Ciprico
Reference Manual

Rimfire 3500

VMEbus SCSI Host Adapter

Sun® End User
Installation Guide

Publication Number 21018002
January 12, 1990
© 1990 by Ciprico Inc.
2955 Xenium Lane
Plymouth, MN 55441
(612) 559-2034

®*Sun is a registered trademark of Sun Microsystems.*

© 1990 by Ciprico Inc.
2955 Xenium Lane
Plymouth, MN 55441
(612) 559-2034

All rights reserved. No part of this publication may be reproduced or transmitted in any form, or by any means, electronic or mechanical (including photocopying and recording), or by any information storage or retrieval system, without the permission of Ciprico, Inc.

Written for Ciprico by Robin Russell

Printed in the United States of America

Preface

This guide is intended to assist in the installation of Rimfire® 3500 Series VMEbus SCSI host bus adapters in Sun® Microsystems Workstations. Unless otherwise specified, references to the Rimfire 3500 adapter indicate any of the mentioned models.

The following items are recommended for successful hardware and software installation:

Equipment:

- Sun-3 or Sun-4 Workstation with a tape drive (1/4 inch or 1/2 inch) or disk drive (5-1/4 inch)
- Rimfire 3500 Series VMEbus SCSI host bus adapter
- SunOS distribution tapes
- SCSI drives and cabling
- Ciprico's SunOS driver

References:

- Sun Microsystems' documentation and reference manual(s) for the appropriate Sun Workstation
- The appropriate drive manufacturer's reference manual(s)

®*Ciprico and Rimfire are registered trademarks of Ciprico Incorporated. Sun is a registered trademark of Sun Microsystems, Inc.*
™ *Maxtor is a trademark of Maxtor Corporation.*
™ *Micropolis is a trademark of Micropolis Corporation.*
™ *Miniscribe is a trademark of Miniscribe Corporation*
™ *Sun-3, Sun-4, Sun Microsystems, Sun Workstation, SPARC, and SunOS are trademarks of Sun Microsystems, Inc.*
™ *UNIX is a trademark of Bell Telephone Laboratories, Inc.*
™ *Wangtek is a trademark of Wang Laboratories, Inc.*
™ *Wren is a trademark of Control Data Corporation.*

Revision History

Publication #	Revision	Date	Description
21018000	B	10.13.89	Class 'B' Manual that contains **preliminary** Rimfire 3523 information, as well as Rimfire 3500 SunOS Device Driver information.
21018001	A	12.01.89	Addition of updated installation script information for bootable and non-bootable sections, and updated SunOS Device Driver information.
21018002	A	01.12.89	Minor update to SunOS Device Driver manual installation information.

Table of Contents

Figures

Tables

Section 1 - Introduction

The Ciprico Rimfire 3500 Host Bus Adapter is a high performance dual function SCSI host bus adapter and floppy disk controller for the VMEbus. Including the Rimfire 3500 itself, eight SCSI bus devices (target ID values) are supported. With the optional floppy-disk controller, up to four ANSI X3.80-1981/SA-450-compatible floppy-disk drives are supported. SCSI bus data rates of 2.0 Mbytes/second in asynchronous mode and 5.0 Mbytes/second in synchronous mode are supported, while the floppy port can support data rates of 250 kHz and 500 kHz. System bus transfers can be sustained at SCSI bus data rates with burst rate capability of up to 20 Mbytes/second with minimum memory response time, or 30 Mbytes/second using block mode transfers.

One key to the Rimfire 3500 adapter's excellent system bus performance is the Short Burst FIFO (SBF). The SBF is a proprietary gate array developed by Ciprico and used to interface system buses and on-board data paths. The gate array is 32 bytes deep, 32 bits wide, and contains built-in byte and word swapping logic. The VMEbus interface is also 32 bits wide and supports 32-bit addressing.

SCSI bus support of the Rimfire 3500 adapter includes the use of pass-through commands, providing flexibility in the use of features available from different peripheral vendors. SCSI commands are passed to the board via a system memory resident circular command queue. This technique allows the system to add commands to the queue as soon as they are ready, with no timing restrictions. The adapter can extract multiple commands from the queue for processing; allowing it to overlap SCSI bus operations using the disconnect/reconnect feature of the SCSI bus. Connections to the bus are through the faceplate connectors and support either single-ended or differential bus transceivers.

For applications requiring cost effective floppy support, the Rimfire 3500 adapter provides an optional floppy port. A local memory buffer for floppy data allows concurrent operations on the SCSI bus and floppy port without degrading SCSI performance.

The Rimfire 3523 Sun-bootable SCSI adapter operates as a Rimfire 3500 SCSI-only board with Sun system-boot capability. It emulates the Sun 472 1/2-inch tape controller to boot SCSI 1/4-inch tape devices, and it emulates the Sun 451 SMD controller to boot SCSI direct access devices.

WARNING
If Sun 472 emulation is enabled, an attached tape unit requires a tape to be loaded when booting.

Section 2 - Bootable Adapter Installation

This section describes procedures for installing the Rimfire 3523 bootable adapter in a Sun Microsystems' workstation. The information may vary between Sun Workstations and versions of SunOS.

This section contains references, values, and file names (for example *sunX* or *4.X*) that represent the Sun system and version of SunOS you are using. When *X* is used, it should be replaced by the version number of your software or hardware. In addition, examples in this section include values and file names that may differ from the display on your screen due to variations in system configuration.

In this section, **bold** type indicates a system-dependent variable.

Hardware Installation

This section describes installation of the Rimfire 3523 bootable adapter in Sun Microsystems' workstations. Installation procedures are dependent on the model of Sun workstation.

Required Equipment:

- Sun 3, Sun 4, or SPARC Workstation
- Rimfire 3523 VMEbus SCSI adapter

To perform hardware installation, UNIX must be shut down and the Sun system powered off. Because there is exposed line voltage, **disconnect the power cord from the system.** Be sure to control the static electricity.

The following diagram illustrates Rimfire 3523 addressing. Suppose the first Rimfire 3523 adapter is to be the primary adapter used for booting. There would also be a second Rimfire 3523 adapter that may be used for adding devices. The adapters would be addressed as follows:

	Disk Emulation Address	Tape Emulation Address	Rimfire Disk Address	Rimfire Tape Address
1st Rimfire 3523:	ØxEE4Ø	ØxEE6Ø	Øx5ØØØ	Øx5ØØØ
2nd Rimfire 3523*:	N/A	N/A	Øx55ØØ	Øx55ØØ

** The second Rimfire 3523 adapter (at address Øx55ØØ) is for adding devices. It is not bootable.*

Figures 2-1 and 2-2 show the Rimfire 3523 jumpers.

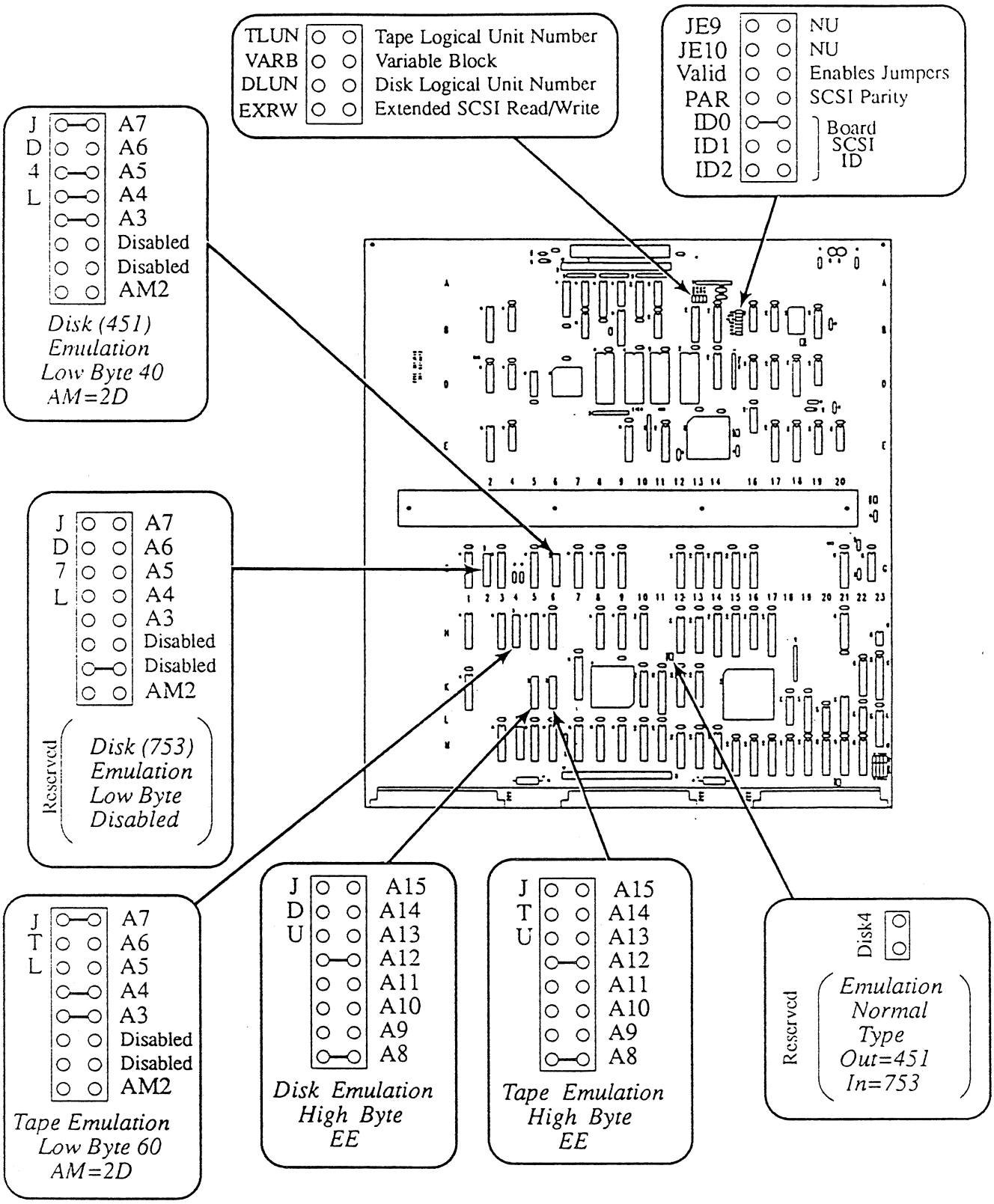


Figure 2-1 Rimfire 3523 Emulation Jumpers

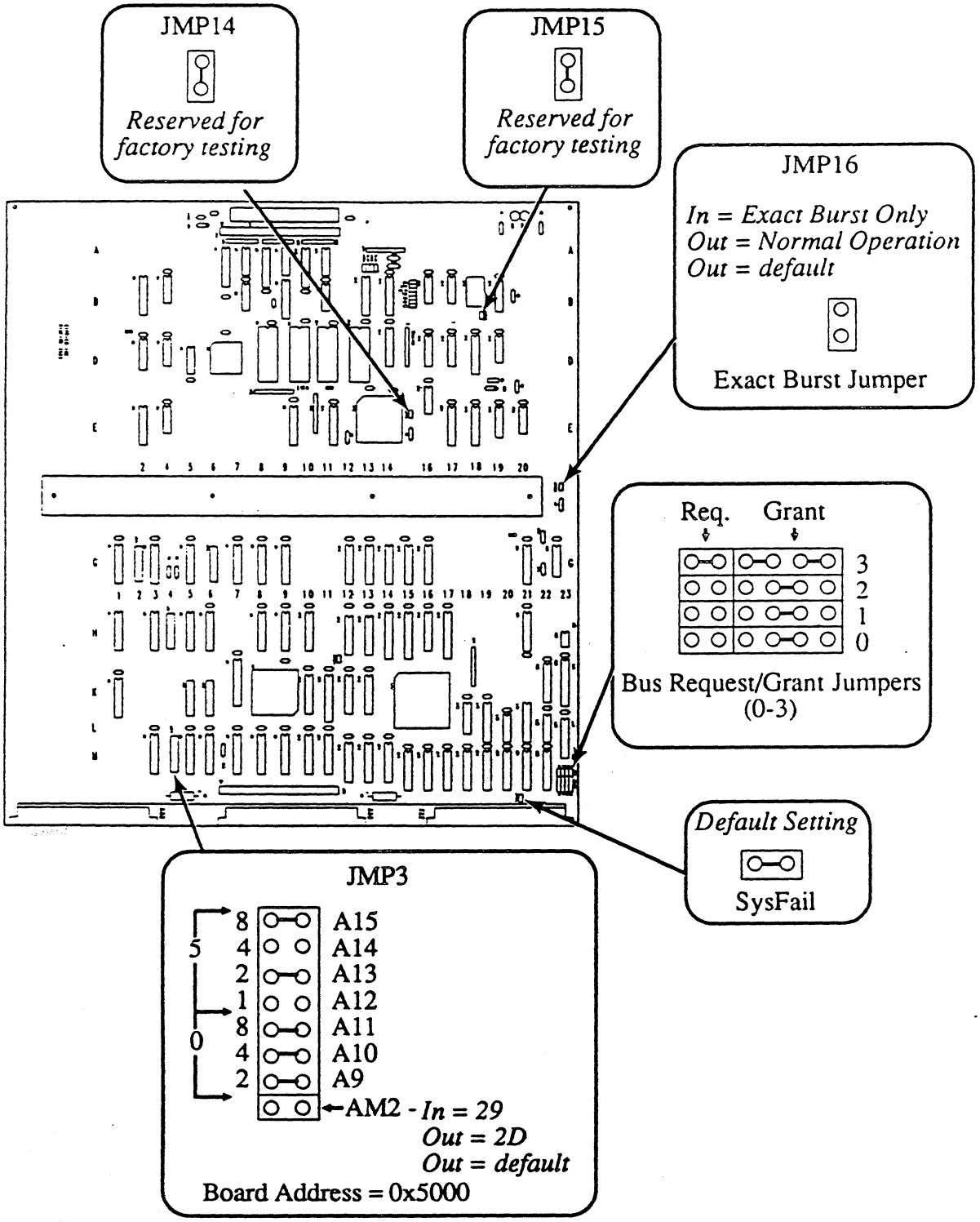


Figure 2-2 Rimfire 3523 Jumpers

Board Configuration

Figures 2-1 and 2-2 illustrate jumper locations and their factory settings. Unless otherwise indicated, a jumper is set to Ø if the jumper is in, and set to *1* if the jumper is out. Inspect your Rimfire 3523 adapter for proper jumper settings. Pay particular attention to the following items:

- The disabled jumper for *JTL* ,and *JD4L or JD7L*, must be out for the Rimfire 3523 adapter to emulate a Sun boot adapter.
- The address jumpers at *JMP3* (A9-A15) set the Rimfire 3523 address. The Rimfire 3523 address is dependent on whether the adapter is the first (address = Øx5ØØØ) or second (address = Øx55ØØ) Rimfire adapter in your system.
- The combined settings of the address jumpers at *JD4L* (A3-A7) and *JDU* (A8-A15) set the boot disk emulation address.
- The combined settings of the address jumpers at *JTL* (A3-A7) *and JTU* (A8-A15) set the boot tape emulation address.

Following are descriptions of the emulation control jumpers:

ID2-IDØ	Specifies the SCSI ID for the adapter. Jumper ID Ø gives default ID 6.
PAR	When this jumper is in, SCSI bus parity checking is enabled. When this jumper is out (the default), SCSI parity is disabled.
Valid	When this jumper is in, jumpers *TLUN, VARB, DLUN*, and *EXRW* are enabled. When this jumper is out, the adapter firmware determines the proper configuration specified by these jumpers for the attached peripherals.
NU	Reserved
EXRW	When this jumper is in, SCSI direct-access devices use 2AH/28H SCSI R/W commands. When this jumper is out, SCSI direct-access devices use Ø8H/ØAH.
DLUN	When this jumper is in, *xy1* (the second boot emulation disk) is SCSI LUN 1 of target Ø. When this jumper is out, *xy1* is SCSI LUN Ø of target 1.

