports the statistics shown in the following sample output and then terminates.

```
# hsi_loop hih1
Enter test type:
1: Internal Test
            (internal data loop, internal clocking)
2: Test using loopback plugs
            (external data loop, internal clocking)
3: Test using local or remote modem loopback
            (external data loop, external clocking)
4: Other, previously set, special mode
> 2
speed=9600, loopback=no, nrzi=no, txc=baud, rxc=rxc
[ checking for quiet line ]
[ Trying first packet ]
[ Trying many packets ]
100
100 packets sent, 100 received
Port    CRC errors     Aborts   Overruns  Underruns     In <-Drops-> Out
hih1:           0          0          0          0      0             0
hih1:   estimated line speed = 9480 bps
#
```

The hsi_stat Command

The hsi_stat command provides information about the packets transmitted and received on a synchronous serial line. The hsi_stat command is a valuable tool for monitoring your serial link.

The syntax for hsi_stat depends on whether you want to display the statistics of a single port or a number of ports.

Displaying Statistics for a Single Port

If you want to display the statistics for a single port, the hsi_stat syntax is:

```
# hsi_stat [-c] [-f] device [period]
```

The *device* is the device name of the port (hih0, hih1, hih3, and so on), which is required to display the statistics from one port (see TABLE 2-6 for a description of the hsi_stat statistics). You can use the *period* option to display a series of port statistics at a specified interval of seconds.

By default, hsi_stat reports the cumulative statistics of the port since the system boot time. However, you can clear the statistics of a port using the -c flag. This flag resets the port statistics to zero.

With the -f flag you can display the full set of statistics of a serial port. This option is useful for debugging purposes.

Displaying Statistics for a Number of Ports

If you want to display the statistics of a number of ports, the hsi_stat syntax is:

```
# hsi_stat [-c] [-f] -a | number_of_ports
```

Replace the *number_of_ports* variable with a decimal number. The hsi_stat command displays the statistics of the specified number of ports. For example, if you type the command hsi_stat 3, hsi_stat displays the statistics of the first three valid ports.

The `-c` and `-f` flag can also be used on a number of ports. For example, `hsi_stat -c 2` clears the statistics of the first two valid ports. Use the `-a` flag to specify all the valid SunHSI/S ports on the system.

`hsi_stat` Output Description and Examples

TABLE 2-6 describes the statistics in the `hsi_stat` output.

TABLE 2-6 `hsi_stat` Statistic Descriptions

Statistic	Description
`speed`	The line speed as set by `hsi_init`. It is the system administrator's responsibility to make this value correspond to the modem clocking speed when clocking is provided by the modem.
`ipkts`	The total number of input packets.
`opkts`	The total number of output packets.
`undrun`	The number of transmitter underrun errors. Such errors occur when the local system is too busy to service the serial port hardware. A frame that is not completely sent is aborted, triggering error recovery. Underrun errors can occur when the signaling rate in use on a link is too fast for the local system.
`ovrrun`	The number of receiver overrun errors. Such errors occur when the local system is unable to accept data fast enough and the port hardware buffers overflow. A frame that is not completely received is aborted, triggering error recovery. Overrun errors can occur when the signaling rate in use on a link is too fast for the local system.
`abort`	The number of aborted received frames. Occurs when the local serial port receives a sequence of eight consecutive ones, in violation of LAPB/SDLC framing rules. Abort errors result from an interruption in the service provided by the link or from clocking problems. Such errors may also be caused by the software running above the driver level. A small number of abort errors probably indicates a software problem rather than a broken link or a persistent clocking problem.
`crc`	The number of received frames with CRC (Cyclical Redundancy Check, an error detection method) errors. A CRC error is recorded when the checksum on a received frame is incorrect. CRC errors occur when there is a clocking problem (different rates on each side) or a noisy line.
`isize`	The average size of input packets.
`osize`	The average size of output packets.
`iutil`	The input line utilization expressed as a percentage.
`outil`	The output line utilization expressed as a percentage.

