Convergent Technologies

RELEASE NOTICE
for
1.13 S/Series Remote I/O Processor

Revised December 15, 1989

Trademark Notice

TABLE OF CONTENTS

1. General Description of the RIOP Option

The Remote I/O Processor (RIOP) at Release 1.13 provides connectivity of up to 96 RS-232 serial devices to S/Series machines. There can be a maximum of 64 terminals and 32 other devices attached to S/Series machines capable of supporting the RIOP. This is achieved by attaching a number of these serial I/O multiplexors to the lines of an RS-422 Expansion Board. Each RIOP can be configured with either eight or sixteen RS-232 ports. There can be up to a total of sixteen RIOP's connected to one system. These can be distributed in any way between the 2 or 4 lines of the RS-422 Expansion card with a maximum of eight RIOP's on one RS-422 line. This Release provides the operational and host interface software required for use of the Remote I/O Processor.

For this Release, the RIOP's can share a single RS-422 line with only one other kind of device. This is the PC running PC Exchange/VINES interconnect software. Both of these devices run the line at 1.8 Mbit per second. Other separate RS-422 lines of the Expansion card, however, can be connected simultaneously to PT's or GT's running at 307 Kbaud. With the addition of PC's, the number of RIOP's on a line should be reduced appropriately.

The software provided with this Release contains user and kernel level code which runs on the S/Series machine as well as a downloadable image to run on the Remote I/O Processor itself. The host side software contains a user level daemon and a loadable virtual tty driver.

The user-level daemon controls downloading of software into the RIOP's. The virtual tty driver makes use of CTIX resident line discipline code, packetizing data gathered into *clists* for output to the cluster driver and depacketizing and demultiplexing input data from the cluster driver into the input *clists*. The cluster driver communicates with the RIOP's using a polled, master/slave protocol. The protocol used is a subset of the unbalanced Normal Response Mode of the Advanced Data Communications Control Procedures (ADCCP).

At the RIOP end, data received from the RS-232 ports is multiplexed and packetized for shipment to the host while output data is received from the host and transmitted to the individual ports.

The RIOP at Release 1.13 has the ability to handle some of the line discipline code at the remote device. This allows *clists* to be bypassed on output and echoing of characters to be handled remotely. Although a great deal of tty processing can still be done at the host, the number of interrupts to the hosts is reduced substantially since blocks of output data are sent in one packet and since input from up to 16 ports can be sent in one packet.

2. Changes from the Previous Release

2.1 Bugs Fixed in this Release

This Release for 6.2 CTIX is functionally equivalent to the RIOP subsystem in 5.25.3 CTIX. The following bug fixes are new for those users moving from a 1.00 RIOP Release (pre 5.25.1 CTIX). These bugs were fixed in 5.25.1 CTIX and 1.10 RIOP and they are included in 1.13 RIOP.

1. The **TCSETAF** tty ioctl command would cause output data to be lost on RIOP ports. Also the **TCSETA** command would wait for output to drain before doing a port reparameterization. (See *termio(7)*). Since the generic tty code in CTIX prior to 5.25.1 had no mechanism for informing the device level that a wait was in progress, RIOP code always waited for all output to drain before doing a port reparameterization. This allowed the more critical **TCSETAW** command to work correctly. But it also caused a flush to occur during the **TCSETAF** command before the wait was specified. This mechanism was added to the generic tty code in 5.25.1 and 6.10 CTIX. The RIOP code then recognizes and handles the wait cases correctly.
2. A problem where lack of buffers at the RIOP could cause loss of a message and consequently hang a terminal was fixed.
3. A problem where rebooting RIOP's would cause RS-422 PT's to reboot was fixed.

The following bug was fixed in 1.11 RIOP (6.1 CTIX) and are included in the 1.13 RIOP Release.

1. Some RS-232 receive interrupt code which read more than one byte of data from the receive FIFO was removed. It was found that in some receive error cases the Z8530 chip does not always reset the bit which indicates whether more data is available. This would in turn cause an infinite loop until the watch-dog timer forced a reboot. The code now only attempts to read one byte of data from the FIFO at each receive interrupt.

The following bugs were fixed in 5.25.3 CTIX and are included

in the 1.13 RIOP Release.

1. The byte sequences 0x9f and 0xdf were being transposed to 0x09 and to 0x0d 0x09. This made the RIOP useless for 8-bit Kanji character set applications. Some code in the RIOP driver main function is now always included.