VARB When this jumper is in, SCSI tape commands are variable block. When this jumper is out, SCSI tape commands are fixed block.

TLUN When this jumper is in, *xt1* (the second non-bootable emulation tape) is LUN 1 of target 4. When this jumper is out, *xt1* is LUN Ø of target 5.

➳ *NOTE: The first boot disk xyØ is SCSI LUN Ø of target Ø . Boot tape xtØ is SCSI LUN Ø of target 4.*

Adapter Installation

1. VME card slots on the Sun workstation are covered by metal plates. Inside each slot, are EMI plates. Select an unused slot and remove the two hex head screws holding the cover plate on the rear of the system. Remove the cover plate and slide the EMI plate out of the slot.

2. Rimfire 3523 adapters are Sun-sized cards and will fit directly into the card cage. Insert the Rimfire 3523 adapter into the slot, pressing it firmly into the connectors.

3. Fasten the adapter into place with the hex head screws from the rear cover plate.

4. Remove the front cover plate from the workstation to allow access to the backplane. On some Sun Workstations, you will also need to move the power supply to access the backplane. If so, remove the four screws holding the power supply cover and tilt open the power supply. Others may have a small removable panel allowing access to just the jumper area of the backplane.

WARNING
When the power-supply cover is removed, line voltage is exposed.

5. Locate the slot being used. (The number is to the right of the connector.) Remove the *BUS GRANT 3* and *IACK* jumpers (it is a good idea to simply move the jumpers down one pin so they are available, if needed). Figure 2-3 illustrates the *BUS GRANT* and *IACK* jumpers for a given slot.

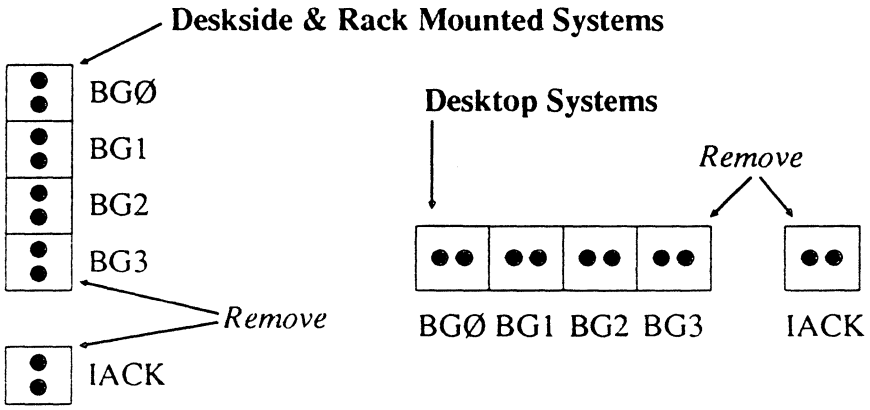


Figure 2-3 BUS GRANT and IACK Jumpers

➠ *NOTE: The BUS GRANT 3 and IACK jumpers must be removed for the Rimfire 3523 adapter to operate properly.*

6. If you moved the power supply to access the backplane, tilt the power supply back to its original position and refasten the power supply cover.

7. Replace the front cover plate (if there is one on your particular system).

8. Connect the drive cables. To set up the Rimfire 3523 to connect to the Sun internal SCSI drives, place the board in the slot with the SCSI cable on the P2 connector (slot 7 TYP.) Add a 15-inch cable from J1A to JP2. If you are using **both** the internal and external SCSI cables, remove terminators RP1, RP2, and RP3. Verify that the terminators are installed on the devices at the end of both cables. Refer to Appendix F for information about cables.

If you are using only the internal cable **or** the external cable, leave terminators RP1, RP2, and RP3 installed. Figure 2-4 illustrates cable connection.

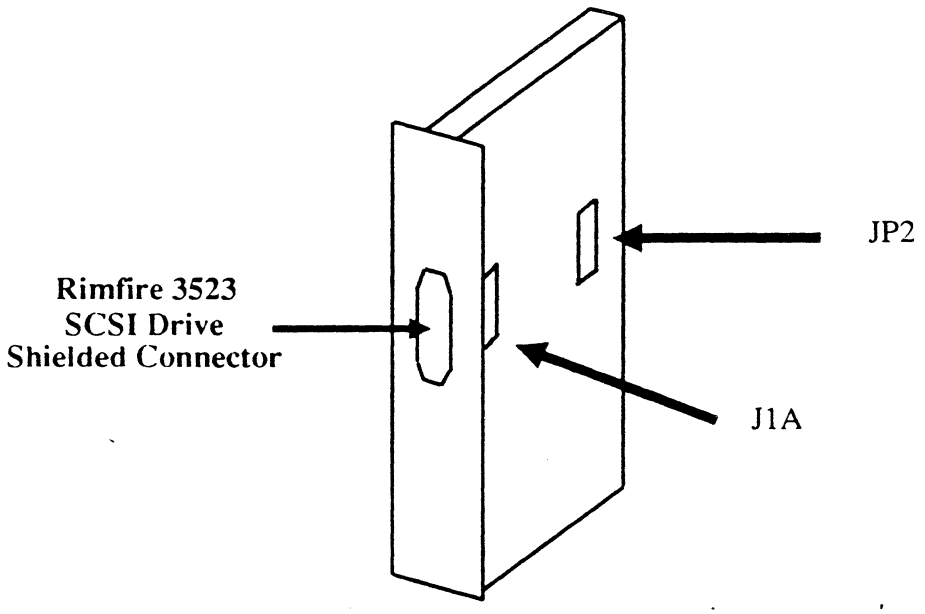


Figure 2-4 SCSI Drive Cable Connection

9. Check the drives to ensure that drive parameters (addressing, terminators, sector switches, etc.) are correct. Consult your drive manufacturer's manual for recommended drive settings.

10. Power on the Sun system and drives, and check for proper operation. If the board is operating correctly, the Fail (red) and Busy (green) lights flash and turn off.

STOP
This concludes the hardware installation procedure for the Rimfire 3523 bootable SCSI adapter. For information about software installation continue with this chapter.

Software Installation

The remainder of this section describes steps for installing *SunOS* and the Rimfire driver.

The examples and procedures shown assume you are using 1/4 inch tape and installing the Rimfire 3523 adapter as the primary boot adapter.

➽ *NOTE: Because most SCSI disk drives are pre-formatted, formatting and verifying drives may not be necessary. However,* ***if you format the drive, you must use the Ciprico standalone utility.***

If you are installing the Rimfire 3523 adapter as a non-bootable adapter, see Section 3 for installation procedures.

References:

- Sun Microsystems' documentation and reference manuals for the appropriate Sun Workstation
- The appropriate drive manufacturer's reference manual

Required Equipment:

- Sun 3, Sun 4, or SPARC Workstation with 1/4-inch tape drive, or access (through a network) to a 1/4-inch tape drive
- Rimfire 3523 SCSI adapter
- SunOS distribution tapes
- Ciprico's *Utility and Installation* tape (includes the Ciprico driver)

Loading the SunOS Boot

1. Power on the system.

2. After the memory check is complete, abort the boot process.

➽ *NOTE: The keys used to abort the boot process will vary with the keyboard you are using.*

3. Insert tape 1 of the Sun Operating System release tapes into the tape drive.

4. At the monitor prompt, load the bootstrap program from the tape.

   Enter the following information:

```
b xt()
```

5. After the boot prompt appears on the screen, remove the SunOS release tape.

6. When the Rimfire 3523 is in emulation mode, or when a Rimfire 35XX is running a standalone program, it expects the target IDs and device types shown in this table:

Unit	Device Type	Target ID	LUN
0	disk	0	0
1	disk	1	0
2	disk	2	0
3	disk	3	0
4	tape	4	0
5	tape	5	0

   Ensure that your devices are set to the proper target IDs.

Loading and Configuring the Standalone *cs35ut*

1. Insert the Ciprico Utility and Installation tape into the tape drive.

2. Load the Rimfire 3500 Standalone cs35ut program. The characters you enter to load the program vary with the system you are using. For example, enter xt(0,0,1) if your system is 68020 based, xt(0,0,2) if your system is SPARC (Sun 4) based, and xt(0,0,3) if your system is 68030 based. Enter the appropriate characters for the system you are using.

3. After the Standalone *cs35ut* program is loaded, the following device table showing the possible device configurations appears:

Unit	Device Type	Target ID	LUN
0	disk	0	0
1	disk	1	0
2	disk	2	0
3	disk	3	0
4	tape	4	0
5	tape	5	0

When the Rimfire 3523 is in emulation mode, it expects the configuration, with the target IDs and device types, shown in the foregoing device table. It is possible to boot from Unit 0, Unit 1, Unit 4 or Unit 5. This configuration only needs to be followed for devices that are intended for boot devices, or other devices that you wish to run with the Standalone utility. The user is prompted to hit the Enter key when done reading the information.

4. Next a device manufacturer menu is displayed, allowing the user to enter in a device maufacturer for each unit. The device manufacturer is used to determine special unit options for each unit. If a device manufacturer is not in the menu for your particular device, just enter the selection for "Other". If there is not a device connected for the unit number you are prompted for, just enter the selection for "No device present". If "Other" is selected, 0 is the unit options value for that device. See Appendix C for more information.

In the following example, the user has a Hewlett Packard disk as Unit 0, and an Archive tape drive for Unit 4. Therefore, the Hewlett Packard drive must be set for target id 0, and the Archive tape drive must be set for target id 4.

```
                    Device Type Menu
1.  Micropolis.
2.  Maxtor
3.  Hewlett Packard
4.  Miniscribe
5.  Fujitsu
6.  Wren
7.  WrenV
8.  Wangtek
9.  Archive
10.  Other
11.  No device present

Enter the number from the menu above
                for Unit 0: (disk)  3
                for Unit 1: (disk)  11
                for Unit 2: (disk)  11
                for Unit 3: (disk)  11
                for Unit 4: (tape)  9
                for Unit 5: (tape)  11
```

5. Next a prompt requests the hexadecimal address of the Rimfire 3523 adapter. Enter the hexadecimal address of the Rimfire 3523 adapter and press the Enter key. This address is 5000 for the first adapter and 5500 for the second adapter.

6. A prompt requests the unit that is to be worked with. Enter a unit number from the table above.

➳ *NOTE: If you have selected a disk that has not been previously formatted using a Rimfire 35XX series adapter, the message* **The disk may not exist or be formatted** *appears. Disregard the error message and proceed to the next step.*

7. A prompt (TERM =) requests the type of terminal that will be used. Some common terminal types are listed below.

   adm3
   adm3a
   sun
   sun24
   tvi925
   tvi9206
   vt52
   vt100

8. Enter the appropriate type.

9. After specifying the type of terminal, the main menu appears. Create a label, and format and verify the drives using the procedures in the following section.

Creating a Disk Label

1. By default, the unit you selected in step 6 of "Loading and Configuring the Standalone cs35ut" section is opened. The following menu appears:

```
Open Device:  /dev/rrs0a                    Size:  16320
cs35ut>

Available Disk Commands:
a     Start Device           o     Open a new device
b     Debug control          q     Quit
c     Identify Controller    r     Read and Display Label
d     Read Capacity          u     Stop Device
f     Format Disk            v     Verify Disk Format
l     Choose Label           w     Write Label to Disk
m     Map Sectors
```

   If you want to open a different drive, select O from the main menu and enter /dev/rrsXa (for disk) or rrtX (for tape), where X indicates the drive number (0 = Unit 0, 1 = Unit 1, 2 = Unit 2 ...).

2. After opening the proper drive, select L from the main menu to display the options for choosing a label. You have the option of editing the current label in the kernel (which would start out to be all 0's), or getting the disk geometry for the label from the disk through a mode sense command. If you choose to get the disk geometry from the disk and it

fails, re-issue the L command and select 0 from the list (edit current label) and then enter the configuration of the drive. The Gap1 and Gap2 sizes will always be zero for SCSI devices. Refer to the drive manufacturer's manual for further details.

3. Press the return key to advance to the partition information. A partition table similar to the one shown below will be displayed on the screen.

```
                Rimfire 35XX cs35ut Partition Table
             Starting     Ending              Size
Partition    Cylinder     Cylinder    Blocks      Mbytes     Overlaps
    a           0           77        32994       16.11        c
    b          78          174        41031       20.03        c
    c           0         1386       586701      286.48      a b g h
    d           0            0            0        0.00
    e           0            0            0        0.00
    f           0            0            0        0.00
    g         175          755       245763      120.00        c
   h*         756         1386       266913      130.33        c

  Disk Capacity:  286.48            Total Cylinders:  1387
```

You will be prompted to enter the free-space hog partition (represented by the asterisk (*). A free-space hog partition is the partition that contains the remaining cylinders at the end of the disk that are not accounted for. The number of cylinders contained in this partition changes as other partitions are changed. Enter the partition letter, or n for no free hog, to the statement below:

```
Enter the free-space hog partition [n<one>]:
```

Next, enter this partition-to-change statement:

```
Partition to change, <CR> when done:
```

The partition that you selected will be displayed on the bottom of the screen with a statement requesting you to enter the starting cylinder as shown below.

```
Partition a
Enter the starting cylinder (0-1386, n<ext cylinder>, q<uit>):
```

Next, you will be prompted for the size of the partition in Megabytes:

```
Partition a
Enter the size in megabytes (0-287, r<est of the disk>, q<uit>):
```

After entering the size of the partition, the number of blocks will be displayed along with the size in megabytes for the value entered. The actual size in megabytes may be slightly larger as the partitions must start and end on cylinder boundaries. Enter the '+' or '-' keys to add or subtract a cylinder to/from the partition. Enter 'y' if the partition is as desired, and enter 'n' to start over for that partition.

```
Partition a
32994 blocks (16.11 Mbytes) OK (y/+/-/n) ?
```

Repeat this procedure for each partition to change and enter <CR> when done.

```
Partition to change, <CR> when done:
```

Next, to write this partition table to the kernel, enter *y*.

```
Enter 'y' to write this partition table to the kernel:
```

4. If you need to change parameters for the selected label, select L from the main menu, select 0 (edit the current label), and then press Enter to display parameters for the current label.

➠ *NOTE: Because most SCSI disk drives are pre-formatted, formatting and verifying drives may not be necessary.*

5. If you do **NOT** need to format your drive, enter a W (Write Disk Label) command from the main menu to write the disk label to the disk.

Formatting and Verifying Drives

If you need to format your drive, perform the steps in this section.

1. Select the f (format the drive) command from the main menu. The format command will first prompt the user for information about reading the defect list. The user can format without the defect list, use only the manufactures defect list, use the grown defect list, or use both the

manufactures and grown defect list while formatting. Next the user will be prompted for an interleave value. In MOST cases the user should enter a 1 for the interleave value. Then the estimated time for the format is displayed giving the user a chance to back out of the format. If the user decides to proceed, a "Formatting:" message is displayed and when complete, the word "Done" will appear after "Formatting".

2. Enter *y* to write the disk label to the disk.

3. Select the v (verify format) command to verify the disk integrity. Ciprico suggests five passes of verification. A prompt appears asking whether *cs35ut* and the adapter should map bad blocks. Enter *y* to map bad blocks. Generally, the entire disk should be verified. Refer to the v command description in Appendix B for further details.

4. Once the verify has completed, type *q* to exit the *cs35ut* program. The disk is ready to store data.

Loading the SunOS

1. Insert the first SunOS release tape into the tape drive. Type *b xt()* and press Enter to display the boot prompt (*boot:*).

2. Perform the SunOS installation using *SETUP* (for SunOS3.X) or *SUNINSTALL* (for SunOS4.X). For further details refer to the following section in your Sun manual:

   - If you are using *SETUP*, refer to the *"Loading the Miniroot"* section.
   - If you are using *SUNINSTALL*, refer to the *"Loading the Mini UNIX"* section.

   Always refer to the Ciprico board as *xy* (for disk commands) or *xt* (for tape commands) during the installation.

   The system name assigned during the SunOS installation will be used as the *Hostname* when installing the Ciprico driver.