TABLE 2-6 `hsi_stat` Statistic Descriptions *(Continued)*

Statistic	Description
`ierror`	The input error count. Errors can be incomplete frames, empty frames, or receive clock (RxC) problems.
`oerror`	The output error count. Errors can be lost clear to send (CTS) signals or transmit clock (TxC) problems.
`inactive`	The number or input packets received when receive is inactive.
`ishort`	The number of short input packets. This is the number of packets received with lengths less than the number of CRC bytes.
`ilong`	The number of long input packets. This is the number of input packets with lengths larger than the MRU.
`olong`	The number of long output packets. This is the number of output packets with lengths larger than the MTU.
`ohung`	The number of times the transmitter hangs. This is usually due to a missing clock.

Note – Errors under `abort`, `undrun`, `ovrrun`, and `crc` may indicate a problem with your serial port hardware, connectors, cables, or line-interfacing equipment. If you experience such errors, use `hsi_loop` (or an equivalent loopback diagnostic) to determine which component of your physical link is causing the errors.

The example below shows the `hsi_stat` command displaying the statistics of the `hih1` port:

```
# hsi_stat hih1
speed   ipkts   opkts  undrun  ovrrun   abort    crc   isize   osize
1536000  101     101      0       0       0       0     100     100
```

If you do not enter the optional `interval` parameter, `hsi_stat` terminates after printing the one-line cumulative total of the line statistics.

If you enter a time interval (expressed in seconds), `hsi_stat` operates in an iterative sampling mode, sampling and then displaying line use data for the period specified. In this mode, `hsi_stat` does not output cumulative totals, but displays the incremental changes in the totals between iterations. For example, the command:

```
# hsi_stat hih1 10
```

would produce output similar to the following. Note that in this example, the display is updated every ten seconds, until you press Control-C to stop the output.

```
ipkts opkts undrun ovrrun abort crc iutil outil
12    10    0      0      0     0     5%    4%
22    60    0      0      0     0     3%   90%
36    14    0      0      0     1    51%    2%
```

Using the `hsi_stat` command with an interval adds two fields to the report: `iutil` and `outil`. These two fields report the level of use for the serial line, as a percentage of incoming bandwidth (`iutil`) and outgoing bandwidth (`outil`). These percentages may occasionally be reported as slightly greater than 100% because of inexact sampling times and differences in the accuracy between the system clock and the modem clock. If the percentage of use greatly exceeds 100%, or never exceeds 50%, then the speed value of the `hsi_init` command probably varies greatly from the speed of the modem.

In the following example, the `-a` option is used to display all of the valid ports on the system.

```
# hsi_stat -a
 port     speed   ipkts   opkts  undrun  ovrrun   abort     crc   isize  osize
 hih8   1536000       0       0       0       0       0       0       0       0
 hih9   1536000       0       0       0       0       0       0       0       0
 hih10  1536000       0       0       0       0       0       0       0       0
 hih11  1536000       0       0       0       0       0       0       0       0
```

If you are experiencing communications problems, leave an `hsi_stat` command running on the console of the machine running an upper-level protocol so that you can see at a glance the recent history of loads and errors. For example, you can run `hsi_stat hih0 60` to get one-minute samples of activity on port 0.

If `hsi_stat` reports that line use is consistently near 100%, you may need a faster line. Also, watch for errors on the line, especially CRC errors in input packets. A small percentage of such errors can cause severe throughput reduction. These errors are almost always caused by troubles in the communication facilities.

If you see output packets but no input packets, either the remote system is not initialized or the line is not properly connected to the remote system.

If you see neither input nor output packets, the physical layer was not successfully initialized.

SunVTS Diagnostic Testing

The SunVTS software executes multiple diagnostic hardware tests from a single user interface. It is used to verify the configuration and the functionality of most hardware controllers and devices. You operate the SunVTS diagnostic primarily from a user interface that enables you to control all aspects of the diagnostic test operation.

The `sunlink` diagnostic test, which is shipped with the SunVTS software, checks the functionality of SunHSI/S adapters. This test can be run from the SunVTS user interface or from the command line. Refer to the SunVTS documentation for more information about the `sunlink` test.

APPENDIX **A**

`hsi_init` Options

This appendix contains information about T1 options and operating modes that can be set by the `hsi_init` command.

T1 Compatibility Options

The version of the `hsi_init` command shipped with the SunHSI/S software has options that enable you to invert data and clock signals to accommodate the requirements of T1 or CEPT transmission equipment.

The `hsi_init` parameters that enable inversion are:

- `txd` - transmit data signal
- `rxd` - receive data signal
- `txc` - transmit clock signal
- `rxc` - receive clock signal

When these parameters are at their default settings, the SunHSI/S 3.0 software does *not* invert the data or clock signal controlled by the parameter. To invert a signal, you specify a setting of the form *param_name=-paramname*, for example, `txc=-txc`.