2. If **cl_deflines** in **/etc/system** was changed to '1' and RIOP's are used, the system crashed with a "**panic:cl_incalxbs: invalid allocation**". Changes were made to the cluster driver at the point where an entry is removed from the xbuf queue to call a low-water routine in an upper layer of the cluster protocol.

3. Executing **stty ff1 and/or cr2** on a serial printer connected to an RIOP port caused the RIOP to reboot. There are changes in the vtd driver routines to fix the character delay. There are fixes in the RIOP Z8530 driver code for character delays and improved input handling.

4. On an S/640 with RIOP's, all terminals on one or more of the RIOP's locked up within a few seconds to a few hours of operation. The system had to be rebooted to recover. The console repeatedly displayed the message: "**panic: watchdog timeout**". There are fixes in the RIOP Z8530 driver code for character delays and improved input handling.

2.2 New Functionality in this Release

The following sections describe enhancements which have been made to the RIOP or Cluster Driver. This Release for 6.2 CTIX is functionally equivalent to the RIOP subsystem in 5.25.3 CTIX. The following features are new for those users moving from a 1.00 RIOP Release.

2.2.1 External Processing on the RIOP

The tty code which handles output processing of characters in line discipline has been ported to the RIOP resident download image. Line discipline is the standard tty interface as described in *termio(7)*. Doing external output processing allowed two other enhancements.

The first is the ability to echo characters local to the RIOP. This is helpful when in cooked mode, for example, when

issuing commands to a shell, as it greatly reduces the number of messages sent between the RIOP and host. Unfortunately, for most screen based applications which operate in raw mode, this has no effect.

The second enhancement made possible is an optimization at the host end. This allows output data to be moved directly from the user space into the output buffer used by the Cluster Driver for DMA transfer onto the RS-422 line. Doing this bypasses *clist* buffering altogether. This optimization is effective in both raw and cooked modes and greatly increases host CPU idle times during periods of heavy tty output.

All of these related features are treated as a single feature which is referred to simply as external processing. This external processing feature can be disabled or enabled on a per RIOP basis (see *riopcfg(1M)*) or on an individual port basis (see *extproc(1M)*). The default state is for external processing enabled on all ports. The reason for making the feature optional is that for small numbers of RIOP ports, output throughput might be somewhat higher (at the expense of host CPU cycles) when the feature is disabled. This is true since the RIOP has more processing to accomplish and might become the bottleneck in some cases when the feature is enabled. Please see Section 6, *"Performance Considerations"*, which gives benchmarking results.

2.2.2 New Cluster Driver Buffer Allocation

In the release prior to 1.11 RIOP, the buffer allocation routines of the Cluster Driver were called by the RIOP virtual terminal driver. Unfortunately, many assumptions based upon PT (RS-422 terminal) needs are embedded in these routines which are not truly suited for RIOP use.

In the current release the virtual terminal driver contains its own buffer allocation routines which are used internally and are also called from below by the Cluster Driver when the drop is of type RIOP. These allocation routines allow a greater number of buffers to be allocated to an RIOP drop than was allowed for a PT. Because of this, processes are less likely to sleep waiting for buffers and fewer process switches should occur.

The new allocation scheme also allocates new pools of buffers dynamically as more RIOP's come on-line. The old scheme used an algorithm based on **cl_defdrops** and **cl_deflines** in **/etc/system** to determine the number of buffers statically at boot time. The new method eliminates the need for setting **cl_defdrops** at an artificially high value just to get a reasonable amount of buffers allocated for RIOP use. This then allows the Cluster Driver overhead to be optimized once again by minimizing this value.

The new method is automated and invisible to the system administrator. The system administrator should be aware however that about 5K of kernel virtual memory will be allocated for each on-line RIOP port. These buffer pools are not deallocated when RIOP's go out of service but they remain available for subsequent use. The pools are only deallocated when the virtual terminal driver is unloaded.