3. After completing the normal SunOS installation, reboot the system by entering *b xy()* at the monitor prompt.

WARNING
If Sun 472 emulation is enabled, an attached tape unit requires that a tape be loaded when booting.

Installing the Ciprico Driver

The Ciprico driver can be installed manually or by using the *Installation Script* that is included on the distribution tape. It is described in the remainder of this chapter.

If you select to manually install the Ciprico driver, refer to Appendix G.

Using the Installation Script

The *Installation Script* is included on Ciprico's *Utility and Installation Tape*. It is an interactive utility that steps you through the procedures for installing the Ciprico driver.

➽ *NOTE: In the following procedures, you are asked if it is OK to copy files to existing directories. To avoid overwriting existing files, filenames are automatically assigned a .nocf extension.*

Throughout the following procedures, there are messages indicating whether the particular step is mandatory to driver installation. Steps noted as mandatory must be performed for successful driver installation.

Due to system variations, the information displayed on your screen (file names, software and hardware references, device counts, etc.) may differ from examples in this manual.

If the install script is interrupted because of an error, you are given a series of choices of what to do next. If you want to re-enter the program, after making the appropriate changes, choose that option, make the changes, then enter **fg** *to re-enter the program.*

1. Create a directory (called *CIPRICO*) for the files on Ciprico's *Utility and Installation* tape. Enter the following line:

```
mkdir /sys/CIPRICO
```

2. Enter the following command to change to the directory you just created:

```
cd /sys/CIPRICO
```

3. Insert the Ciprico *Utility and Installation* Tape. If you are using 1/4-inch tape connected to the Rimfire 3523 adapter, read the tape by entering the following command:

```
tar xvbf 126 /dev/rmt0
```

If you are using 1/4-inch tape connected to a Sun internal drive, read the tape by entering this command:

```
tar xvbf 126 /dev/rst8
```

If your system uses a high-density tape drive, you can read the tape as /dev/rst0.

4. Enter the following command to switch to the *cf/install* directory:

```
cd cf/install
```

5. Start the installation script by entering the following command:

```
doinstall cf
```

```
(C)  C O P Y R I G H T   1 9 8 9

Ciprico, Inc.
2955 Xenium Lane
Plymouth, MN  55441
(612) 559-2034

All Rights Reserved.

+ Determining CPU architecture... CPU is a sunX
+ Determining hostname... Hostname is Rimfire
+ Checking for mount of /usr... /usr is mounted.
+ Trying to find system files
+ You are running SunOS release 4.X
+ Your system configuration directory should be RIMFIRE
+ System configuration file is /sys/sunX/conf/RIMFIRE
```

➠ *NOTE: The Hostname (indicated by RIMFIRE) will be the system name assigned during SunOS installation.*

```
[The next step is mandatory for driver installation]

About to copy cf device driver files to /sys/sundev and
/sys/sunX/OBJ. OK? (y/n) y
```

6. Enter *y* to copy the cf device driver files to /sys/sundev and /usr/unclude/sundev. The following messages appear:

```
+ cp ../sundev/cs35.c /sys/sundev/cs35.c
+ cp ../sundev/cs35err.h /sys/sundev/cs35err.h
+ cp ../sundev/cs35err.h /usr/include/sundev/cs35err.h
+ cp ../sundev/cs35flp.h /sys/sundev/cs35flp.h
+ cp ../sundev/cs35flp.h /usr/include/sundev/cs35flp.h
+ cp ../sundev/cs35if.h /sys/sundev/cs35if.h
+ cp ../sundev/cs35if.h /usr/include/sundev/cs35if.h
+ cp ../sundev/cs35int.h /sys/sundev/cs35int.h
+ cp ../sundev/cs35int.h /usr/include/sundev/cs35int.h
+ cp ../sundev/cs35io.h /sys/sundev/cs35io.h
+ cp ../sundev/cs35io.h /usr/include/sundev/cs35io.h
+ cp ../sundev/cs35lib.c /sys/sundev/cs35lib.c
+ cp ../sundev/cs35prm.h /sys/sundev/cs35prm.h
+ cp ../sundev/cs35prm.h /usr/include/sundev/cs35prm.h
+ Copy of cf driver files done.

[The next step is mandatory for driver installation]

About to modify /sys/sun/conf.c. OK? (y/n) y
```

7. Enter *y* to modify the /sys/sun/conf.c file. The following lines appear:

```
+ Beginning to add cf entries to /sys/sun/conf.c.
+ Making backup copy of /sys/sun/conf.c as
/sys/sun/conf.c.nocf.
+ Adding defines to /sys/sun/conf.c.
+ Counting block device entries...24 found.
+ Counting character device entries...71 found.
+ Adding bdevsw entry to /sys/sun/conf.c.
+ Adding cdevsw entry to /sys/sun/conf.c.
+ Checking for new entries in /sys/sun/conf.c.

+ Successful cf device description addition to
/sys/sun/conf.c

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/files. OK? (y/n) y
```

8. Enter *y* to modify the /sys/sunX/conf/files file. The modifications will be made and a backup copy (files.nocf) will be written. The following lines appear:

```
+ Beginning to add cf entries to /sys/sunX/conf/files.
+ Making backup copy of /sys/sunX/conf/files as
/sys/sunX/conf/files.nocf.
+ Adding configuration lines to /sys/sunX/conf/files.
+ Checking for new entries in /sys/sunX/conf/files.

+ Successful cf device addition to /sys/sunX/conf/files

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/RIMFIRE. OK? (y/n) y
```

9. Enter *y* to modifiy /sys/sunX/conf/RIMFIRE. A backup copy (/sys/sunX/conf/RIMFIRE.nocf) will be made. The following lines appear:

```
+ Making backup copy of /sys/sunX/conf/RIMFIRE as
/sys/sunX/conf/RIMFIRE.nocf.
+ Extracting controller information from
/sys/sunX/conf/RIMFIRE...done.
+ No cfc controllers currently installed.
+ Adding controller cfc0 to /sys/sunX/conf/RIMFIRE

[Note: Controller address MUST NOT be the same or overlap]
[any existing devices specified in file
/sys/sunX/conf/RIMFIRE.]
[Default value is displayed in braces below]
[Please be sure to preface hexidecimal number with "0x" ]
```

10. Enter the address of the Rimfire 35XX adapter and hit the return key, or just enter the return key if the address of the adapter is 0x5000. Then enter the interrupt vector of the controller, or just hit the return key if the interrupt vector 0xFE is ok.

```
VME address of controller? [0x5000]
VME interrupt vector of controller? [0xFE]

Add controller cfc0 at address 0x5000 vector 0xFE to
/sys/sunX/conf/RIMFIRE? (y/n) y
```

11. Enter *y* to confirm your selection of the adapter address and interrupt vector. The following lines appear:

```
+ Successful cfc device addition to /sys/sunX/conf/RIMFIRE

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/RIMFIRE for device addition.
OK? (y/n) y
```

12. Enter *y* to add the devices to the /sys/sunX/conf/RIMFIRE file. The following lines appear:

```
+ Backup copy of /sys/sunX/conf/RIMFIRE already exists.
+ Extracting controller information from
/sys/sunX/conf/RIMFIRE...done.
+ No devices attached to controller cfc0.
+ Extracting device information from
/sys/sunX/conf/RIMFIRE...done.
+ No cf devices configured in system.

Configure device cf0 at controller cfc0? (y/n) y
```

13. If you enter *y*, the following type of menu appears:

```
1. Archive QIC Tape Drive
2. Exabyte Tape Drive
3. Fujitsu Asyncronous Disk Drive
4. Generic Disk Drive
5. Generic Tape Drive
6. HP Asyncronous Disk Drive
7. HP Syncronous Disk Drive
8. Kennedy 9612 Tape Drive
9. Maxtor Asyncronous Disk Drive
10. Maxtor Syncronous Disk Drive
11. Micropolis Asyncronous Disk Drive
12. Micropolis Syncronous Disk Drive
13. Miniscribe Asyncronous Disk Drive
14. Patriot Tape Drive
15. Quantum Disk Drive
16. Wangtek QIC Tape Drive
17. Wren Asynchronous Disk Drive
18. Wren Synchronous Disk Drive
Device type =
```

Enter the device type from the menu above. For example, we will configure a system for a Maxtor Synchronous Disk drive. The first device type would be 10.

```
SCSI Target ID of device (0-6) = 0
```

Enter the SCSI target ID of the device being configured.

```
Logical Unit Number of device (0-7 or 0 if none) = 0
```

Enter the SCSI Logical Unit Number of device.

```
Configure device cf1 at controller cfc0? (y/n) n
```

Enter *y* to this question, then go on to select the device type for cf1, or enter n if you won't be configuring this device.

```
Done configuring cf units? (y/n) n
```

Enter *y* if you are done configuring devices, otherwise enter n to repeat the configuring steps for additional devices. When you enter y; the following messages appear:

```
+ Successful addition of cf device(s) to
/sys/sunX/conf/RIMFIRE

[Next step required ONLY for Ciprico RF3523 boards in boot
emulation mode]
[Should NOT be performed when installing any other
controllers]

Removing xy entry from /sys/sunX/conf/RIMFIRE. OK? (y/n) y
```

14. Enter *y* to remove the xy entry from /sys/sunX/conf/RIMFIRE. When you enter *y*, the following lines appear:

```
Remove entries for device xycØ at csr address Øxee4Ø ?
(y/n) y
```

15. Enter *y* to confirm that you want to remove *xycØ* at address *Øxee4Ø* from the sun config file. The following lines appear:

```
+ Successful removal of xyc0 from RIMFIRE

+ Making backup copy of fstab as fstab.nors.

     ******************
     CURRENT fstab FILE
     ******************

/dev/xy0a / 4.2 rw,nosuid 1 1
/dev/xy0g /usr 4.2 rw 1 2
/dev/xy0h /home 4.2 rw 1 3
/dev/xy0f /export/exec 4.2 rw 1 4
/dev/xy0d /export/root 4.2 rw 1 5
/dev/xy0e /export/swap 4.2 rw 1 6

Change mounting for disk xy0 in fstab to be on the Ciprico
disk (rs)? (y/n) y
```

16. Enter *y* to change the xy entries in fstab to rs for the so that the rs disks are mounted during boot. The following lines appear:

```
+ Successful edit of fstab

[Next step required ONLY for Ciprico RF3523 boards in the
boot emulation]"
[mode and Should NOT be performed when installing any other
controllers]"

[NOTE:  If you are planning to use a xt 472 tape
controller, answer no]"
[to the following question and install the JTL disable
jumper.]"

Removing xt entry from RIMFIRE. OK? (y/n) y
```

17. Enter *y* to remove the xt entry from /sys/sunX/conf/RIMFIRE. The following lines appear:

```
Remove entries for device xtc0 at csr address 0xee60 ?
(y/n) y

+ Successful removal of xtc0 from RIMFIRE

+ Checking /sys/sunX/conf/RIMFIRE for root file system
specification.
+ Checking /sys/sunX/conf/RIMFIRE for swap file system
specification.
+ Root file system currently specified as:  xy0
+ Primary swap file system currently specified as:  xy0

[Next step is required ONLY for Ciprico RF3523 boards in
the boot]
[emulation mode with the $dev disks as root or swap disks
AND you]
[are not using root generic and swap generic]

Editing config file to change root and swap specifications.
OK? (y/n) y
```

➠ *NOTE: If you are installing the Rimfire 3523 adapter with Drive 0 as the primary disk, you will need to change the root and swap specifications in the config file.*

18. Enter *y* to proceed with specifying the root and swap devices. The following lines appear:

```
+ Making temporary backup copy of /sys/sunX/conf/RIMFIRE

Root disk is currently: xy0. Change? (y/n) y
```

19. Enter *y* to change the root disk. The following type of selections appear:

```
1. generic   2. rf0   3. rf1   4. rf2
5. rf3       6. rf4   7. rf5   8. rf6
9. rf7       10. cf0

     Select "generic" for determining the root device at
boot time.

Enter the number of the disk that contains the root
filesystem. (1/2/3/4/5/6/7/8/9/10) 10
```

20. Enter the number corresponding to *cfØ* to specify *cfØ* as the root disk. The following message appears:

```
Root disk is now: cf0. OK? (y/n)y

Swap specification is currently: xy0. Change? (y/n)y
```

21. Enter *y* to change the swap disk. The following type of selections appear:

```
1. generic    2. rf0    3. rf1    4. rf2
5. rf3        6. rf4    7. rf5    8. rf6
9. rf7        10. cf0

Select "generic" for determining the primary swap device at
boot time.

You may specify ONLY ONE primary swap device in SunOS 4.X.

Enter the number of the disk to be used as the primary swap
device.(1/2/3/4/5/6/7/8/9/10)
```

22. Enter the number corresponding to *cfØ* to specify *cfØ* as the swap disk. The following message appears:

```
Swap specification is now: cf0. OK? (y/n)y
```

Enter *y* to confirm your selection for the swap device.

```
+ Successful edit of root and swap specification in
/sys/sunX/conf/RIMFIRE.

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/devices. OK? (y/n) y
```

23. Enter *y* to modify the /sys/sunX/conf/devices file and create a backup file called /sys/sunX/devices.nocf. The following text appears:

```
+ Beginning to add cf entry to /sys/sunX/conf/devices.
+ Making backup copy of /sys/sunX/conf/devices as
/sys/sunX/conf/devices.nocf.
+ Adding line to /sys/sunX/conf/devices.
+ Checking for new entry in /sys/sunX/conf/devices.

+ Successful cf device addition to /sys/sunX/conf/devices

[The next step is required if cf disks are to be root or
swap disks]
```

```
[Although not required in other cases, it is still strongly
recommended]

About to modify /sys/sun/swapgeneric.c. OK? (y/n) y
```

24. Enter *y* to modify the /sys/sun/swapgeneric.c file and create a backup file called /sys/sun/swapgeneric.c.nocf. The following information appears:

```
+ Beginning to add cf entries to /sys/sun/swapgeneric.c.
+ Making backup copy of /sys/sun/swapgeneric.c as
/sys/sun/swapgeneric.c.nocf.
+ Adding defines to /sys/sun/swapgeneric.c.
+ Checking for new entries in /sys/sun/swapgeneric.c.

+ Successful cf device description addition to
/sys/sun/swapgeneric.c

[Next step is optional, but strongly recommended]

About to make and run cs35mk. OK? (y/n) y
```

25. Enter *y* to make and run the *cs35mk* file to create the nodes. The following information appears:

```
+ cd ../cs35makedev
+ make DFLAGS=-DSunOS4 install
Installing cs35mk in /etc
+ ./cs35mk y /dev 24 71

[Next step is optional, but strongly recommended]

About to make and install cs35ut. OK? (y/n) y
```

26. Enter *y* to compile *cs35ut.c*. The following text appears:

```
+ cd ../cs35util
+ make DFLAGS=-DSunOS4 install
Installing cs35ut in /etc

[The next step can be done later, but is recommended to do
now]

Run the config program on RIMFIRE? (y/n) y

+ cd /sys/sunX/conf

+ config RIMFIRE

Doing a "make depend"
```

```
[The next step can be done later, but it is recommended to
do it now]

Run the make program to build a new vmunix? (y/n) y
```

27. Enter *y* to run the make program. This will build a new vmunix kernel. The following text appears:

```
+ cd /sys/sunX/RIMFIRE

+ make
```

Lines similar to the following will appear during the building process:

```
cc -sparc -c -O -DsunX -DRIMFIRE -DSUN4_260 -DCRYPT
-DTCPDEBUG -DIPCSHMEM -DIPCSEMAPHORE -DIPCMESSAGE -DSYSACCT
-DLOFS -DNFSSERVER -DNFSCLIENT -DUFS -DQUOTA -DINET
-DKERNEL -I. -I.. -I../.. ../../os/uipc_proto.c

loading vmunix
rearranging symbols
text    data    bss    dec    hex
802816  119496  111000  1033312 fc460

[The next step can be done later]

Save old vmunix and copy /sys/sunX/RIMFIRE/vmunix to
/vmunix? (y/n) y
```

28. Enter *y* to save the old vmunix and copy /sys/sunX/RIMFIRE/vmunix to /vmunix. The following lines appear:

```
+ cp /vmunix /vmunix.nocf

+ cp /sys/sunX/RIMFIRE/vmunix /vmunix

[The next step can be done later]

Reboot the system? (y/n) n
```

29. Enter *y* to exit the installation script and reboot the system. A message appears indicating the system is shutting down.