As an example, suppose you wanted to invert the transmit and receive data signals on the first SunHSI/S port (port 0) on the second SunHSI/S adapter in your system, you would type the following command:

```
# hsi_init hih4 txd=-txd rxd=-rxd
```

To invert both clock and data signals, you would type:

```
# hsi_init hih4 txd=-txd rxd=-rxd txc=-txc rxc=-rxc
```

Inverted Settings

This section discusses the background and requirements for these inverted settings.

The reason for inverting data signals is distinct from the reason for inverting clock signals.

Data Signal Inversion

The requirement for inverting data signals arises from the “ones density” problem you encounter with most T1 transmission lines in North America. The T1 transmission scheme uses a signaling mechanism known as Alternate Mark Inversion (AMI), in which one bits are represented by a positive or negative pulse, while zero bits are represented by the absence of a pulse. In this scheme, the polarity of each pulse must be the opposite of the polarity of the pulse that immediately preceded it. This signaling scheme makes it possible to embed a reference clock for the data into the data stream itself.

Various types of T1 transmission equipment, such as Data Service Units (DSU), Channel Service Units (CSU), repeaters, and various telephone central office equipment, must be able to keep a phase locked loop (PLL) circuit locked on to this reference clock. This PLL circuit uses the pulses generated when one bits are transmitted to lock the embedded clock to a local reference oscillator. To keep the PLL circuit locked on the extracted clock, a certain density of pulses (one bits) must be guaranteed. For North American T1 lines, the density requirement dictates that at least one out of every 16 bits must be a one (see *AT&T Technical Publication 62411*). In other words, no more than 15 consecutive zero bits can occur anywhere in the data stream.

T1 lines were originally intended to carry voice traffic, wherein the digitized voice signals could be altered to meet the ones-density requirement by forcing every eighth bit of a voice channel to be a one. This practice introduces a small—but virtually inaudible—amount of distortion in the voice signal. Digital data streams between two computers are another matter, since the corruption of even one data bit causes a packet to be rejected. Note that in a typical data packet, it is quite easy to produce bit patterns that violate the ones-density requirement. A random file could easily contain a sequence of bytes that would produce 16 or more consecutive zero bits if transmitted serially.

There are many different schemes for circumventing the ones-density requirement. The most common technique simply reserves every eighth bit of the signal for a “density bit” and forces this bit to be a one. Obviously, these bits are not available for data transmission, which means that 12.5 percent of the bandwidth of the T1 line is wasted. When you consider that the lease cost for a coast-to-coast T1 line can be exceedingly expensive, this waste of bandwidth can be unacceptable.

There are other alternatives. One of them uses a special code that transmission equipment can generate when using the AMI signaling scheme. This special code depends on the fact that two successive one bits that are represented by pulses of the same polarity result in a signal known as a “Bipolar Violation.” A CSU can be designed so that it will automatically replace any string of eight consecutive zeros with a special code pattern that contains two of Bipolar Violations. A compatible, receiving CSU recognizes this special code and converts it back to a pattern of eight zeros. This technique is known by the acronym B8ZS, which stands for Bipolar with 8-Zero Substitution.

All CEPT lines (the European equivalent of T1) mandate the use of a variant of B8ZS that holds the density requirement down to no more than three consecutive zeros. However, telephone companies in North America have been slow to adopt B8ZS, because it would entail a significant capital investment. Therefore, the B8ZS solution will not solve the ones-density problem in the short term.

An alternative to B8ZS—an alternative used by the SunHSI/S product—is based on the HDLC framing rules, which specify that any data stream that contains five or more consecutive one bits requires that the transmitting end insert a zero bit after the fifth one bit. This guarantees that the HDLC flag pattern, `01111110` (hex 7E), does not occur randomly inside a frame. The receiving end must automatically discard the zero bit that follows a pattern of five consecutive ones. So, the HDLC framing used by the SunHSI/S software guarantees that in any set of six bits, at least one bit will be a zero (except for the flag pattern). If you include the flag pattern, you can say that in any set of seven bits, at least one bit will be a zero.