2.2.3 Modem Control Signaling

The RIOP tty ports support the new modem control capability accessible through the *ioctl(2)* system call.
This has the following format:

ioctl(fildes, command, arg)

The new commands are:

TCGEXT — Get the parameters associated with the terminal. The parameters are passed back as the return value from the ioctl() function. The return value is defined as follows (see **/usr/include/sys/tty.h**):

```
#define CDBIT    0x04    (CD is present)
#define CTSBIT   0x08    (CTS is present)
#define DSRBIT   0x10    (DSR is present)
#define RIBIT    0x20    (RI is present)
```

TCSEXT — Set the parameters associated with the terminal from arg. The bits in arg are defined as follows:

```
#define RTSBIT   0x08    (set/clear RTS)
```

```
#define DTRBIT   0x10      (set/clear DTR)
#define SETEXT   0x80      (1 = set, 0 = clear)
```

If the SETEXT bit is set, then the RTS and/or DTR lines might be turned on. If the SETEXT bit is cleared, then either or both of these lines might be turned off.

TCSEXTW — Wait for the output to drain before setting the new parameters as in TCSEXT.

2.2.4 Other New Features

1. PC Exchange and RIOP's now work concurrently on the same RS-422 line.
2. A new RIOP memory query function allows reading the remote device's memory space. See *riopqry(1M)* and Section 5.5, *"Diagnostic Output"*, for a description.
3. The existence of the file **KRIOP** in the */etc/rcopts* directory is used to determine whether the RIOP driver is loaded and the RIOP daemon is executed at boot time. See Sections 4.4 and 4.5 for further discussion.
4. Applications that used the RIOP experienced a degradation of performance. The fix was to add **packet coalescing** to the driver. Most of the changes are to the vtcl_output routine. This routine determines when it is appropriate to coalesce, and calls the new coalesce() procedure to add the caller's packet to the packet held in a staging buffer. Two forms of coalescing are possible. The first is to add the new vtd_tpkt to the end of the vtd_pkt, the second is to move the data from the caller's vtd_tpkt to the existing vtd_tpkt in the staging buffer. The second mechanism is the preferred one, but will only be done if the last vtd_tpkt is the same type as the caller's. If the type of the caller's vtd_pkt is different from the staging buffer, the staging buffer will be sent, and the caller's packet set up as the new staging buffer.

3. Product Dependencies

3.1 Software Dependencies

The current release Remote I/O Processor as well as the compatible Enhanced Cluster Driver are incorporated into the 6.2 CTIX product. This Release for 6.2 CTIX is functionally equivalent to the RIOP subsystem in 5.25.3 CTIX. In other words it is derived from the same source code with minor porting differences. **This Release Notice is part of the bill of materials for 6.2 CTIX.**

3.2 Hardware Dependencies

The Remote I/O Processor Software works with the following systems:

S/120, S/22x, S/320, S/480, S/640

These systems must have the Two-port RS-422 Expansion board (60-00434) or Four-port RS-422 Expansion board (60-00371) installed in the I/O processor slot.

3.3 Memory Requirements

Since this product allows the connection of many users to an S/Series system, the amount of memory available on the host for application processing, up to some limit, will directly affect overall system performance. A rough approximation of this limit can be calculated by adding up the memory requirements of each application which must run concurrently. The data and stack space required for each process must be multiplied by the maximum number of concurrent users whereas the text or code portion is added only once if it was compiled for shared text usage.

The size of the kernel and loadable drivers also needs to be taken into consideration. Please see Section 5.1, *"Driver Loading"*, on driver loading to determine the memory used. The buffer allocation mechanism for RIOP communication uses approximately 5K of memory per RIOP port between 1 and 96 ports.

While using the optimum amount of memory will give the highest performance, many applications will operate satisfactorily in a smaller amount of memory by taking advantage of the virtual memory features of the system.

4. Configuring the System for RIOP's

Please read all sections of this document before attempting to configure and use RIOP's. The following sections describe steps required to configure CTIX for RIOP use.

4.1 Creating RIOP Devices

Three major devices[1] are used for RIOP operation. The first is the controlling major device which has two minor devices. These are:

```
crw-r--r--  1 root sys   67, 0   /dev/vtctl
crw-r--r--  1 root sys   67, 1   /dev/vtrdiag
```

The first of these minor devices is used by the *riopd* daemon program for downloading the RIOP's. The second minor device is used by the *riopstat(1M)* program as well as other diagnostic or administration programs wishing to query RIOP information from the kernel driver.

The other two major devices are used for the tty interfaces to the ports of the RIOP's. This allows for the use of 96 RIOP ports, 96 through each major device.