Adding Adapters to the System

The installation script adds only one adapter, and the devices for it, to the Sun system configuration file. For information about installing additional adapters and drives, refer to Appendix G.

Making Filesystems

You will need to make filesystems on all newly created partitions. Newly created partitions include the following partitions:

- All partitions created during installation of the Ciprico driver (manually or with the installation script).
- All partitions created during installation of additional adapters and drives.

Make file systems only on newly created partitions by entering the following command for each partition you created:

```
newfs -n /dev/rrsab
```

In the foregoing example, *a* represents the chosen drive (Ø, 1, 2, or 3) and *b* represents the chosen partition (a, d, e, f, g, or h).

➽ *NOTE: Do not make a filesystem on the swap partition (b) or on the master partition (c), which includes all other partitions.*

Section 3 - Non-bootable Adapter Installation

This section describes procedures for installing Rimfire 3500 non-bootable hardware and software in a Sun Microsystems workstation. The information may vary between Sun Workstations and versions of SunOS.

This section contains references and file names (for example *sunX* or *4.X*) that depend on the Sun system and version of SunOS you are using. When *X* is used, it should be replaced by the version number for your software or hardware.

In addition, examples in this section include values and file names that may differ from those displayed on your screen, due to variations in system configuration. In this section, **bold** type indicates system-dependent variables.

Hardware Installation

This section describes installation of a Rimfire 3500 VMEbus SCSI Host Bus Adapter (HBA) in Sun Microsystems' workstations. Installation procedures are dependent on the model of Sun workstation.

Required Equipment:

- Sun 3 or Sun 4 Workstation
- Rimfire 3500 VMEbus SCSI Host Bus Adapter

To perform hardware installation, UNIX must be shut down and the Sun system powered off. As an added precaution, disconnect the power cord from the system.

Board Configuration

Figures 3-1 through 3-3 illustrate jumper locations and their factory settings. Unless otherwise indicated, a jumper is set to Ø if the jumper is in and is set to *1* if the jumper is out. Inspect your Rimfire 3500 adapter for proper jumper settings.

➽ *NOTE: The first Rimfire adapter in your system should be jumpered for a Board Address of ØxSØØØ. Rimfire 3500 adapters ordered as "Sun Specials" are already jumpered for this address. If a second Rimfire adapter is installed, the Board Address jumpers on the second adapter must be reconfigured for an address of ØxSSØØ.*

If the Rimfire 3523 adapter is not being used for booting, jumpers JD4L, JD7L, and JTL on the Rimfire 3523 adapter should have disable jumpers installed (see pages 2-3 and 2-4).

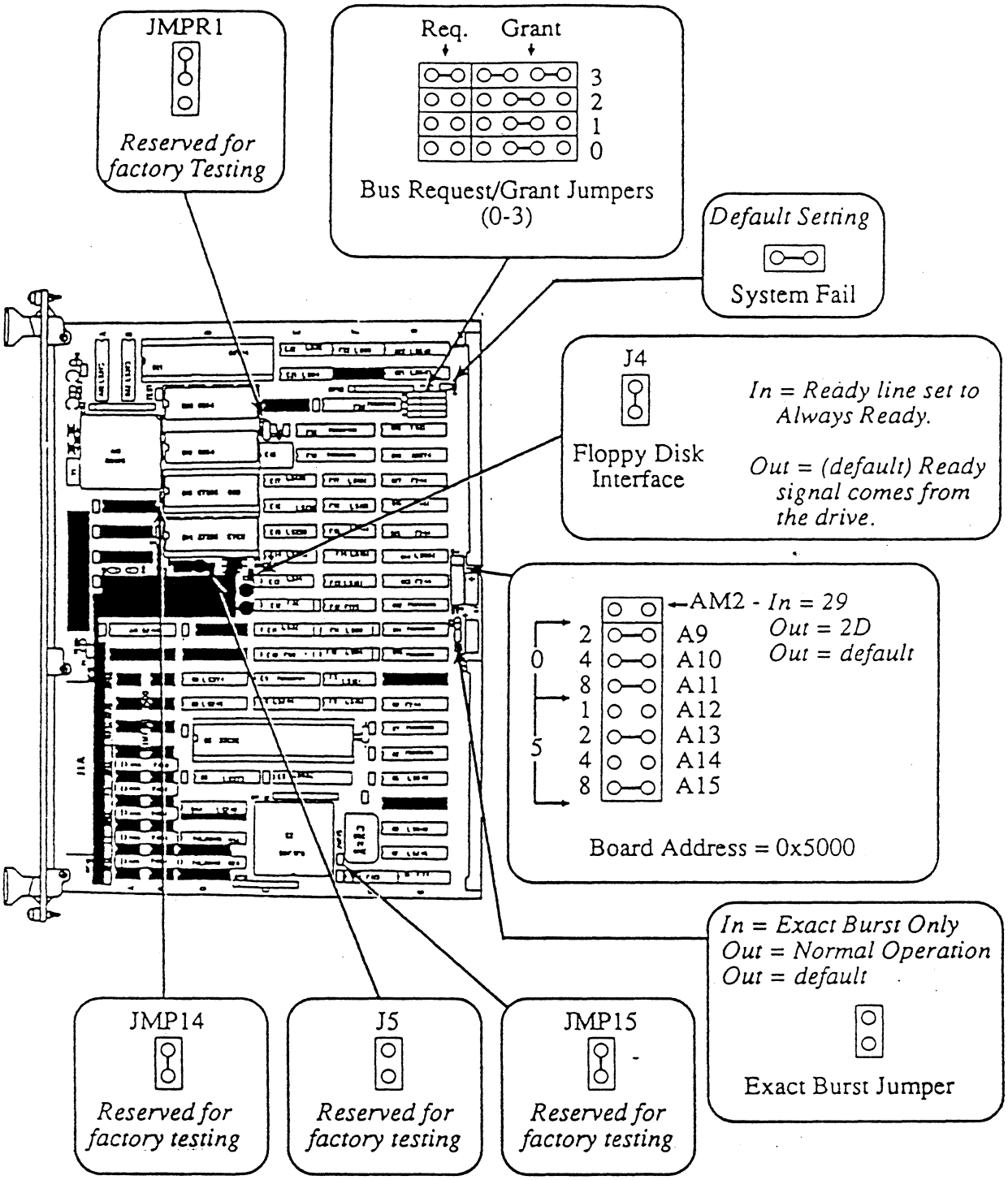


Figure 3-1 Rimfire 3501/3503 Jumpers

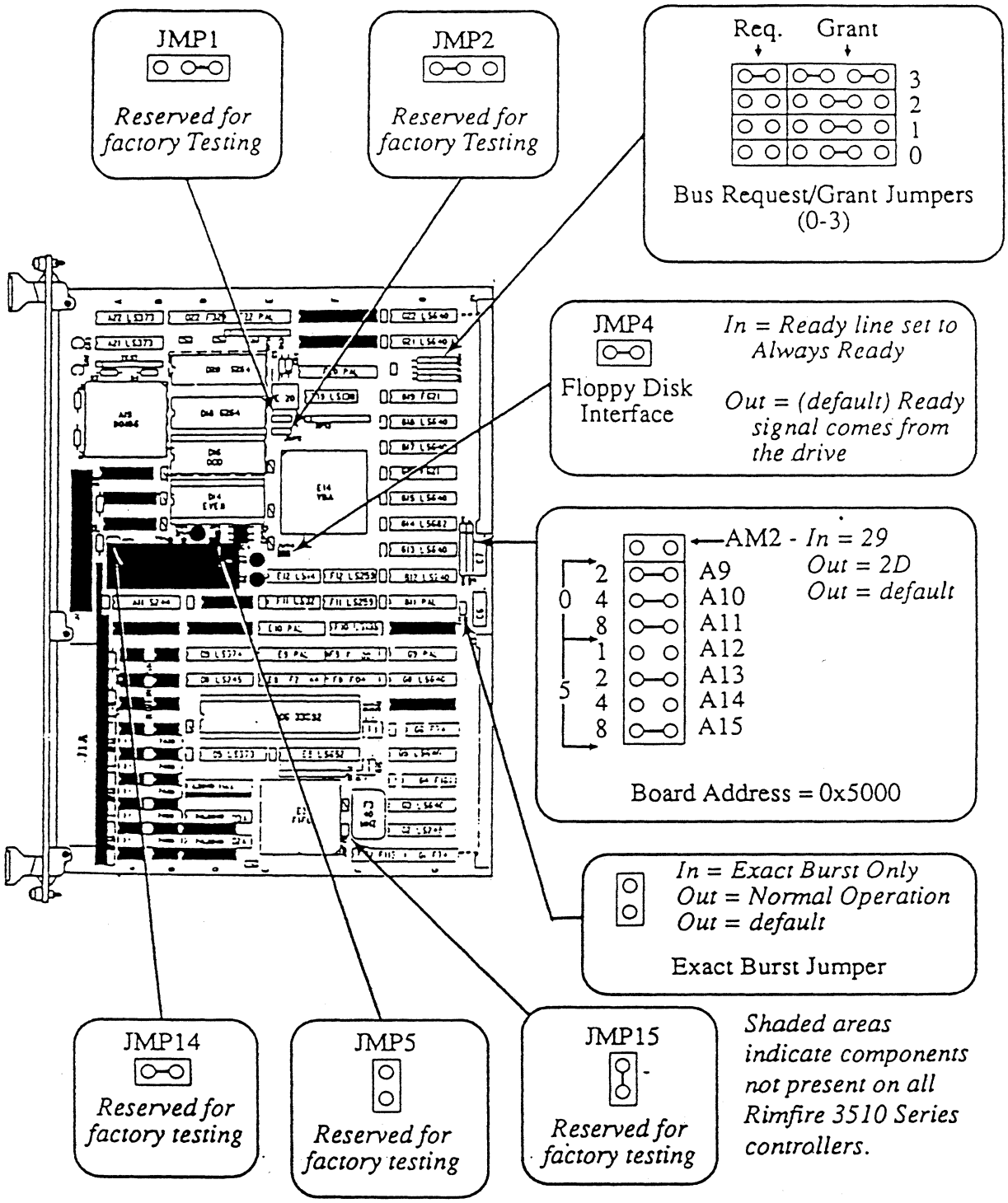


Figure 3-2 Rimfire 3511/3512/3513/3514/3515 Jumpers

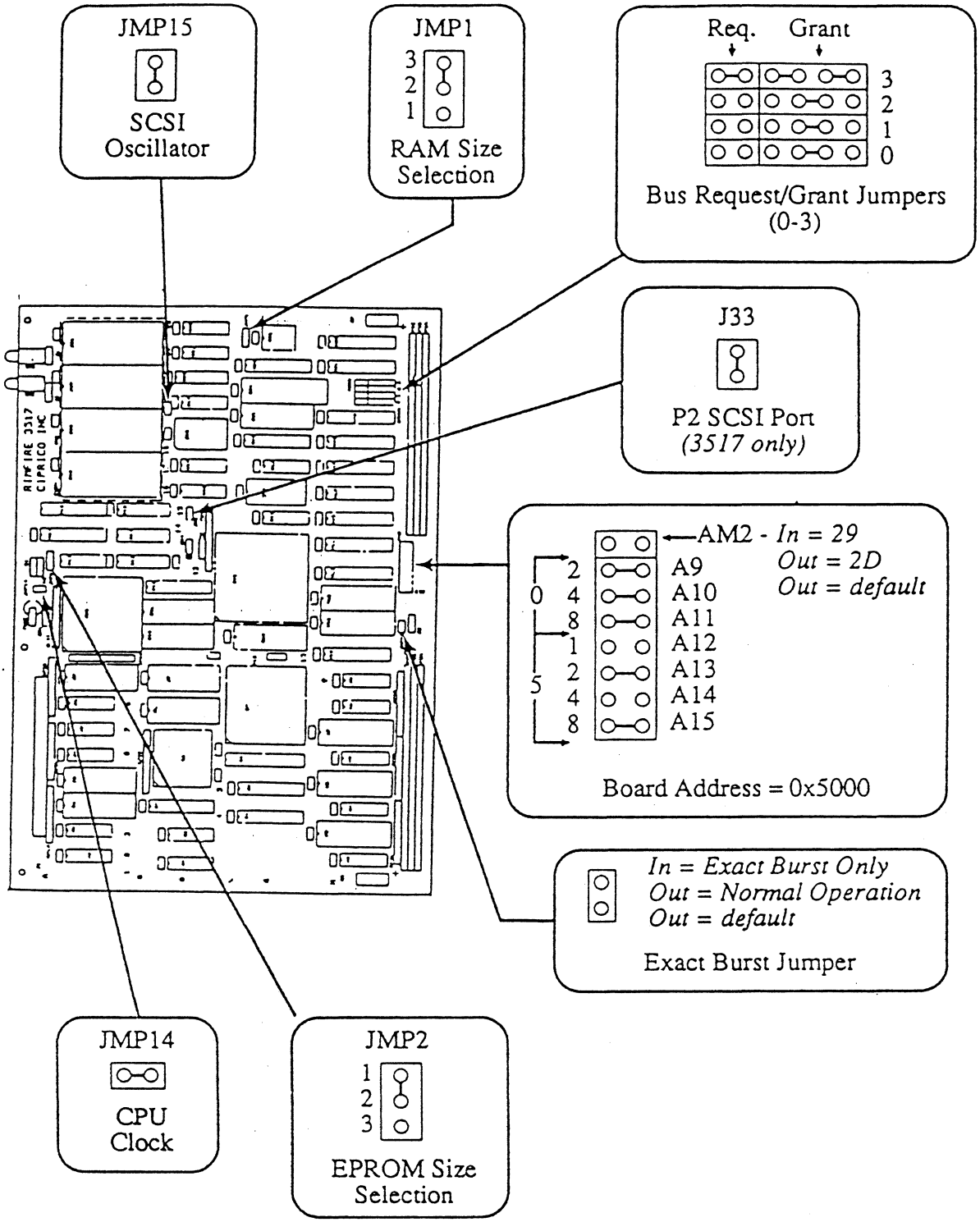


Figure 3-3 Rimfire 3517/3518 Jumpers

Adapter Installation

1. VME card slots on the Sun workstation are covered by metal plates. Select an unused slot and remove the two hex head screws holding the cover plate on the rear of the system. Remove the cover plate and slide the EMI plate out of the slot.

2. Fit the Rimfire 3500 adapter into the adapter frame and connect the cables.

3. Insert the Rimfire 3500 adapter and adapter frame into the slot. Press the adapter and adapter frame firmly into the connectors.

➽ *NOTE: Do not install a Rimfire 3517/3518 in slots 1 through 7 unless you use an adapter frame that isolates the* P2A and C row signals.

4. Fasten the adapter into place with the hex head screws from the rear cover plate.

5. Remove the front cover plate from the workstation to allow access to the backplane. On some Sun Workstations, you will also need to move the power supply to access the backplane. If so, remove the four screws holding the power supply cover and tilt open the power supply. Others may have a small removable panel allowing access to just the jumper area of the backplane.

6. Locate the slot being used. (The number is to the right of the connector.) Remove the *BUS GRANT 3* and *IACK* jumpers (it is a good idea to simply move the jumpers down one pin so they are available, if needed). Figure 3-4 illustrates the *BUS GRANT* and *IACK* jumpers for a given slot.