By inverting the data signal with HDLC framing on both ends of a link, the HDLC zero insertion algorithm becomes a ones insertion algorithm. This guarantees that in any set of seven bits, at least one bit will be a one. Thus, the HDLC data stream meets the density requirements of North American T1 lines without sacrificing any bandwidth.

Clock Signal Inversion

The need to invert clock lines is separate from the need to invert data lines. Most computer, modem, and terminal vendors adhere to an industry standard specification known as RS-334. This specification defines the relationship between a data bit and a reference clock on a synchronous serial link. The specification also

says that a device should transmit data with reference to the rising edge of the clock signal and that data should be received with reference to the falling edge of the clock signal.

When using long cables or cables not carrying a clock signal, a phase shift may occur, causing a high number of errors. In such cases, inverting the clock signal may correct the phase shift. You may also need to invert the clock signal when connecting a SunHSI/S port to equipment not adhering to the RS-334 standard.

Operating Modes

The SunHSI/S driver operates in two main operating modes: the high-level data link control (HDLC) mode and the IBM (SDLC) mode. The HDLC mode always operates in a full-duplex, point-to-point fashion. While the IBM mode defaults to a full-duplex, point-to-point, operation, you can also set this mode to be either a half-duplex or a multi-point operation.

HDLC Mode

The default operating mode used by the SunHSI/S driver is the HDLC full-duplex protocol (`mode=fdx`). In this mode, the transmitter is always enabled, and it sends flag bytes continuously when it is not sending a data frame.

If no message is currently being transmitted, the driver will attempt to start sending its next message. At this point, the driver indicates that it is busy transmitting to prevent the transmission of another message concurrently. The driver also activates a mechanism that ensures that the transmit operation will not hang if the hardware is not responding.

When the transmission is completed, the busy mechanism previously set is cleared and the next message can be transmitted. If the transmission is hung, an abort sequence is sent instead of the CRC so that the receiver will not interpret the frame as valid data. The message is discarded, and the output error statistic is incremented, which allows for a proper recovery by higher level protocols.

The received data is buffered until a complete frame has been received. If any error occurs during the reception of a frame, the appropriate statistic is incremented and the frame is discarded.

IBM (SDLC) Mode

This mode is designed to support IBM system network architecture (SNA) communications. It uses most of the same protocols used in HDLC mode, with two major exceptions:

- When the line is idle, instead of sending flag bytes, the transmitter is disabled.
- The request-to-send (RTS) and clear-to-send (CTS) signals are used to gate transmission.

IBM Full-Duplex Mode

When the SunHSI/S software is set to this mode (`mode=ibm-fdx`), the software uses a full-duplex point-to-point communication protocol. Both ends of the link are expected to have RTS and CTS signals asserted at all times when data is being exchanged. When starting a message transmission, the interface raises the RTS signal and expects the CTS signal to be asserted immediately. If this is not done, all messages currently queued for transmission are discarded, and the write operation returns an error.

If the CTS signal drops before the frame transmission is complete, the frame is discarded and the abort error statistic is incremented. If the transmission underruns, an abort sequence is *not* sent and the frame is silently discarded. The RTS signal remains asserted until the data transmission is complete.

IBM Half-Duplex Mode

Half-duplex is a sub-mode of the IBM mode (`mode=ibm-hdx`). Half-duplex mode operates in the same manner as full-duplex mode, except that transmission cannot occur while receiving, and vice versa. When a transmission is completed, the RTS signal is dropped to “turn the line around.” Dropping the RTS signal tells the remote station to begin transmitting if needed.

IBM Multi-Point Mode

In a multi-point configuration (`mode=ibm-mpt`), more than two stations “share” a link. This configuration designates one station as a primary station and the rest as secondary stations. In this mode, the port acts as a secondary station. The primary station arbitrates traffic on the link by polling the secondary stations, asking them all if they are ready to transmit.

If a secondary station has data to transmit, it will raise its RTS signal and check for CTS signals. When a CTS signal comes up, the station may begin transmitting, following the same rules for RTS and CTS signals used in half-duplex mode. When

the transmission is complete, the secondary station drops the RTS signal, which enables another station to respond to a poll and begin transmitting. The RTS signal cannot be dropped until the transmission is complete.

APPENDIX **B**

Software Functional Description

This appendix contains a functional description of the SunHSI/S 3.0 software.