In 5.23 CTIX and beyond these device nodes reside in a subdirectory of **/dev** named **/dev/rtty** and are named **r000** through **r095.** These should appear as:

```
crw--w--w-  1 root sys   70, 0   /dev/rtty/r000
    :       :      :       :
crw--w--w-  1 root sys   70, 95  /dev/rtty/r095
```

These devices are mapped in a straight forward way to the ports of the RIOP's. The RIOP with the ordinal number of 0 in **/etc/riop/rtab** will be accessible through the first 16 devices of the above array. The RIOP with the ordinal number of 1 will be accessible through the second 16 devices of the above array,

1. See Section 4.4, *Enabling the RIOP Virtual Terminal Driver*

etc.

It is not necessary and not recommended to create the full array of device nodes above. This however means that new device nodes will need to be created later if the originally configured number of RIOP's is exceeded.

If this is the first installation of the product then this file will not exist and so the program queries for the number of RIOP's to be supported and creates a template **/etc/riop/rtab** file based on that number. It then continues by using this file as a guide to create the correct devices.

If this is the first time RIOP's have been configured on the system or you are supporting additional RIOP's then you will need to make the correct device nodes. The use of the program *riopcfg(1M)* is recommended for this purpose. To create the required control and tty entries in **/dev** run the RIOP configuration program:

/etc/riop/riopcfg -d -r 1.13

Since we have not passed explicit RIOP numbers to this program, it uses the file **/etc/riop/rtab** as a guide in making the tty devices.

If the program detects the presence of the directory **/dev/rtty** (which should exist in 5.23 CTIX or higher), it creates device nodes in this directory beginning with **r000,** otherwise it creates device nodes in **/dev** starting at **tty400.**

If two RIOP's were specified then the transaction would look something like the following:

```
Assume initial RIOP installation.
A template version of /etc/riop/rtab will be created.
How many RIOP's are to be supported? 2
RIOP #0: /dev/rtty/r000 through /dev/rtty/r015 made
RIOP #1: /dev/rtty/r016 through /dev/rtty/r031 made
```

Notice that the release level,"**1.13**", passed to this program is only required when **/etc/riop/rtab** does not exist. It is used to fill in the release level field of the template file. If

/etc/riop/rtab was created for a previous installation of RIOP, the last field should be changed to the current release level: **1.13**.

4.2 Inittab Administration

RIOP terminal login sessions are controlled in the same way as for other system tty ports. This is through the administration of the file **/etc/inittab** and the use of the *getty(1)* program. For each RIOP tty device described above a *getty* can be started by adding a line of the form:

rNNN:2:respawn:/etc/getty rtty/rNNN 9600

where NNN is the ordinal number of the port, for 5.23 CTIX and beyond. The installation should have added entries for the number of RIOP's that you specified but with the third field set to *off*. You must change these to *respawn* before login prompts appear at RIOP ports. Any legal baud rate can be specified here (see *termio(7)*).

If this is the first time RIOP's have been configured on the system or you are supporting additional RIOP's then you will need to make the correct **/etc/inittab** entries (assuming logins are desired). The use of the program *riopcfg(1M)* is recommended for this purpose. To create the required entries run the RIOP configuration program:

/etc/riop/riopcfg -i

Since we have not passed explicit RIOP numbers to this program, it uses the file **/etc/riop/rtab** as a guide in making the entries. This file should already exist (see section 4.1, "*Creating RIOP Devices*"). The program should respond with output like the following if two RIOP's are configured.

RIOP #0: 16 inittab entries made
RIOP #1: 16 inittab entries made

If the program detects the presence of valid entries for an RIOP from a previous installation, it will abort its processing for that RIOP only, but then continue for subsequent RIOP's.

4.3 Editing the RIOP Configuration File

The RIOP configuration file, **/etc/riop/rtab** is the file which defines what RIOP's are known to the system. Each entry in this table is one line with up to four ascii fields separated by a colon (':') which gives information about one RIOP in the system. A short example follows:

```
03fa:0:1.13: Computer Room
5e10:1:1.13: Marketing
0032:2:1.13: Technical Support
```

The first field is the unique identification number which is coded into an ID prom on the RIOP. In the RIOP this is a 4 byte number where the most significant byte is the product code which is always 0x20 for the RIOP. The **rtab** field should **only contain the lower 3 significant bytes of this number** (ie, the last 6 characters), and be expressed in hexadecimal. You can ignore leading 0's if you wish. For example, 0x2000013a could be entered as 13a, 013a or 00013a. Please see section 5.3 step 1, *"Steps to Add an RIOP"*, for details on obtaining the unique ID from the RIOP. This number can also be obtained from an error message in the file **/etc/riop/elog** when an unknown RIOP attempts to boot.