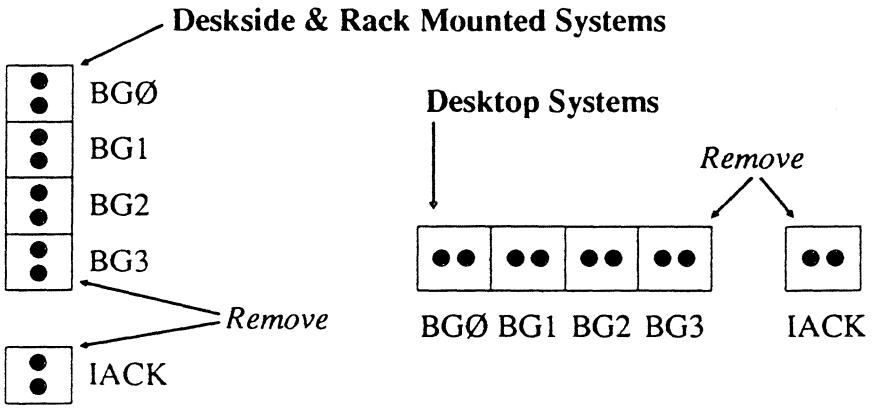


Figure 3-4 BUS GRANT and IACK Jumpers

➽ *NOTE: The BUS GRANT 3 and IACK jumpers must be removed for the Rimfire 3500 adapter to operate properly.*

7. If you moved the power supply to access the backplane, tilt the power supply back to its original position and refasten the power supply cover.

8. Replace the front cover plate (if there is one on you particular system).

9. Connect the drive cables. Figure 3-5 illustrates proper cable connections.

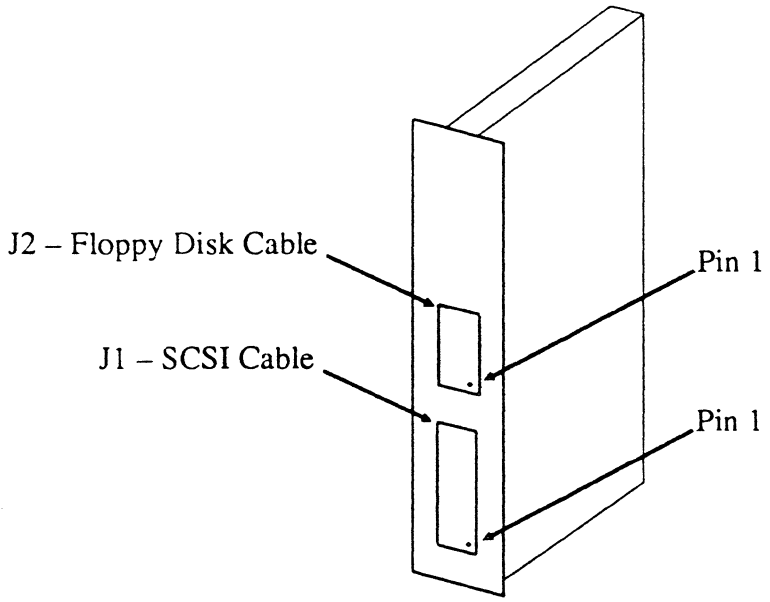


Figure 3-5 Cable Connections

10. Check the drives to ensure that drive parameters (addressing, terminators, sector switches, etc.) are correct. Consult your Drive Manufacturer's manual for recommended drive settings.

11. Power on the new drive(s) and wait until they are ready for operation.

12. Power the Sun system on and check for proper operation. If the board is operating correctly, the Fail and Busy lights will flash on and then turn off.

STOP

This concludes the hardware installation section for the Rimfire 3500 non-bootable SCSI adapter. For information about software installation, continue with this chapter.

Software Installation

Before beginning this, or any other software installation procedure, be sure to have a CURRENT backup of the system files. Make a backup copy of the current vmunix kernel as well.

References:

- Sun Microsystem's documentation and reference manuals for the appropriate Sun Workstation.
- The appropriate drive manufacturer's reference manual.

Required Equipment:

- Sun 3 or Sun 4 Workstation with a tape drive (1/4 inch or 1/2 inch)
- Rimfire 3500 VMEbus SCSI host bus adapter (with adapter frame)
- A copy of the latest version of the Rimfire 3500 SunOS driver on 1/4 inch or 1/2 inch tape.
- Text editing software (Installation procedures in this document assume you are using UNIX's *vi* editor).

BEFORE PROCEEDING

If you are installing your driver manually, refer to Appendix G for installation procedures. If you are using the installation script, continue with this chapter.

Installing the Ciprico Driver

The Ciprico driver can be installed manually or by using the *Installation Script* that is included on the distribution tape. If you select to manually install the Ciprico driver, refer to Appendix G.

Using the Installation Script

The *Installation Script* is included on Ciprico's *Utility and Installation Tape*. It is an interactive utility that steps you through the procedures for installing the Ciprico driver.

➽ *NOTE: In the following procedures, you are asked if it is OK to copy files to existing directories. To avoid overwriting existing files, filenames are automatically assigned a .nocf extension.*

Throughout the following procedures, there are messages indicating whether the particular step is mandatory to driver installation. Steps noted as mandatory must be performed for successful driver installation.

Due to system variations, the information displayed on your screen (file names, software and hardware references, device counts, etc.) may differ from examples in this manual.

If the install script is interrupted because of an error, you are given a series of choices of what to do next. If you want to re-enter the program, after making the appropriate changes, choose that option, make the changes, then enter **fg** *to re-enter the program.*

1. Create a directory (called *CIPRICO*) for the files on Ciprico's *Utility and Installation* tape. Enter the following line:

```
mkdir /sys/CIPRICO
```

2. Enter the following command to change to the directory you just created:

```
cd /sys/CIPRICO
```

3. Insert the Ciprico *Utility and Installation* Tape. If you are using 1/4 inch tape, read the tape by entering the following command:

```
tar xvbf 126 /dev/rst8
```

If your system uses a high-density tape drive, you can read the tape as /dev/rst0.

4. Enter the following command to switch to the *cf/install* directory:

```
cd cf/install
```

5. Start the installation script by entering the following command:

```
doinstall cf
```

The following information appears:

```
(C)  C O P Y R I G H T   1 9 8 9


Ciprico, Inc.
2955 Xenium Lane
Plymouth, MN  55441
(612) 559-2034
All Rights Reserved.

+ Determining CPU architecture... CPU is a sunX
+ Determining hostname... Hostname is Rimfire
+ Checking for mount of /usr,.. /usr is mounted.
+ Trying to find system files
+ You are running SunOS release 4.X
+ Your system configuration directory should be RIMFIRE
+ System configuration file is /sys/sunX/conf/RIMFIRE
```

➳ *NOTE: The Hostname (indicated by RIMFIRE) will be the system name assigned during SunOS installation.*

```
[The next step is mandatory for driver installation]

About to copy cf device driver files to /sys/sundev and
/sys/sunX/OBJ. OK? (y/n) y
```

6. Enter *y* to copy the *cf* device driver files to /sys/sundev and /usr/unclude/sundev. The following messages appear:

```
+ cp ../sundev/cs35.c /sys/sundev/cs35.c
+ cp ../sundev/cs35err.h /sys/sundev/cs35err.h
+ cp ../sundev/cs35err.h /usr/include/sundev/cs35err.h
+ cp ../sundev/cs35flp.h /sys/sundev/cs35flp.h
+ cp ../sundev/cs35flp.h /usr/include/sundev/cs35flp.h
+ cp ../sundev/cs35if.h /sys/sundev/cs35if.h
+ cp ../sundev/cs35if.h /usr/include/sundev/cs35if.h
+ cp ../sundev/cs35int.h /sys/sundev/cs35int.h
+ cp ../sundev/cs35int.h /usr/include/sundev/cs35int.h
+ cp ../sundev/cs35io.h /sys/sundev/cs35io.h
+ cp ../sundev/cs35io.h /usr/include/sundev/cs35io.h
+ cp ../sundev/cs35lib.c /sys/sundev/cs35lib.c
+ cp ../sundev/cs35prm.h /sys/sundev/cs35prm.h
+ cp ../sundev/cs35prm.h /usr/include/sundev/cs35prm.h
+ Copy of cf driver files done.

[The next step is mandatory for driver installation]

About to modify /sys/sun/conf.c. OK? (y/n) y
```

7. Enter *y* to modify the /sys/sun/conf.c file. The following lines appear:

```
+ Beginning to add cf entries to /sys/sun/conf.c.
+ Making backup copy of /sys/sun/conf.c as
/sys/sun/conf.c.nocf.
+ Adding defines to /sys/sun/conf.c.
+ Counting block device entries...24 found.
+ Counting character device entries...71 found.
+ Adding bdevsw entry to /sys/sun/conf.c.
+ Adding cdevsw entry to /sys/sun/conf.c.
+ Checking for new entries in /sys/sun/conf.c.

+ Successful cf device description addition to
/sys/sun/conf.c

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/files. OK? (y/n) y
```

8. Enter *y* to modify the /sys/sunX/conf/files file. The modifications will be made and a backup copy (files.nocf) will be written. The following lines appear:

```
+ Beginning to add cf entries to /sys/sunX/conf/files.
+ Making backup copy of /sys/sunX/conf/files as
/sys/sunX/conf/files.nocf.
+ Adding configuration lines to /sys/sunX/conf/files.
+ Checking for new entries in /sys/sunX/conf/files.

+ Successful cf device addition to /sys/sunX/conf/files

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/RIMFIRE. OK? (y/n) y
```

9. Enter y to modify /sys/sunX/conf/RIMFIRE. A backup copy (/sys/sunX/conf/RIMFIRE.nocf) will be made. The following lines appear:

```
+ Making backup copy of /sys/sunX/conf/RIMFIRE as
/sys/sunX/conf/RIMFIRE.nocf.
+ Extracting controller information from
/sys/sunX/conf/RIMFIRE...done.
+ No cfc controllers currently installed.
+ Adding controller cfc0 to /sys/sunX/conf/RIMFIRE

[Note: Controller address MUST NOT be the same or overlap]
[any existing devices specified in file
/sys/sunX/conf/RIMFIRE.]
[Default value is displayed in braces below]
[Please be sure to preface hexidecimal number with "0x" ]
```

10. Enter the address of the Rimfire 35XX adapter and hit the return key, or just enter the return key if the address of the adapter is 0x5000. Then enter the interrupt vector of the controller, or just hit the return key if the interrupt vector 0xFE is ok.

```
VME address of controller? [0x5000]
VME interrupt vector of controller? [0xFE]

Add controller cfc0 at address 0x5000 vector 0xFE to
/sys/sunX/conf/RIMFIRE? (y/n) y
```

11. Enter y to confirm your selection of the adapter address and interrupt vector. The following lines appear:

```
+ .Successful cfc device addition to /sys/sunX/conf/RIMFIRE

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/RIMFIRE for device addition.
OK? (y/n) y
```

12. Enter *y* to add the devices to the /sys/sunX/conf/RIMFIRE file. The following lines appear:

```
+ Backup copy of /sys/sunX/conf/RIMFIRE already exists.
+ Extracting controller information from
/sys/sunX/conf/RIMFIRE...done.
+ No devices attached to controller cfc0.
+ Extracting device information from
/sys/sunX/conf/RIMFIRE...done.
+ No cf devices configured in system.

Configure device cf0 at controller cfc0? (y/n) y
```

13. If you enter *y*, the following type of menu appears:

```
1. Archive QIC Tape Drive
2. Exabyte Tape Drive
3. Fujitsu Asyncronous Disk Drive
4. Generic Disk Drive
5. Generic Tape Drive
6. HP Asyncronous Disk Drive
7. HP Syncronous Disk Drive
8. Kennedy 9612 Tape Drive
9. Maxtor Asyncronous Disk Drive
10. Maxtor Syncronous Disk Drive
11. Micropolis Asyncronous Disk Drive
12. Micropolis Syncronous Disk Drive
13. Miniscribe Asyncronous Disk Drive
14. Patriot Tape Drive
15. Quantum Disk Drive
16. Wangtek QIC Tape Drive
17. Wren Asynchronous Disk Drive
18. Wren Synchronous Disk Drive
Device type =
```

Enter the device type from the menu above. For example, we will configure a system for a Maxtor Synchronous Disk drive. The first device type would be 10.

```
SCSI Target ID of device (0-6) =
```

Enter the SCSI target ID of the device being configured.

```
Logical Unit Number of device (0-7 or 0 if none) = 0
```

Enter the SCSI Logical Unit Number of device.

```
Configure device cf1 at controller cfc0? (y/n)
```

Enter *y* if you will configure this device, then go on to select the device type for cf1, or enter *n* if you won't be configuring this device.

```
Done configuring cf units? (y/n)
```

Enter *y* if you are done configuring devices. To repeat the configuring steps for additional devices, enter *n*. When you enter *y*, the following messages appear:

```
+ Successful addition of cf device(s) to
/sys/sunX/conf/RIMFIRE
```

➽ *NOTE: Devices can be added. Refer to Appendix I for details.*

```
[Next step required ONLY for Ciprico RF3523 boards in boot
emulation mode]
[Should NOT be performed when installing any other
controllers]

Removing xy entry from RIMFIRE. OK? (y/n) n
```

14. Enter *n* to keep the xy entry in /sys/sunX/conf/RIMFIRE. Removing the xy entry should be performed only for installing bootable disks. The following lines appear:

```
+ Skipping removal of xy entry from /sys/sunX/conf/RIMFIRE.

+ Making backup copy of fstab as fstab.nors.

     ******************
     CURRENT fstab FILE
     ******************

/dev/xy0a / 4.2 rw,nosuid 1 1
/dev/xy0g /usr 4.2 rw 1 2
/dev/xy0h /home 4.2 rw 1 3
/dev/xy0f /export/exec 4.2 rw 1 4
/dev/xy0d /export/root 4.2 rw 1 5
/dev/xy0e /export/swap 4.2 rw 1 6

Change mounting for disk xy0 in fstab to be on the Ciprico
disk (rs)? (y/n) n
```

15. Enter *n* to keep the xy entries in /etc/fstab. Replacing the xy entries with rs entries should only be performed for installing bootable disks. The following lines appear:

```
[Next step required ONLY for Ciprico RF3523 boards in the
boot emulation]"
[mode and Should NOT be performed when installing any other
controllers]"

[NOTE:  If you are planning to use a xt 472 tape
controller, answer no]"
[to the following question and install the JTL disable
jumper.]"

Removing xt entry from RIMFIRE. OK? (y/n) n
```

16. Enter n to keep the xt entry in /sys/sunX/conf/RIMFIRE. Removing the xt entry should only be performed for installing bootable disks. The following lines appear:

```
+ Skipping removal of xt entry from /sys/sunX/conf/RIMFIRE.

+ Checking /sys/sunX/conf/RIMFIRE for root file system
specification.
+ Checking /sys/sunX/conf/RIMFIRE for swap file system
specification.
+ Root file system currently specified as:  xy0
+ Primary swap file system currently specified as:  xy0

[Next step is required ONLY for Ciprico RF3523 boards in
the boot]
[emulation mode with the $dev disks as root or swap disks
AND you]
[are not using root generic and swap generic]

Editing config file to change root and swap specifications.
OK? (y/n) n
```

17. Enter *n* to leave the root and swap specifications as they are. The following lines appear:

```
+ Skipping specification of root and swap disks in
/sys/sunX/conf/RIMFIRE.

[The next step is mandatory for driver installation]

About to modify /sys/sunX/conf/devices. OK? (y/n) y
```

18. Enter *y* to modify the /sys/sunX/conf/devices file and create a backup file called /sys/sunX/devices.nocf. The following text appears:

```
+ Beginning to add cf entry to /sys/sunX/conf/devices.
+ Making backup copy of /sys/sunX/conf/devices as
/sys/sunX/conf/devices.nocf.
+ Adding line to /sys/sunX/conf/devices.
+ Checking for new entry in /sys/sunX/conf/devices.

+ Successful cf device addition to /sys/sunX/conf/devices

[The next step is required if cf disks are to be root or
swap disks]
[Although not required in other cases, it is still strongly
recommended]

About to modify /sys/sun/swapgeneric.c. OK? (y/n) y
```