Software Initialization

The SunHSI/S 3.0 driver software is dynamically loadable and unloadable to help conserve memory resources. During software installation, the driver is installed to the system with `add_drv(1m)`, which temporarily loads the driver into the kernel and uses the attach routine of the driver to dynamically create the device nodes for each hardware instance installed. This autoconfiguration feature is provided in the Solaris software, and it is also supported in the SunHSI/S 3.0 driver software.

After the autoconfiguration, the module is unloaded. The driver module is loaded to the system again when the driver is first referenced. The autoconfiguration and initialization of the SunHSI/S driver software is performed through a set of standard SBus device driver routines:

```
static int hsidentify (dev_info_t *dip)
```

This routine is called at initialization time to find out whether the driver controls the device specified by parameter `dip`. The driver compares `ddi_get_name(9E)` with a hard-coded string "HSI" with `strcmp(3C)`. This routine returns the `DDI_IDENTIFIED` message if both strings match. Otherwise, the `DDI_NOT_IDENTIFIED` message is returned.

`static int hsprobe (dev_info_t *dip)`

This routine is called at initialization when the calling of `hsidentify` succeeded. Since SunHSI/S hardware is a self-identifying device, the system performs the probe function. This routine always returns the `DDI_PROBE_SUCCESS` message.

`static int hsattach (dev_info_t *dip, ddi_attach_cmd_t cmd)`

This routine is called at autoconfiguration time when the driver module is loaded into the system and the calling of `hsprobe` routine succeeded. It calls the protocol-dependent routine `hih_attach` to create device nodes that the driver needs to access the hardware and `hih_init` to set up a control structure for each port on the board.

The structure includes information such as hardware address, transmission state, minor number, and port configuration parameters. The `hsattach` routine also calls the standard DDI routines to map SunHSI/S hardware registers and to add device interrupt service routine to the kernel.

`static int hsdetach (dev_info_t *dip, ddi_detach_cmd_t cmd)`

This routine is called when the driver module is unloaded from the system. It calls the protocol-dependent routine `hih_detach` to remove the device node and reset the hardware. It also calls standard DDI routines to un-map the hardware registers and delete the device interrupt service routine from the system.

External Interfaces

The SunHSI/S 3.0 driver provides a streams-based interface to the Solaris kernel and user program. The driver software can be reached from the user program through standard `open(2)`, `close(2)`, `putmsg(2)`, `getmsg(2)`, and `ioctl(2)` system calls. The driver software communicates with other kernel resident (upper protocol) streams modules using the standard utility `putnext(9F)`.

The driver software can also be reached through hardware interrupts from the SunHSI/S hardware. Hardware interrupts, both standard serial control interrupts and on-chip DMA interrupts, are received through the SunHSI/S hardware from the Integrated Serial Communications Controller (ISCC).

The SunHSI/S driver software interface diagram, shown in FIGURE B-1, shows the external interfaces of the driver.