The second field is a decimal ordinal number for this RIOP. This field is used to order each RIOP from 0 to 5 such that RIOP #0 is related to the first group of 16 virtual tty's, RIOP #1 is related to the second group of 16 virtual tty's, etc. The numbers of this field in different lines of the file do not have to be sequentially ordered although it is recommended for ease of administration.

The third field is the version number suffix string which is appended to the string **/etc/riop/riop** by the RIOP daemon to form the full path name of the executable object file to be downloaded into the RIOP. For this Release, this field will always contain "**1.13**". This mechanism allows for the simultaneous use of multiple RIOP's operating at different download image release levels.

The fourth field is optional and can be used as a comment field.

This will allow a system administrator to comment on the physical location of a specific RIOP. It is started by a colon (:) and includes all text up to a following newline.

4.4 Enabling the RIOP Virtual Terminal Driver

To have the system automatically load the RIOP virtual terminal driver when it boots, you must create the zero length file **/etc/rcopts/KRIOP.** The script */etc/drvload(1M)* uses the presence of this file as a key to load the driver. This script then executes a line like the following:

./lddrv -a vtctl vtty0 vtty1 && echo "Riop-terminals Loaded."

The RS-422 board must also be installed so that the cluster driver will automatically be loaded. The cluster driver needs to be loaded in order for the RIOP driver to operate.

This should not be necessary, but check that the file **/etc/master** has the following entries uncommented (no leading '*'):

vtctl	1137	44	vtc	0	67	1
vtty0	37	44	vt0	0	70	256
vtty1	37	44	vt1	0	71	256

4.5 Enabling the RIOP Daemon

To have the system automatically start the RIOP daemon when it boots, you must create the zero length file **/etc/rcopts/KRIOP.** This causes the initialization scripts to start the download daemon running when it goes to multi-user states.

4.6 Editing the System File

A new variable, **vtd_defcnt** in the file **/etc/system** is used by the RIOP virtual terminal driver to determine how many tty and other structures to allocate in the kernel. This variable can be tuned to avoid wasting memory. Take the largest RIOP number which you intend to use in **/etc/riop/rtab** plus 1. Then multiply this number by 16 to get the optimal value for **vtd_defcnt.** The default value will be 256, good for sixteen RIOP's, if this line is left commented.

The variable **cl_defdrops** defines the maximum number of RIOP's (or 'drops') that you can have on one RS-422 line. If it is not set, the system will default to 8. You only need to change it if you wish to have more than 8 RS-422 devices (RIOP's, PCX PC's or PT/GT's) on one line.

To implement changes, edit the file **/etc/system** and remove the '*' from the front of the lines you wish to uncomment. Then change the number to the appropriate value.

You must reboot the system any time you edit this file for it's new values to take effect. If you are enabling RIOP's for the first time you should also reboot the system now.

5. Using the Product

The following sections contain information useful in administering RIOP's. Please read all of this material before attempting to use the product.

5.1 Driver Loading

The kernel virtual terminal driver used for RIOP communication is a loadable device driver which interfaces directly with another driver, ie. the enhanced cluster driver. As such it is necessary that this driver be loaded before the RIOP virtual terminal driver can successfully be loaded. To check that both drivers are correctly loaded, login as root and issue the following command:

/etc/lddrv/lddrv -s **RETURN**

A group of lines similar to the following should be contained within the display:

DEVNAME	ID	TYPE	BLK	CHAR	SIZE ...
cluster	2	DRIVER	-	66	0x8000 ...
			-	1	
			-	21	
			-	22	
vtctl	3	DRIVER	-	67	0x7000 ...
			-	70	
			-	71	

This indicates that both the cluster and virtual terminal drivers have been loaded.

It is possible to unload and reload one or both of these drivers as long as all processes using them have been killed and the cluster driver is unloaded last and reloaded first. **This operation is only recommended for expert users. It is safer to reboot the system if you need the drivers to be re-loaded. If you intend to unload the RIOP driver, please take all RIOP's out of service first by powering them down.** To reload, use the lines in **/etc/drvload** as a guide and be sure to restart the **/etc/riop/riopd** daemon.