19. Enter *y* to modify the /sys/sun/swapgeneric.c file and create a backup file called /sys/sun/swapgeneric.c.nocf. The following information appears:

```
+ Beginning to add cf entries to /sys/sun/swapgeneric.c.
+ Making backup copy of /sys/sun/swapgeneric.c as
/sys/sun/swapgeneric.c.nocf.
+ Adding defines to /sys/sun/swapgeneric.c.
+ Checking for new entries in /sys/sun/swapgeneric.c.

+ Successful cf device description addition to
/sys/sun/swapgeneric.c

[Next step is optional, but strongly recommended]

About to make and run cs35mk. OK? (y/n) y
```

20. Enter *y* to make and run the *cs35mk* file to create the nodes. The following information appears:

```
+ cd ../cs35makedev
+ make DFLAGS=-DSunOS4 install
Installing cs35mk in /etc
+ ./cs35mk y /dev 24 71

[Next step is optional, but strongly recommended]

About to make and install cs35ut. OK? (y/n) y
```

21. Enter *y* to compile *cs35ut.c*. The following text appears:

```
+ cd ../cs35util
+ make DFLAGS=-DSunOS4 install
Installing cs35ut in /etc

[The next step can be done later, but is recommended to do
now]

Run the config program on RIMFIRE? (y/n) y
```

22. Enter *y* to configure the RIMFIRE file. The following lines appear:

```
+ cd /sys/sunX/conf
+ config RIMFIRE
Doing a "make depend"

[The next step can be done later, but it is recommended to
do it now]

Run the make program to build a new vmunix? (y/n) y
```

23. Enter *y* to run the make program. This will build a new vmunix kernel. The following text appears:

```
+ cd /sys/sunX/RIMFIRE
+ make

     Lines similar to the following will appear during the
building process:

cc -sparc -c -O -DsunX -DRIMFIRE -DSUN4_260 -DCRYPT
-DTCPDEBUG -DIPCSHMEM -DIPCSEMAPHORE -DIPCMESSAGE -DSYSACCT
-DLOFS -DNFSSERVER -DNFSCLIENT -DUFS -DQUOTA -DINET
-DKERNEL -I. -I.. -I../.. ../../os/uipc_proto.c

loading vmunix
rearranging symbols
text      data      bss      dec      hex
802816   119496   111000   1033312   fc460

[The next step can be done later]

Save old vmunix and copy /sys/sunX/RIMFIRE/vmunix to
/vmunix? (y/n) y
```

24. Enter *y* to save the old vmunix and copy /sys/sunX/RIMFIRE/vmunix to /vmunix. The following lines appear:

```
+ cp /vmunix /vmunix.nocf
+ cp /sys/sunX/RIMFIRE/vmunix /vmunix

[The next step can be done later]

Reboot the system? (y/n) n
```

25. Enter *y* to exit the Installation Script and reboot the system. A message appears indicating the system is shutting down.

Adding Adapters to the System

The installation script adds only one adapter, and the devices for it, to the Sun system configuration file. For information about installing additional adapters and drives, refer to Appendix G.

Making Filesystems

You will need to make filesystems on all newly created partitions. Newly created partitions include the following partitions:

- All partitions created during installation of the Ciprico driver (manually or with the installation script).
- All partitions created during installation of additional adapters and drives.

Make file systems only on newly created partitions by entering the following command for each partition you created:

```
newfs -n /dev/rrsab
```

In the foregoing example, *a* represents the chosen drive (Ø, 1, 2, or 3) and *b* represents the chosen partition (a, d, e, f, g, or h).

➳ *NOTE: Do not make a filesystem on the swap partition (b) or on the master partition (c), which includes all other partitions.*

Appendix A - Specifications

Physical

Rimfire 35*XX*	Single slot, double height VME Eurocard form factor board (233mm x 160mm)
Rimfire 3523	Single-slot, triple-height, full-depth board (400mm x 366.66mm)

Electrical

Voltage:	4.75 Vdc to 5.25 Vdc
Current:	4.0 A typical (at +5 Vdc)

Capacity

Up to eight SCSI devices

Up to four floppy disk drives (with optional floppy disk interface)

Transfer Rate

SCSI data rates to 2.0 Mbytes/second (asynchronous mode) or 5 Mbytes/second (synchronous mode)

Sustained VMEbus transfer rates at SCSI bus speeds for SCSI operations. Sustained VMEbus transfer rates at floppy data rates for floppy operations. Burst VMEbus transfer rates of up to 20 Mbytes/second with minimum memory response time, or 30 Mbytes/second using block mode transfer.

Optional floppy disk port transfer rates of 250 or 500 Kbits/second

Environmental

Operating:	Temperature:	0 °C – +55 °C for SCSI only board
		+15 °C – +45 °C for floppy disk option
	Air Flow:	200 linear feet per minute
	Humidity:	10% to 80% non-condensing
	Elevation:	0 feet to 10,000 feet
Non-Operating:	Temperature:	-40 °C – +85 °C
	Humidity:	10% to 95% non-condensing
	Elevation:	40,000 feet maximum

Bus Interface

VMEbus standard (Revision C.1)

Device Interface

ANSI X3.131 - 1986 for SCSI Interface

ANSI X3.80 - 1981 for Floppy Disk Interface

Appendix B - *cs35ut* Program

The *cs35ut* program is a utility to work with the Rimfire 3500 host bus adapter driver. It is intended for use with the SunOS 3.X and 4.0 operating system on Sun Workstations. It supports commands for formatting, track and sector mapping, and various tape commands. It allows access to data structures the device driver uses for configuring and partitioning.

cs35ut Commands

Table B-1 lists the *cs35ut* disk device commands:

Table B-1 cs35ut Disk Device Commands

Code	Command	Code	Command
a	Start Device	o	Open a New Device
b	Debug Control	q	Quit
c	Identify Controller	r	Read and Display Label
d	Read Capacity	u	Stop Device
f	Format drive	v	Verify Disk Format
l	Choose Label	w	Write Label to Disk
m	Map Sectors		

Table B-2 lists the *cs35ut* tape device commands:

Table B-2 cs35ut Tape Device Commands

Code	Command	Code	Command
a	Load Device	q	Quit
b	Debug Control	r	Rewind
c	Identify controller	s	Search Filemark
d	Read Capacity	t	Retension Tape
e	Erase Tape	u	Unload Device
n	Mode Select Command	w	Write Filemark
o	Open a New Device		

Table B-3 lists the *cs35ut* dummy device commands:

Table B-3 cs35ut Dummy Device Commands

Code	Command
b	Debug control
c	Identify Controller
o	Open a Device
q	Quit

The calling parameters for the cs35ut utility are:

```
cs35ut
cs35ut /dev/rrs0a
```

cs35ut is the program name, and /dev/rrs0a is the device to be opened. The disk device name must contain a #X as the last two letter/number values of its name; # is the sun configuration number, and X is a letter representation of the partition to open (a=0, b=1, c=2, d=3, e=4, f=5, g=6, h=7). If the command line does not contain the device to open, the user is prompted for the device.

If the user enters a disk device, the menu for disk devices appears.

```
Open Device:  /dev/rrs0a                    Size:  16320
cs35ut>

Available Disk Commands:
a     Start Device            o     Open a new device
b     Debug control           q     Quit
c     Identify Controller     r     Read and Display Label
d     Read Capacity           u     Stop Device
f     Format Disk             v     Verify Disk Format
l     Choose Label            w     Write Label to Disk
m     Map Sectors
```

If the user enters a tape device, the menu for tape devices appears:

```
Open Device:  /dev/rrt1
cs35ut

Available Tape Commands:
 a     Load Device              q     Quit
 b     Debug control            r     Rewind
 c     Identify Controller      s     Search Filemark
 d     Read Capacity            t     Retension Tape
 e     Erase Tape               u     Unload Device
 n     Mode select command      w     Write Filemark
 o     Open a new Device
```

The user also can set up a dummy device (see Appendix G) in the system configuration so that the dummy device can be opened, allowing the user to issue a small number of commands that only pertain to the adapter, not the SCSI devices. This type of menu appears:

```
Open Device:  /dev/rrd3
cs35ut

Available commands:
 b     Debug control
 c     Identify Controller
 o     Open a device
 q     Quit
```

Commands for All Device Types

The commands described in this section are available for all device types.

Start/Load Device (a)

The start device command will issue the SCSI optional command that starts the device. What a "start device" command will actually do is device specific. For disk drives that support this command, this will usually cause the disk to spin up and return to the online state. For tape drives that support this command, this will usually cause the tape to be loaded and return to the online state.

Debug Control (b)

The debug control command gives the user the option of selecting from 0 to 0xffff for the debug value. With a value of 0, no debug messages will be displayed on the screen. As the debug value approaches 0xffff, more debug messages will appear on the screen.

Identify Controller (c)

The Identify Controller command displays the firmware revision, the engineering revision, the firmware date, and determines whether a floppy controller exists on this board, as this information shows:

```
Open Device:  /dev/rrs0a                    Size:  16320
Last Command:  Identify Controller
cs35ut>


              FW Rev:  07
              Eng Rev:  57
              FW Date:  Apr 14, 1989
              floppy controller present
```

Read Capacity (d)

The Read Capacity command asks the user for the logical block address for the SCSI Read Capacity command, displays the last logical block, and the block length as the following information shows:

```
Open Device:  /dev/rrs0a                          Size: 16320
Last Command:  Read Capacity
cs35ut>


                            Dec      Hex
 Last Logical block      285039    4396f
 Block Length               512      200
```

➽ *NOTE: This is the last logical block address, not the number of blocks on the unit.*

Open a New Device (o)

The Open New Device command prompts the user for a new device. Only one device may be open at a time with the utility. Therefore, opening a new device closes a previously selected device.

Quit

The Quit command exits the cs35ut program and returns to the system prompt.

Stop/Unload Device (u)

The Stop Device command issues the SCSI optional Stop Device command to the device. What the command does is device specific. With most disk devices, issuing this command will cause the disk to spin down. For most tape devices, the tape will be unloaded.

Commands for Disk Devices

The commands described in this section are available for disk devices.

Format (f)

The Format command prompts the user for information about reading the defect list. The user can format without the defect list, use only the manufactures defect list, or use the grown defect list while formatting (if the device supports this feature). This type of information appears on the screen:

```
Open Device: /dev/rrs0a                          Size:  16320
Command: format
cs35ut>
```

Then, the user is prompted for an interleave value. If the value entered by the user is greater than the number of sectors/track, the user must enter another interleave value. The format tells the user that the adapter is formatting blocks 0 to the last block, and gives him the option to abort the format command. Then, the user is informed that the format command is in process, and of the estimated time required to complete the format for that device. This type of information appears on the screen:

```
Open Device: /dev/rrs0a                          Size:  16320
Command: format
cs35ut>

Do you want to include the defect list in the format? y
Enter choice of defect list from menu below:

1. Manufacturers defect list read from disk
2. Grown defect list read from disk
3. Both the Manufacturers and the Grown list

What interleave? 1

Formatting blocks 0 to _____ in approximately 15 minutes.
Are you sure?

Formatting:
```

When the format command is complete, the word "Done" appears after the word "Formatting:", the second line changes from "Command:" to "Last Command:" as this information shows:

```
Open Device: /dev/rrs0a                          Size:  16320
Last Command: format
cs35ut

Interleave value?  Enter a 1 for this value unless you
are certain that you want a different value:

Formatting blocks 0 to _____.
Are you sure?


Formatting:  Done.
```

Choose Label (l)

The Choose Label command allows the user to get the disk geometry information by issuing a SCSI mode sense command (get the geometry from the disk), or edit the current label. If the user chooses to get the geometry from the disk, and the mode sense command fails, the user should enter the L command to edit the current lable, changing the parameters as set in the device manual. The disk capacity is displayed as the user is prompted for the partition values. Consult the Sun manual for recommended partition sizes. This kind of information appears:

```
Open Device:  /dev/rrs0a                         Size:  16320
Command:  Choose Label
cs35ut>

Please enter your choice:  1

 0 - edit current label
 1 - Get disk geometry information from disk thru mode sense
```

If the user selects number 1 (get geometry from disk), a SCSI mode sense command is issued to the device to get the geometry. The partitions appear, giving the user a chance to modify them. The user is asked to enter the free spcae hog partition. It is the partition that holds the remainder of the disk as the partition are selected. The user can enter *n* if he does not want the free space hog partition.

After he has changed the partitions and pressed the enter key, the following type of information appears:

```
<Micropolis 1355 cyl 1022 alt 0 hd 8 sec 34 apc 0>
Checksum in label is ok.
Magic number in label is ok.
# of alt/cylinder    0          # of alt cylinders    0
size of gap1         0          # of heads            8
size of gap2         0          # of sectors/track    34
interleave factor    1          label location       0
# of data cylinders  1022       physical partition # 0

              starting     ending          size
  Partition   cylinder    cylinder   blocks /  Mbytes    Overlaps
     a           0          139     29400 /   15.05       c
     b         140          419     58800 /   30.11       c
     c           0          780    164010 /   83.97       a b h
     d           0            0         0 /   0.00
     e           0            0         0 /   0.00
     f           0            0         0 /   0.00
     g*          0            0         0 /   0.00
     h         420          780     75810 /   38.81       c
```

Map Sectors (m)

The Map Sectors command issues the SCSI Reassign Blocks command to the device. The user is prompted for a starting block to be mapped, and the number of blocks. This type of information appears:

```
Open Device:  /dev/rrs0a                        Size:  16320
Command:  Map Sectors
cs35ut>


'q' to quit
Starting Block:
Number of Blocks:

Enter 'y' to map blocks _____ through _____          Yes
```

Read and Display Label (r)

The Read Label command reads the label from the disk and displays the label. This type of information appears:

```
<Micropolis 1355 cyl 1022 alt 0 hd 8 sec 34 apc 0>
Checksum in label is ok.
Magic number in label is ok.
# of alt/cylinder    0          # of alt cylinders    0
size of gap1         0          # of heads            8
size of gap2         0          # of sectors/track    34
interleave factor    1          label location       0
# of data cylinders  1022       physical partition # 0

              starting     ending          size
  Partition   cylinder     cylinder   blocks /  Mbytes    Overlaps
     a           0          139     29400 /   15.05       c
     b         140          419     58800 /   30.11       c
     c           0          780    164010 /   83.97       a b h
     d           0            0         0 /   0.00
     e           0            0         0 /   0.00
     f           0            0         0 /   0.00
     g*          0            0         0 /   0.00
     h         420          780     75810 /   38.81       c

      Disk Capacity:  286.48          Total Cylinders:  1387
```

Verify Disk Format (v)

The Verify Disk Format command verifies the disk between the starting block and ending block entered by the user. The command allows the user to verify a single block many times, or the whole disk. It also allows the user to map bad sectors as they are discovered.

➽ *NOTE: If your disk supports automatic block relocation, you need not run Verify to map bad sectors.*

This verify is read-only. It does not destroy data. The following type of information appears:

```
Open Device:  /dev/rrs0a                          Size: 16320
Command:  Verify disk
cs35ut
```

```
How many passes do you want to make?  2
OK, doing 2 verify passes.
Do you want to map bad blocks as they are discovered?  Yes
Ready to verify disk, enter 'y' to proceed:  Yes
```

Then, this type of information appears:

```
Open Device:  /dev/rrs0a                          Size: 16320
Command:  Verify disk
cs35ut>


Enter 'y' to verify the whole disk:  No

Starting Block:  0
Ending Block:  5000

Verifying blocks 0 to 5000
Are you sure?   Yes

Verifying:

Block:  0
```

As the verify command proceeds, the block currently being verified appears on the bottom line. When done verifying, the word "Done" appears after the word "Verifying", like this:

```
Open Device:  /dev/rrs0a                          Size: 16320
Command:  Verify disk
cs35ut>

Enter 'y' to verify the whole disk:  No

Starting Block:  0
Ending Block:  5000

Verifying blocks 0 to 5000
Are you sure?   Yes

Verifying:  Done

Block:  5000
```

Write Label to Disk (w)

The write label command writes the current chosen label to the disk.

Commands for Tape devices

The commands described in this section are avaliable for tape devices.