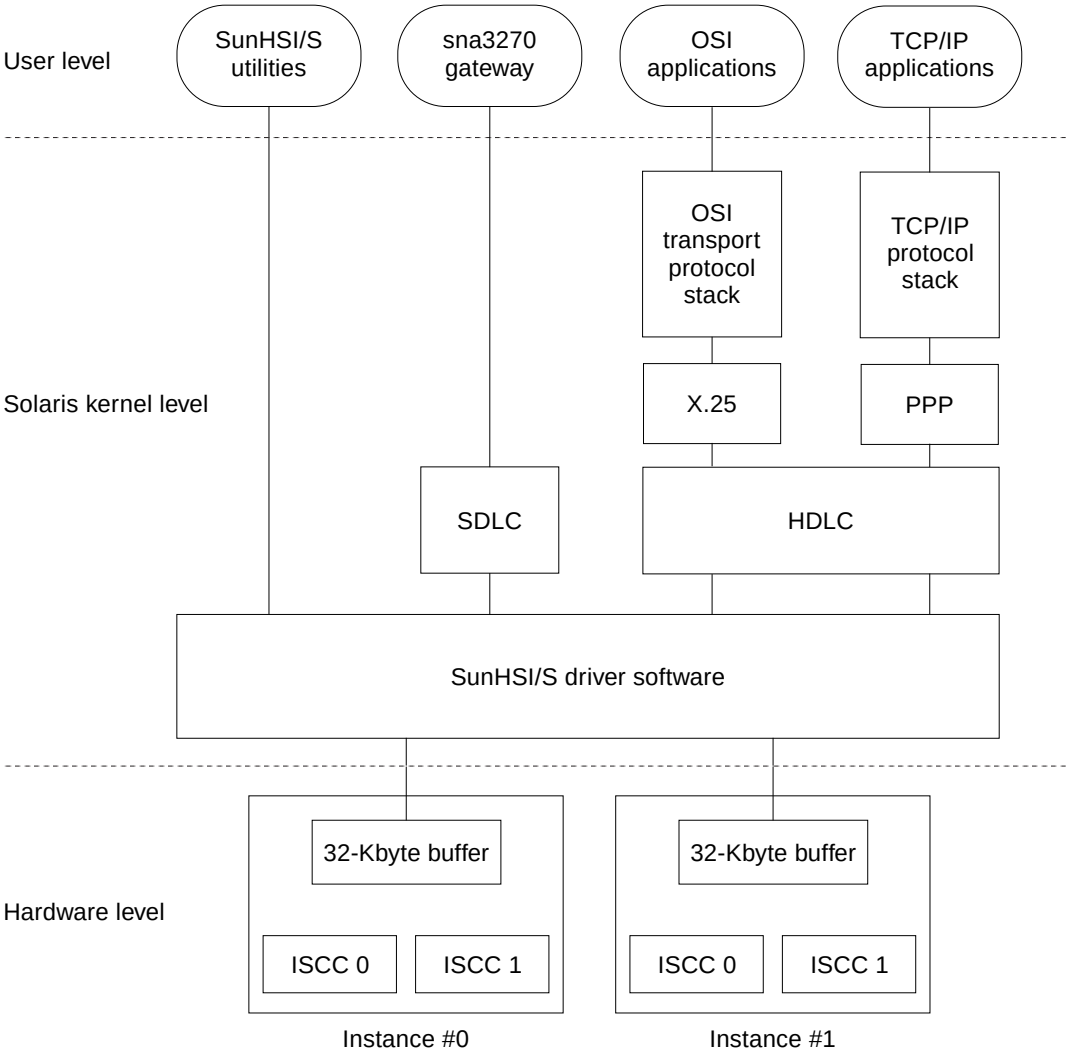

FIGURE B-1 SunHSI/S 3.0 Driver Software Interface

IOCTLs

All driver control is achieved through IOCTL system calls (refer to the `ioctl(2)` man page for more information). TABLE B-1 lists the `IOCTL` parameters for the SunHSI/S Driver.

TABLE B-1 IOCTL Parameters for the SunHSI/S Driver

IOCTL Name	Purpose and Structure

`S_IOCGETMODE`	Retrieves the current transmission parameters setting for a particular port. Structure required:
	`struct scc_mode {`
	`char sm_txclock; /*transmit clock sources */`
	`char sm_rxclock; /*receive clock sources */`
	`char sm_iflags; /*data and clock invert flags */`
	`u_char sm_config; /*boolean configuration options*/`
	`int sm_baudrate; /*real baud rate */`
	`int sm_retval; /*SMERR codes go here,`
	`query with GETMODE */`
	`};`

`S_IOCSETMODE`	Reinitializes a particular port with new transmission parameters setting. Structure required: Same as `S_IOCGETMODE`
`S_IOCGETSPEED`	Retrieves the current baud rate setting for a particular port. Structure required: `int speed;`
`S_IOCGETMRU`	Retrieves the current Maximum Receiving Unit (MRU) setting for a particular port. Structure required: `int mru;`
`S_IOCSETMRU`	Sets to a new Maximum Receiving Unit (MRU) for a particular port. Structure required: `int mru;`
`S_IOCGETMTU`	Retrieves the current Maximum Transmission Unit (MTU) setting for a particular port. Structure required: `int mru;`
`S_IOCSETMTU`	Sets to a new Maximum Transmission Unit (MTU) setting for a particular port. Structure required: `int mru;`
`S_IOCGETMCTL`	Retrieves the current DCD/CTS state for a particular port. Structure required: `u_char mctl;`