5.2 RIOP Status

The program **/etc/riopstat** is actually the following one line shell script:

/etc/riop/riopcfg -s $1

Either of these forms can be used to determine the status of RIOP's in the system. Given no arguments this program will use the file **/etc/riop/rtab** as a guide and query the driver for the status of each RIOP specified there. For instance a system with four RIOP's might respond with the following output:

RIOP	State	Unique Id	Line	Drop	Ports
0	UP	4	0	0	8
1	UP	f	0	1	8
2	DOWN				
3	BOOTING	a	1	0	16

One argument can also be passed which specifies that a single RIOP number is to be queried and reported on.

5.3 Steps to Add an RIOP

The following steps should be taken in adding a new RIOP to a system which already supports one or more RIOP's:

1. Get the hexadecimal unique ID of the RIOP to be added. This might be done by powering on the RIOP with a terminal attached to port 0 at 9600 BAUD or to port 1 at 1200 BAUD. The RIOP will print out its unique ID. (Remember to only use the last 6 characters of the ID, ignore the **0x20**. See the section 4.3, "Editing the RIOP Configuration File".) An alternate method for identifying the Unique ID is to examine **/etc/riop/elog** for an entry "**id: 0xab not found, boot failed**", where **0xab** is an RIOP not listed in **/etc/riop/rtab.**
2. Add an entry with this unique ID and the next available RIOP number to the file **/etc/riop/rtab.**
3. Issue the command **/etc/riop/riopcfg -m n** where **n** is the RIOP ordinal number (the second field of the **rtab** entry added). This will report the range of tty devices

supported by this RIOP.

4. Check to see if these devices exist. If they do not then issue the command **/etc/riop/riopcfg -d n** where **n** is the RIOP ordinal number. This will create the necessary tty devices for the RIOP.

5. Take this step only if you desire to start *getty's* on the RIOP. Check to see if entries exist for the RIOP in **/etc/inittab.** If they do not then issue the command **/etc/riop/riopcfg -i n** where **n** is the RIOP ordinal number. This will create the necessary **inittab** entries for the RIOP. (NOTE: The last two steps might be folded into one by issuing the **d** and **i** options concurrently)

6. Physically connect the RIOP to one of the lines of the S/Series RS-422 board. To avoid disrupting other devices already in service, the RIOP should be added to the end of a daisy chain by connecting a cable between the currently terminated end of the chain and the RIOP, and then terminating the remaining plug of the RIOP.

7. Power on the RIOP. A terminal attached to port 0 at 9600 BAUD or to port 1 at 1200 BAUD can be used to act as an RIOP console. The RIOP should automatically request and receive its download image. It will print out a colon (':') when the download starts, followed by a series of dots ('.') when each download packet is received, followed by a semi-colon (';') when the download is complete. The *riopstat(1M)* program should now report that the RIOP is up.

8. If desired, edit **/etc/inittab** to turn on any *getty's.* Normally this is done by changing the field containing the word **off** to **respawn**. Give the command **telinit q**. The requested login's should be appear on RIOP terminals within 15 seconds.

5.4 Steps to Remove an RIOP

If the RIOP to be removed from service is in the middle of a daisy chain, physically disconnecting it might take all devices further down the chain out of service also. In this case take steps to warn users of these devices that they might be temporarily down.

The following steps should be taken to remove an RIOP from service:

1. Determine the ordinal number of the RIOP to be removed. It is recommended to label each RIOP physically with its unique ID so the ordinal number can easily be looked up in **/etc/riop/rtab.** Failing this, try to determine the tty device name of one of the terminals connected through it. This can be obtained by issuing the **tty** command at the terminal in question. Then use the **/etc/riop/riopcfg -m** command to get a list of all RIOP to device mappings in the system.

2. Warn all users on the RIOP in question that it will be taken out of service. See *ps(1),*whodo(1M) and *wall(1M).*

3. Turn off all *getty's* associated with this RIOP in **/etc/inittab.** Give the command **telinit q.**

4. Kill all processes associated with the devices attached to the RIOP. See *kill(1)* and *fuser(1M).*

5. Power down the RIOP and physically disconnect it. If it is in the middle of a daisy chain, be sure to reconnect the devices before and after it in the chain. The *riopstat(1M)* program should report that it is down.

6. If this is a permanent removal, delete the RIOP entry from **/etc/riop/rtab,** otherwise the entry can safely be left intact.