Erase Tape (e)

The Erase Tape command issues the SCSI Erase Tape command. What actually happens during an erase tape is device specific. Some devices will erase the remaining tape from the position that the tape is currently positioned, but some will start erasing from the beginning of tape. Since the erase can be time consuming once it is started, a warning message will be displayed giving the user a way to abort the command.

Mode Sense/Select (n)

The Mode Sense/Select command for tape selects the density used when writing to, or reading from, the tape. The density code can be determined by examining the tape drive manual. This type of infromation appears:

```
Open Device:  /dev/rrt1
Command:  Mode Sense/Select
cs35ut>

                      Number to change,  when done:

 1. Speed code (hex):                    0
 2. Buffered flag ( 1 = buffered):       1
 3. Density code (hex):                  0
 4. Block length (hex):                  200

Number of blocks = 000000
```

Rewind Device (r)

The rewind tape command causes the tape to be rewound. There are no special features for this command.

Search Filemark (s)

The Search Filemark command will prompt the user for the number of filemarks to search. Then the tape will be repositioned after the user entered count of filemarks on the tape.

Retension Tape (t)

The retension tape command will retension and load the currently opened tape device.

Write Filemark (w)

The write filemark command causes the tape drive to write a filemark on the tape.

Appendix C - Unit Options Defines

Unit options defines specify operating parameters for the particular drive and are defined by "ORing" together the desired flag defines. Flag defines are stored in the */sys/sundev/cs35prm.h* and *cs35flp.h* files, and created device defines are stored in *cs35prm.h* files. Table C- 1 lists the unit-option-flag defines for disk and tape drives, Table C-2 lists the request sense counts, and Table C- 3 lists the unit-option-flag defines for floppy drives.

Table C-1 Unit Options Flag Defines (Disk & Tape)

Flag Define	Define Value	Description
NOOPN_RDCAP	0x000001	Do not issue Read Capacity at open().
NOOPN_MDSEL	0x000002	Do not issue Mode Select at open().
ONEFILEMARK	0x000004	Tape drive can only write one filemark at a time.
GEN_MODE	0x000008	Tape drive has modes of operation. Add this if you can only do certain commands (i.e., *Mode Select*) when the drive is in general mode, in contrast to read/write mode.
NORESERVE	0x000010	Don't do *Reserve* and *Release* commands.
SYNCHRONOUS	0x000080	This drive is synchronous.
LONGRDYWAIT	0x000100	This drive may take a while to pass *Test Unit Ready* commands.
EXABYTE	0x000200	This is a flag to handle vendor unique parameters for the Exabyte tape drive.
FIXEDBLK	0x000400	Force Fixed Block mode.
REQLENLO	0x000800	Set the combination of REQLENLO and REQLENHI for the desired extended status size. (see Table C-2 for REQUEST SENSE count.)
REQLENHI	0x001000	
NORETRYSOFT	0x002000	This specifies no retries on soft errors. An error will be printed to the screen.
SORTCMB	0x004000	Enable Sort and Combine for this device. **Tape devices should not use this feature.** Your adapter must have firmware revision 9, or higher, for the 3500, and non-beta firmware for the 3510, to use this feature.
NOREWIND	0x008000	This is a no rewind device for tapes.
ONEFM	0x10000	This flag terminates tape with one EOF (like Sun), instead of two. Practice caution when using this flag. If it is desired to append files to the end of tape, it may be hard to determine were EOT is, as a filemark is also written between files.

Table C-2 Request Sense Count

REQLENHI	REQLENLO	REQUEST SENSE COUNT
0	0	8
0	1	16
1	0	24
1	1	32

Table C-3 Unit Options Flag Defines (Floppy)

Flag Define	Define Value	Description
FLM_200SSSD	(0x05)	8" (200 mm) SS/SD, 48 tpi
FLM_200DSSD	(0x6)	8" (200 mm) DS/SD, 48 tpi
FLM_200SSDD	(0x9)	8" (200 mm) SS/DD, 48 tpi
FLM_200DSDD	(0xA)	8" (200 mm) DS/DD, 48 tpi
FLM_130SSSD48	(0xD)	5.25" (130 mm) SS/SD, 48 tpi
FLM_130DSDD48	(0x12)	5.25" (130 mm) DS/DD, 48 tpi
FLM_130DSDD96	(0x16)	5.25" (130 mm) DS/DD, 96 tpi
FLM_130DSQD96	(0x1A)	5.25" (130 mm) DS/QD, 96 tpi
FLM_90DSDD135	(0x1E)	3.5" (90 mm) DS/DD, 135 tpi
FLSS_128	(0)	128 byte sectors
FLSS_256	(0x40)	256 byte sectors
FLSS_512	(0x80)	512 byte sectors
FLSS_1024	(0xC0)	1024 byte sectors
FLSS_2048	(0x100)	2048 byte sectors
FLSPT_5	(5<<10)	5 sectors per track
FLSPT_8	(8<<10)	8 sectors per track
FLSPT_9	(9<<10)	9 sectors per track
FLSPT_10	(10<<10)	10 sectors per track
FLSPT_15	(15<<10)	15 sectors per track
FLSPT_16	(16<<10)	16 sectors per track
FLSPT_18	(18<<10)	18 sectors per track
FLSPT_31	(31<<10)	31 sectors per track

The unit options defines are used in the *unit opts* field of the *rfdinfo* array to specify device information for the drives in your system.

Procedures for specifying flag and unit options defines vary with the device installed (disk, tape, or floppy).

Disk Drive Defines

The procedures for specifying defines for a disk drive are as follows:

1. Enter the following command to access the *cs35prm.h* file with the *vi* editor:

```
vi cs35int.h
```

2. Using the *vi* editor, enter the define statement(s) for the disk drive. Specify drive characteristics by "ORing" together the appropriate Flag defines (listed in Table C-1).

   For example, suppose you add the following drive reference to the *RIMFIRE* file:

```
#A SCSI Disk (Hathorpak)
disk cf2 at cfc1 drive Ø flags Ø
```

   The reference specifies a synchronous drive that takes a while to pass *"unit ready"* test and should not issue *"read capacity"* at open. You would enter the following define statement in the *cs35prm.h* file:

```
#define HATHORPAK (WD_NORMAL|SYNCHRONOUS|NOOPN_RDCAP|LONGRDY)
```

The following example illustrates the *rfdinfo* array in the *cs35int.h* file. The shaded portion of the array illustrates the added device-specific information for the defined disk drive.

```
struct device_word rfdinfo[] = {
/* device      dev      target   logical      unit      Partition */
/* index       id         id      unit        opts      number */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      Ø},     /* minor Ø */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      1},     /* minor 1 */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      2},     /* minor 2 */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      3},     /* minor 3 */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      4},     /* minor 4 */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      5},     /* minor 5 */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      6},     /* minor 6 */
    {Ø,      DIR_ACC,     5,       Ø,     Micropolis,      7},     /* minor 7 */
    {1,      SEQ_ACC,     3,       Ø,     Exabyte,         Ø},     /* minor 8 */
    {1,      SEQ_ACC,     3,       Ø,     Exabytenr,       Ø},     /* minor 9 */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       Ø},     /* minor 1Ø */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       1},     /* minor 11 */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       2},     /* minor 12 */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       3},     /* minor 13 */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       4},     /* minor 14 */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       5},     /* minor 15 */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       6}*     /* minor 16 */
    {2,      DIR_ACC,     4,       Ø,     Hathorpak,       7},     /* minor 17 */
```

3. Inspect the disk drive to insure that the drive settings match the *target id* and *logical unit* values specified in the *rfdinfo* array. The *target id* of the disk drive is set by changing the address of the drive (by changing switches, jumpers, etc.). The *logical unit* number is set by the manufacturer, but is usually adjustable (again, by changing switches, jumpers, etc.).

The following example illustrates the *rfdinfo* array in the *cs35int.h* file. The shaded portion of the array illustrates the added device specific information for the defined tape drive.

```
struct device_word rfdinfo[] = {
/* device      dev      target  logical      unit      Partition */
/* index       id        id      unit        opts      number */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      Ø},     /* minor Ø */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      1},     /* minor 1 */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      2},     /* minor 2 */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      3},     /* minor 3 */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      4},     /* minor 4 */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      5},     /* minor 5 */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      6},     /* minor 6 */
    {Ø,      DIR_ACC,     5,      Ø,     Micropolis,      7},     /* minor 7 */
    {1,      SEQ_ACC,     3,      Ø,     Exabyte,         Ø},     /* minor 8 */
    {1,      SEQ_ACC,     3,      Ø,     Exabytenr,       Ø},     /* minor 9 */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       Ø},     /* minor 1Ø */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       1},     /* minor 11 */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       2},     /* minor 12 */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       3},     /* minor 13 */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       4},     /* minor 14 */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       5},     /* minor 15 */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       6},     /* minor 16 */
    {2,      DIR_ACC,     4,      Ø,     Hathorpak,       7},     /* minor 17 */
    {3,      SEQ_ACC,     2,      Ø,     Anabasis,        Ø},     /* minor 18 */
    {3,      SEQ_ACC,     2,      Ø,     Anabasisnr,      Ø},     /* minor 19 */
```

3. Inspect the tape drive to ensure that the drive settings match the *target id* and *logical unit* values specified in the *rfdinfo* array. The *target id* of the drive is set by changing the address of the drive (by changing switches, jumpers, etc.). The *logical unit* number is set by the manufacturer, but is usually adjustable (again, by changing switches, jumpers, etc.).

Tape Drive Defines

The procedures for specifying defines for a tape drive are as follows:

1. Enter the following command to access the *cs35prm.h* file with the *vi* editor:

```
vi cs35int.h
```

2. Using the *vi* editor, enter the define statement(s) for the tape drive. Specify drive characteristics by "ORing" together the appropriate Flag defines (listed in Table C-1).

   For example, suppose you add the following drive reference to the *RIMFIRE* file:

```
#A SCSI Tape (Anabasis)
tape cf3 at cfc1 drive Ø flags Ø
```

   The reference specifies a drive that can write only one filemark at a time, runs in general mode, is used as a rewindable and no-rewind device, but is otherwise a normal tape device. You would enter the following define statement in the *cs35int.h* file to define the drive as a rewindable device:

```
#define ANABASIS (TP_NORMAL|GEN_MODE|ONEFILEMARK)
```

   To define the drive as a no-rewind device, you would also enter the following define statement:

```
#define ANABASISnr (TP_NORMAL|GEN_MODE|ONEFILEMARK|NOREWIND)
```

Floppy Drive Defines

The procedures for specifying defines for a floppy drive are as follows:

1. Enter the following command to access the *cs35flp.h* file with the *vi* editor:

```
vi cs35int.h
```

2. Using the *vi* editor, enter the define statement(s) for the floppy drive. Specify drive characteristics by "ORing" together the appropriate Flag defines (listed in Table C-3).

   For example, suppose you add the following drive reference to the *RIMFIRE* file:

```
#A Floppy Drive - special characteristics
disk cf4 at cfc1 drive Ø flags Ø
```

   The reference specifies a 5.25 inch DS/DD, 96 tpi, 128 byte sectors drive with 18 sectors per track. The following Flag defines would be used to specify the drive characteristics:

Characteristics	Flag Define
5.25", DS/DD, 96 tpi	FLM_130DSDD96
128 byte sectors	FLSS_128
18 sectors per track	FLSPT_18

   The following define statement (including the illustrated Flag defines) would be entered in the *cs35int.h* file:

```
#define FLOPPYØ (FLM_13ØDSDD96|FLSS_128|FLSPT_18)
```

The following example illustrates the *rfdinfo* array in the *cs35int.h* file. The shaded portion of the array illustrates the added device specific information for the defined floppy drive.

```
struct device_word rfdinfo[] = {
/* device       dev       target  logical      unit      Partition */
/* index         id         id      unit        opts      number */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     Ø},     /* minor Ø */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     1},     /* minor 1 */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     2},     /* minor 2 */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     3},     /* minor 3 */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     4},     /* minor 4 */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     5},     /* minor 5 */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     6},     /* minor 6 */
    {Ø,       DIR_ACC,      5,       Ø,     Micropolis,     7},     /* minor 7 */
    {1,       SEQ_ACC,      3,       Ø,     Exabyte,        Ø},     /* minor 8 */
    {1,       SEQ_ACC,      3,       Ø,     Exabytenr,      Ø},     /* minor 9 */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      Ø},     /* minor 1Ø */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      1},     /* minor 11 */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      2},     /* minor 12 */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      3},     /* minor 13 */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      4},     /* minor 14 */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      5},     /* minor 15 */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      6},     /* minor 16 */
    {2,       DIR_ACC,      4,       Ø,     Hathorpak,      7},     /* minor 17 */
    {3,       SEQ_ACC,      2,       Ø,     Anabasis,       Ø},     /* minor 18 */
    {3,       SEQ_ACC,      2,       Ø,     Anabasisnr,     Ø},     /* minor 19 */
    {4,       FLOPPY,     ØxFE,      Ø,     FLOPPYØ,        Ø},     /* minor 2Ø */
```

3. Inspect the floppy drive to ensure that the drive settings match the *target id* and *logical unit* values specified in the *rfdinfo* array. The *target id* is a fixed value (*ØxFE*) for any floppy drives used. The *logical unit* number corresponds to the physical address (jumper settings) selected on the drive. If more than one floppy device is used, no two physical addresses should be the same. Otherwise, addressing of both devices occurs.

Appendix D - Manually Making Device Nodes

Device nodes can be made manually or by using the *cs35mk* utility. To make nodes manually, you will need to enter a *mknod* command for each minor device number in the *rfdinfo* array (located in the *cs35int.h* file). The arguments entered for a *mknod* command depend on whether you are making nodes for a disk device or tape device.

Disk Devices

The following lines illustrate typical *mknod* commands for block and character devices respectively:

```
mknod /dev/rsAB b XX ZZ
mknod /dev/rrsAB c YY ZZ
```

Variables (illustrated in **bold** print) for the previous *mknod* commands are as follows:

- **A** - the Device Index in the *rfdinfo* array (located in the *sys/sundev/cs35int.h* header file)
- **B** - the disk partition (a=Ø, b=1, c=2, d=3, e=4, f=5,g=6, h=7)
- **XX** - the Block Device Major number from the *bdevsw* table in the */sys/sun/conf.c* file
- **YY** - the Character Device Major number from the *cdevsw* table in the */sys/sun/conf.c* file
- **ZZ** - the Minor number for the device from the *rfdinfo* array located in the */sys/sundev/cs35int.h* header file

➽ *NOTE: For use with the cs35ut utility program, device names must follow the structures illustrated.*

The UNIX Link command can be used if you require different node names.

Tape Devices

The following lines illustrate typical *mknod* commands for block and character devices respectively:

```
mknod /dev/rtA b XX ZZ
mknod /dev/rrtA c YY ZZ
```

Variables (illustrated in **bold** print) for the previous *mknod* commands are as follows:

- **A** - the device index from the *rfdinfo* array in the */sys/sundev/cs35int.h* header file
- **XX** - the Block Device Major number from the *bdevsw* table in the *sys/sun/conf.c* file
- **YY** - the Character Device Major number from the *cdevsw* table in the */sys/sun/conf.c* file
- **ZZ** - the Minor number for the device from the *rfdinfo* array in the */sys/sundev/cs35int.h* header file

➳ *NOTE: For use with the cs35ut utility program, device names must follow the structures illustrated.*

The UNIX Link command can be used if you require different node names.

Dummy Devices

The following lines illustrate typical *mknod* commands for block and character devices:

```
mknod /dev/rrdA c YY ZZ
```

Because the dummy device is used with the *cs35ut* utility, only the character node is required.

Variables (illustrated in **bold** print) for the previous *mknod* commands are as follows:

- **A** - the device index from the *rfdinfo* array in the */sys/sundev/cs35int.h* header file
- **YY** - the Character Device Major number from the *cdevsw* table in the */sys/sun/conf.c* file
- **ZZ** - the Minor number for the device from the *rfdinfo* array in the */sys/sundev/cs35int.h* header file

Appendix E - Error Codes

This appendix contains a listing of the error codes generated by the Rimfire 3500 adapter. Error codes are given in hexadecimal, and descriptions are included.