TABLE B-1 IOCTL Parameters for the SunHSI/S Driver *(Continued)*

IOCTL Name	Purpose and Structure
S_IOCGETSTATS	Retrieves the data or errors statistics that SunHSI/S driver has accumulated for a particular port. Structure required:

```
struct hs_stats  {
    long     ipack;      /*input packets*/
    long     opack;      /*output packets*/
    long     ichar;      /*input bytes*/
    long     ochar;      /*output bytes*/
    long     abort;      /*abort received*/
    long     crc;        /*CRC error*/
    long     cts;        /*CTS timeouts*/
    long     dcd;        /*Carrier drops*/
    long     overrun;    /*receiver overrun*/
    long     underrun;   /*xmitter underrun*/
    long     ierror;     /*input error (rxbad)*/
    long     oerror;     /*output error
                         (watchdog timeout)*/
    long      nobuffers;  /*no active receive block available
    long     ishort;     /*input packet too short
                         (<CRC-bytes+1)*/
    long     ilong;      /*input packet too long (> mru)*/
    long     inactive;   /*input packet received
                         when inactive*/
    long     idma;       /*receive dma error*/
    long      olong;      /*output packet too long (> mtu)*/
    long     ohung;      /*transmit hung (usually
                         missing clock)*/
    long     odma;       /*transmit dma error*/
};
```

or:

```
struct sl_stats  {
    long     ipack;     /*input packets*/
    long     opack;     /*output packets*/
    long     ichar;     /*input bytes*/
    long     ochar;     /*output bytes*/
    long     abort;     /*abort received*/
    long     crc;       /*CRC error*/
    long     cts;       /*CTS timeouts*/
    long     dcd;       /*Carrier drops*/
    long     overrun;   /*receiver overrun*/
    long     underrun;  /*xmitter underrun*/
    long     ierror;    /*input error (rxbad)*/
    long     oerror;    /*output error (watchdog timeout)*/
    long     nobuffers; /*no active receive block available*/
};
```

TABLE B-1 IOCTL Parameters for the SunHSI/S Driver *(Continued)*

IOCTL Name	Purpose and Structure
S_IOCCLRSTATS	Clears the data and error statistics that SunHSI/S driver has accumulated for a particular port. Structure required: Same as S_IOCGETSTATS

The scc_mode and sl_stats structures defined in the system include file (/usr/include/sys/ser_sync.h).

Interrupts

Hardware interrupts are serviced through the hsintr interrupt service routine. This routine determines the source of the interrupt using an interrupt vector read from the ISCC chip. If the interrupt is *not* from the SunHSI/S adapter(s), the procedure returns a zero value. If the interrupt *is* from the ISCC chip, the interrupt is serviced. The possible hardware interrupts are listed in TABLE B-2.

TABLE B-2 Hardware Interrupts

Interrupt	Cause
External Status	Either a Break or an Abort was transmitted over the DLC line. An Abort is series of 15 successive ones sent over the line by the transmitting side.
Transmit Interrupt	A full packet was transmitted over the line.
Receive Interrupt	Special circuitry on the SunHSI/S adapter detects the receipt of a complete frame and interrupts the system processor(s).
DMA Transmit Terminal Count	A complete packet was generated. In addition to this interrupt, a Transmit Interrupt occurs at transmission.
DMA Receive Terminal Count	The received packet is larger than the DMA Receive Buffer.

Packet Transmission and Reception

When an upper-protocol layer or a user program has a packet ready for transmission by the interface, it calls either the `putnext(9F)` utility or the `putmsg(2)` system call to pass the packet to the SunHSI/S driver. If the SunHSI/S driver transmission buffer is empty, the `hih_wput` routine of the driver copies the packet into the RAM buffer on the SunHSI/S adapter. From there, the packet is transmitted across the serial line. If the transmit buffer is full, the packet is queued at the local `WRITE` queue for later transmission by the `hih_wsrv` routine.

If the local `WRITE` queue has too many packets (beyond the high-water mark), the upper layers detect the congestion by calling `canputnext` and slow down the traffic until the congestion is resolved.

When a correct packet is received, it is copied from the SunHSI/S board RAM to a stream message buffer. When a complete packet is received, the `hih_rsrv` routine of the driver is called to send the packet to upper layers.

The Z16C35 ISCC support an internal status FIFO of approximately ten packets. As a result, you can queue many packets during reception without servicing an interrupt. Since most synchronous protocols require relatively fast reception of control packets, you do not want packets to queue at the driver level for long. To alleviate this possible problem, an algorithm based on the receive queue size and a timer is used. Either event causes the packets to be sent to the upper level.

Index

R

S

T

V