5.5 Diagnostic Output

The **/etc/riop/riopd** program will report non-fatal error conditions to the file **/etc/riop/elog.** If the daemon is running and you experience difficulty in downloading an RIOP, please read this file for the latest error messages. The daemon will always append to the end of this file unless it reaches a size of 10240 bytes in which case it will truncate it to 0 before writing.

It is possible to enable certain informational or error messages from the RIOP virtual terminal driver. If you are experiencing trouble with your system this can be helpful in helping to isolate the problem. To do this you must have the debugger loaded in your system. Type **^B** at the console to enter the debugger and then enable selective prints as follows:

```
> kp      RETURN
> kq v    RETURN
> pm      RETURN
> go      RETURN
```

Messages from the underlying cluster driver can also be enabled with the command:

```
> kq y    RETURN
```

This driver will occasionally print out a message reporting that it had to retransmit a message to a RS-422 drop. Many messages of this type might indicate a noisy line or a bad physical connection. Enabling this option during heavy RS-422 use might cause excessive printing that might interfere with the timing sensitive Cluster protocol.

For more information on how to use the kernel debugger please see the manual *Writing MightyFrame Device Drivers* (DAC-120). Although the material on S/Series drivers in this manual is somewhat out of date, the manual has the most complete set of information on the use of the kernel debugger, the interactive loader, and the kernel routines (section 2K).

If the RIOP reaches a panic condition, for instance, detecting a checksum error on its download program image then it will print the condition on its designated console and reboot.

The designated console will always default to port 0 at 9600 BAUD. This can be changed to port 1 at 1200 BAUD by hitting ^B from a port 1 terminal running at 1200 BAUD shortly after the RIOP prints out its version number and unique ID while rebooting. The RIOP will drop into its ROM debugger. Type **go** to continue operation with port 1 as the designated console.

The entire contents of an RIOP's memory can be uploaded into a CTIX file by issuing the following command:

```
/etc/riop/riopdump n > dump
```

where **n** is the RIOP number to be dumped.

6. Performance Considerations

This section is intended for those interested in details of RIOP performance. Skip to the Section 6.5, *"General Guidelines"*, if that is all you require. In the following discussion *raw* mode refers to that used by a typical screen based application where output postprocessing is turned off. The *cooked* mode is typical of interaction with a shell where output postprocessing is turned on.

6.1 External Processing

The measurements discussed in this section were done with the 1.11 RIOP Release.

The external processing feature of this Release allows for great increases in host CPU idle time for a given amount of output through an RIOP. (See Section 2.2.1, *"External Processing on the RIOP"*, for a general description of external processing.) For example, with 32 ports over four RIOP's in *cooked* mode on an S/320, around 22K chars/sec were delivered with the external processing feature enabled or disabled. The CPU idle time measured, however, went from around 2% to 95% when the feature was enabled. Similar increases in idle time were measured in both *cooked* and *raw* mode for both the S/320 and S/640. This feature should be used by those running applications which are CPU bound. This feature can be more critical for S/320 users, since less CPU cycles are available on slower machines.

Since the external processing feature moves character handling tasks from the host CPU to the RIOP CPU, it might also decrease total throughput slightly when the RIOP CPU becomes the bottleneck. For example, with eight RIOP ports in *cooked* mode on an S/320, around 6.8K chars/sec were delivered with the external processing feature disabled and only 5.5K with the feature enabled. The idle times measured were 73% and 99% respectively. So, for small numbers of ports or where idle times are not a critical factor, better throughput can be achieved with the external processing feature disabled.

The external processing feature also greatly improves the character echo latency since echo is done local to the RIOP

with this feature. This helps to improve the perceived performance when typing commands. This does not help screen based applications which do their own echoing.

The external processing feature does not affect the performance of other line disciplines such as SLIP or Shell Layers.

6.2 Eight-port Versus Sixteen-port RIOP's

In general it is not recommended to use more than eight RIOP ports concurrently. The expansion serial board with eight extra ports are provided for customers who have a need for the greater connectivity where all ports will not normally be in use at the same time. The following table shows output throughput of an example system configuration for Eight- versus Sixteen-port RIOP's.

tty Output Test, S/320				
RIOP ports	*total ch/sec*	*ave ch/ sec/port*	*% idle*	*proc mode*
8_8	6565	821	99.6	RX
8_8	5561	695	99.4	CX
8_8	6732	842	88.6	RN
8_8	6860	858	73.3	CN
16_8	5556	695	98.7	RX
16_8	4890	611	99.5	CX
16_8	6087	761	85.4	RN
16_8	6168	771	71.7	CN
16_16	9846	615	95	RX
16_16	6677	417	97	CX
16_16	6879	430	64.9	RN
16_16	7302	456	50	CN

M_N = M port RIOP, N ports tested
RX = raw mode with external processing
CX = cooked mode with external processing
RN = raw mode with no external processing
CN = cooked mode with no external processing

It can be seen that Eight-port RIOP's exhibit about 10% higher throughput than Sixteen-port RIOP's with only eight ports in use. This can be attributed to greater buffer space available per

port on the Eight-port version. The Sixteen-port RIOP with all 16 ports in use performs fairly well in raw mode with external processing on, but suffers in the remaining modes.

6.3 RIOP Distribution over RS-422

The performance results with RIOP's will vary somewhat depending upon their configuration on RS-422 lines. Testing has shown that the best performance results can be obtained when RIOP's are distributed evenly over 4 RS-422 lines. Even so, the following results show that the impact of putting many RIOP's on one RS-422 line are significant only when the external processing feature is turned off.

Eight-port RIOP's, S/640				
RS-422 lines	*total ch/sec*	*ave ch/ sec/port*	*% idle*	*proc mode*
4	39073	813	93.6	RX
4	33052	688	94.8	CX
4	37353	778	53.9	RN
4	37829	788	15	CN
1	37819	788	94	RX
1	33065	689	92.6	CX
1	26570	554	14.2	RN
1	26950	561	6.3	CN

6.4 Mixing Bare Serial and RIOP Ports

The current architecture of the S Series does not allow for use of both the RS-422 and IOP cards. This means that when RIOP's are used, the base ports will have no I/O support and all interrupts will go directly to the main CPU. Therefore some care should be used in making sure that base port use is kept to a minimum when using RIOP's. It is recommended that heavy bare base serial port use be limited to 8 ports on an S/640 and 4 ports on an S/320.

6.5 General Guidelines

The following guidelines are recommended for achieving good performance results with RIOP's.

1. Eight-port RIOP's are recommended for most applications. Sixteen-port RIOP's are suitable where high

connectivity is required or where throughput requirements are moderate. In other words, it is not recommended that more than 8 ports be used simultaneously for applications requiring high screen repainting rates. In this case 2 Eight-port RIOP's should be substituted for 1 Sixteen-port RIOP.

2. Base serial port use should be kept to a minimum when using RIOP's.

3. The external processing feature should be enabled for CPU intensive applications.

4. RIOP devices should be distributed as much as possible between the available RS-422 lines.

The following test results show the output and idle times for the rated number of users when these guidelines are followed. Each test used Eight-port RIOP's distributed over 4 RS-422 lines.

Eight-port RIOP's, S/320				
RIOP ports	*total ch/sec*	*ave ch/ sec/port*	*% idle*	*proc mode*
8_32	25165	786	77.3	RX
8_32	21807	681	94.7	CX
8_32	24672	771	38.7	RN
8_32	22358	699	1.7	CN

Eight-port RIOP's, S/640				
RIOP ports	*total ch/sec*	*ave ch/ sec/port*	*% idle*	*proc mode*
8_64	51657	807	90.0	RX
8_64	43935	686	91.3	CX
8_64	45273	707	22.7	RN
8_64	44766	699	0	CN

7. Known Errors, Warnings, and Restrictions

7.1 Warnings

1. When using RIOP's for any type of machine to machine communication flow control in both directions should be enabled. Also no more than 4 lines at 9600 BAUD or 2 lines at 19200 BAUD will accept sustained input without incurring occasional lost characters due to receiver overruns.

2. When switching the use of a RS-422 line from one kind of device to another **you must reboot the S/Series system.** It is possible for PT's which have not received logins to steal station addresses while awaiting a download. This could restrict the number of station addresses left for use, and therefore the number of drops which can be added to the line.

7.2 Restrictions

1. Shell layers has been converted to STREAMS. It does not work now on RS422 ports nor on RIOP ports.

2. The current maximum number of RIOP's on a single RS-422 line is 15.

3. The current version of WGS is not fully supported by 1.13 RIOP. When using PT or GT terminals over the RIOP, the windows will often be trashed.