Rimfire 3500 Error Code Descriptions

01H Invalid Board Command

This is returned when a general board command which is not a recognized command number is issued. This error is returned only for general board commands (Target ID = FFH).

02H Bad Unit Or ID Number

This error is returned when a command is issued to a SCSI ID number which is greater than 07H and less than FEH. ID values of FEH and FFH are assigned special meanings for the Rimfire 3500 adapter, but all other ID numbers greater than 07H are impossible ID values.

03H Floppy Disk Option Not Installed

This error is generated when a command for the floppy disk unit is issued to a board which does not have the floppy disk option present. The host can check this by issuing an *Identify* command, which returns a flag in the Options Flag byte indicating the presence or absence of the floppy disk option.

06H Sector Count = 0

Sector count indicates disk boot emulation command is invalid.

07H Cylinder Address Error

Cylinder indicates disk boot emulation command is invalid.

08H Sector Address Error

Sector indicates disk boot emulation command is invalid.

09H Head Address Error

Head indicates disk boot emulation command is invalid.

0BH Reserved Field Not Zero

A *Reserved* field in the parameter block was found to contain a nonzero value. All *Reserved* fields must be set to zero to assure proper operation and allow for future enhancements to the board.

0EH Command List Stopped

This is not an error; it is returned in the *Error* field of the last status block placed in the command list when a *Stop Command List* command is executed. This code signals that the command list has actually been stopped and all parameter blocks in the list have completed.

0FH Bad Command List Size Field

During the execution of a *Start Command List* command, either the number of status blocks or the number of command blocks, or the total of the two, exceeded the maximum allowable value. A command list must be no longer than 64 Kbytes, counting the list header.

11H List Already Active

A *Start Command List* command was issued, but the command list was already active. This may happen if the list is being stopped, but has not yet had all pending commands completed.

14H Bus Timeout

A VMEbus transfer did not complete. The Rimfire 3500 adapter detects this with a software timer and aborts the command at fault.

15H Bus Error

During a data transfer, the bus did not respond or entered an illegal state. The Rimfire 3500 hardware detects this and returns this error code. This is most commonly the result of an invalid Source/Destination memory address in a parameter block. An invalid Address Modifier can also produce this error.

16H Scatter/Gather Descriptor Block Read Error

During a scatter/gather memory transfer, the board unsuccessfully attempted to transfer a new scatter/gather memory block. This is usually caused by a bad address in the scatter/gather chain.

1EH SCSI Select Timeout

An attempt to select a SCSI device failed because the device did not respond. This is usually caused by specifying an incorrect Target ID in a parameter block or by setting the wrong ID on the target itself.

1FH SCSI Disconnect Timeout

A device took too long to reselect the Rimfire 3500 adapter and continue an operation. This can be caused by a hardware error in the device. It can also be caused by an incorrect setting of the *Disconnect Timeout* length in the Unit Options parameter block. Tape devices, in particular, can take many minutes to finish operations such as rewinds or (for cartridge tape drives) searches for filemarks.

20H SCSI Parity Error

If the *General Options* command specified parity generation and detection for SCSI operations, this error code signals that the parity of the SCSI bus was found to be in error. This can be caused by hardware errors in the bus. It can also happen if one or more SCSI devices on the bus are not configured to generate parity. Parity is a system option and must be set the same for all devices.

21H Unexpected SCSI Disconnect

If, during a SCSI operation, the target disconnects unexpectedly, the Rimfire 3500 adapter will return this error. This indicates a problem with the device itself or with the cable.

22H General SCSI Bus Error

The SCSI interface chip on the Rimfire 3500 adapter detected an erroneous condition but did not give sufficient data to indicate the source of the error. This normally indicates an abnormal SCSI bus phase or a device which is violating bus specifications.

23H SCSI Device Returned Bad Status

On completion of a command to a SCSI device, the status returned was something other than *GOOD (0)*. The *SCSI Status* field in the status block contains the status returned. If the returned status was *CHECK CONDITION (2)* and the *IRS* bit was not set in the command, the status block also contains sense information about the error.

24H Unexpected SCSI Phase Encountered

During normal SCSI operations, the SCSI target requests the transfer of a number of different kinds of data, including commands out, status in, data in or out, and various messages in both directions. Each command has a normal sequence for these data phases, as well as defined variations for error conditions. The adapter will handle the management of these phases automatically; however, if the target requests a data transfer that cannot be interpreted (such as a message out when none was requested or a command out in the middle of a data transfer) the operation will be terminated and this error returned.

The *SCSI Status* field of the status block contains information on the phase encountered. The last three bits of the *SCSI Status* byte define the phase encountered and correspond to the *-MSG* (bit 2), *-C/D* (bit 1), and *-I/O* (bit Ø) lines on the SCSI bus.

The following example illustrates the *SCSI Status* byte and lists the phase definitions for the last three bits:

7	6	5	4	3	2	1	0
X	X	X	X	X	M	C	I

Bit	SCSI Bus Line	Setting	Description
M	-MSG	1	Requested phase was a Message phase
		Ø	Requested phase was a Data or Command Phase
C	-C/D	1	Command/status information was requested
		Ø	Ordinary data was requested
I	-I/O	1	Requested direction was in from the device
		Ø	Transfer was to be out to the device

This error can be caused by a number of things. If vendor-unique commands are being used or if the DBV bit in the parameter block is set to 1, a mis-setting of the DAT or DIR bits in the parameter block Flags byte can cause this error. It can also be generated by a hardware error in the device or in the adapter.

25H Bad Byte Seen by SCSI Adapter Chip

During the execution of a SCSI command, the adapter chip received a command, data, or status byte (usually a status byte) that was not interpretable.

This is normally caused by ID conflicts between the board and the device, or by mistakes in parity setting.

26H Error in Synchronous Transfer Negotiation

The Rimfire 3500 adapter attempted to negotiate synchronous transfer parameters (transfer period and maximum REQ/ACK offset), but the target does not support the negotiation and responded improperly. This error will not occur when the target negotiates correctly, even if the target itself does not support synchronous transfers. It will occur only if the target responds with incorrect or incomprehensible requests during the negotiation process.

If this error occurs, the adapter will clear the error condition and abort the command, which may require a device or bus reset. If the SCSI bus is reset, this may affect the operation of other devices on the bus that were active at the time of the negotiation. This error can be avoided by disabling the synchronous transfer option for devices which do not support negotiation.

27H SCSI Bus Reset During Operation

This error indicates that the SCSI bus was reset (by the */RST* signal line) during a SCSI operation. Any device may reset the SCSI bus. This error occurs if a device other than the Rimfire 3500 adapter resets the bus while the Rimfire 3500 adapter is executing a command. The operation in process is aborted, but subsequent commands will be attempted. Note that a reset may cause errors in subsequent commands while other devices connected to the bus go through their power-on/reset recovery sequences.

28H Target Command not Found

The Rimfire 3500 HBA has not received an *Enable Target Mode* command from the host. As a result, it cannot act as a Target.

29H This command must be issued with a command list

The command that was issued requires a command list. Reissue the command with a command list.

2AH Drive is write protected

Disable drive write protection and reissue the command.

2BH Vendor Unique command set up improperly, modifier field zero.

2CH Bad SCSI chip condition

The SCSI adapter chip received a bad command or the wrong target reselected the adapter.

When this error occurs, the *SCSI STATUS* field in the status block contains the status register value from the SCSI adapter chip.

Diagnostic Errors

61H Static RAM Error

The Rimfire 3500 adapter static RAM is not working properly. The status block contains the address and expected/found data for the error.

62H PROM Checksum Error

The Rimfire 3500 firmware does not contain the appropriate checksum, a PROM is incorrect or defective, or hardware has failed. The firmware PROMs must be replaced to guarantee correct board operation.

63H Undefined Diagnostic Specified

The *Diagnostic* command selected a bit for a diagnostic procedure that was not defined. This is the only diagnostic error which is caused by a bad setting in the parameter block.

Internal Errors

80H and above

Firmware errors. The code and the circumstances of the error should be reported to Ciprico.

Boot Emulation Errors

Error codes used by the Rimfire 3523 adapterer depend on the boot emulation disable jumper setting. The following table lists Rimfire error codes and their corresponding boot emulation error codes.

Rimfire Error		Tape	Disk	Emulation Error
00H	Successful completion	00H	00H	Successful completion
01H	Invalid command	15H	15H	Unimplemented cmd
02H	Bad Unit number	04H	04H	Operation timeout
03H	Floppy disk option not installed	04H	04H	Operation timeout
04H	Not used	01H	01H	Interrupt pending
05H	Not used	01H	01H	Interrupt pending
06H	Not used Sector count = 0	01H	17H	Interrupt pending Sector count = 0
07H	Not used Cylinder address error	01H	07H	Interrupt pending Cylinder Address Error
08H	Not used Sector address error	01H	0AH	Interrupt pending Sector address error
09H	Not used Head address error	01H	20H	Interrupt pending Head address error
0AH	Not used	01H	01H	Interrupt pending
0BH	Field not zero	03H	03H	Busy conflict
0CH	Not used	01H	01H	Interrupt pending
0DH	Not used	01H	01H	Interrupt pending
0EH	Command list stopped	03H	03H	Busy conflict
0FH	Bad Command List size	03H	03H	Busy conflict
10H	Not used	01H	01H	Interrupt pending
11H	Command list already active	03H	03H	Busy conflict
12H	Not used	01H	01H	Interrupt pending
13H	Not used	01H	01H	Interrupt pending
14H	Bus timeout	04H	04H	Operation timeout
15H	Bus error	04H	04H	Operation timeout
16H	Scatter/gather descriptor block read error	03H	03H	Busy conflict

Rimfire Error		Tape	Disk	Emulation Error
17H	Not used	01H	01H	Interrupt pending
18H	Not used	01H	01H	Interrupt pending
19H	Not used	01H	01H	Interrupt pending
1AH	Not used	01H	01H	Interrupt pending
1BH	Not used	01H	01H	Interrupt pending
1CH	Not used	01H	01H	Interrupt pending
1DH	Not used	01H	01H	Interrupt pending
1EH	SCSI select timeout	16H	04H	Drive offline
1FH	SCSI disconnect timeout	04H	04H	Operation timeout
20H	SCSI parity error	06H	18H	Hard error
21H	Unexpected SCSI disconnect	06H	18H	Hard error
22H	General SCSI bus error	06H	18H	Hard error
23H	SCSI device returned bad status	06H	18H	Hard error
24H	Unexpected SCSI phase encountered	06H	18H	Hard error
25H	Bad byte seen by SCSI adapter chip	06H	18H	Hard error
26H	Error in synchronous transfer negotiation	06H	18H	Hard error
27H	Error in scatter/gather operation	03H	03H	Busy conflict
28H	SCSI bus reset during operation	06H	18H	Hard error
29H	Not used	01H	01H	Interrupt pending
2AH	Not used	01H	01H	Interrupt pending
2BH	Vendor-unique cmd not set up properly	03H	03H	Busy conflict
2CH	Emulation not supported	15H	15H	Unimplemented cmd

Appendix F - Cables and Connections

Figures F-1 and F-2 illustrate cabling and board interconnection for the Rimfire 3500 Host Bus Adapter.

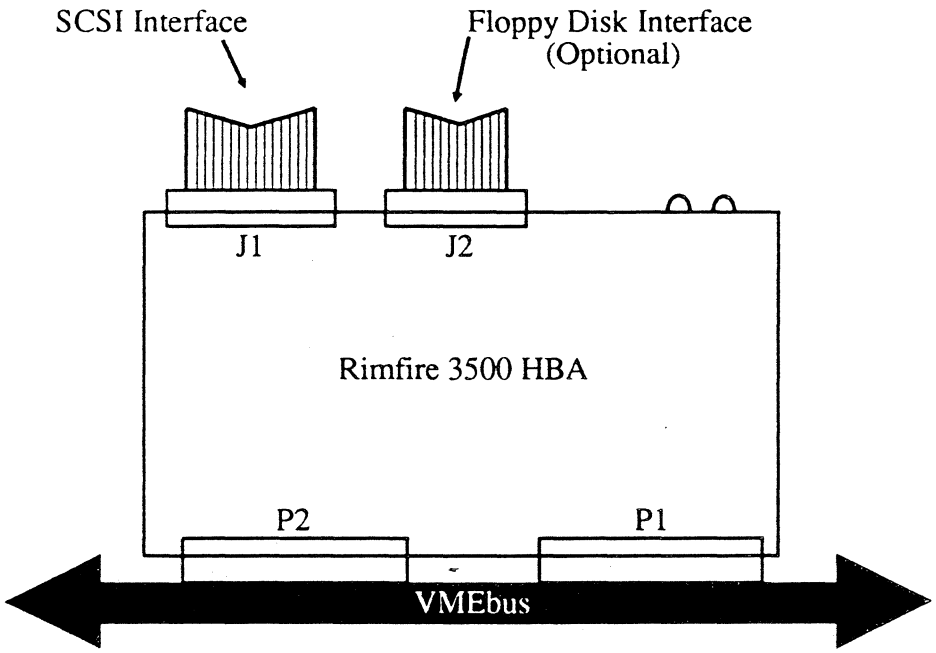


Figure F-1 Rimfire 3500 Interconnection Diagram

Table F-1 lists suggested cable parts.

➽ *NOTE: On the Rimfire 3517 and Rimfire 3518 HBAs, the SCSI interface may be through P2. For the Rimfire 3517, this is determined by the SCSI Port Selection jumper (J33) setting.*

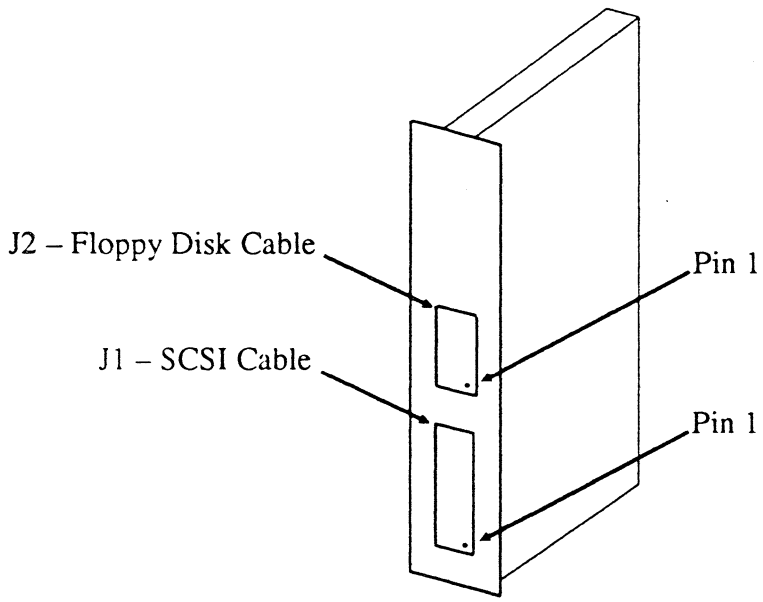


Figure F-2 Rimfire 3500 Series Cable Connections

Table F-1 Rimfire 3500 Series Cable Parts

Cable	Quantity	Description	Suggested Parts
J1	As Required	50 Conductor Flat Ribbon Cable (10 Feet)	3M 3365-50
	1	50 Pin Connector, Socket, Without Strain Relief	3M 3425-6000 T&B Ansley 609-5000M
J2	As Required	34 Conductor Flat Ribbon Cable (10 Feet)	3M 3365-34
	1	34 Pin Connector, Socket	3M 3414-6034 T&B Ansley 609-3401M
	1	34 Pin Card Edge Connector	3M 3463-0001 T&B Ansley 609-3415M

➳ *NOTE: J1 cable may be daisy chained if multiple drives are used.*

3M is a registered trademark of 3M company. T&B Ansley is a registered trademark of Thomas & Betts.

Figures F-3 illustrates cabling and board connection for the Rimfire 3523 SCSI Boot Adapter.

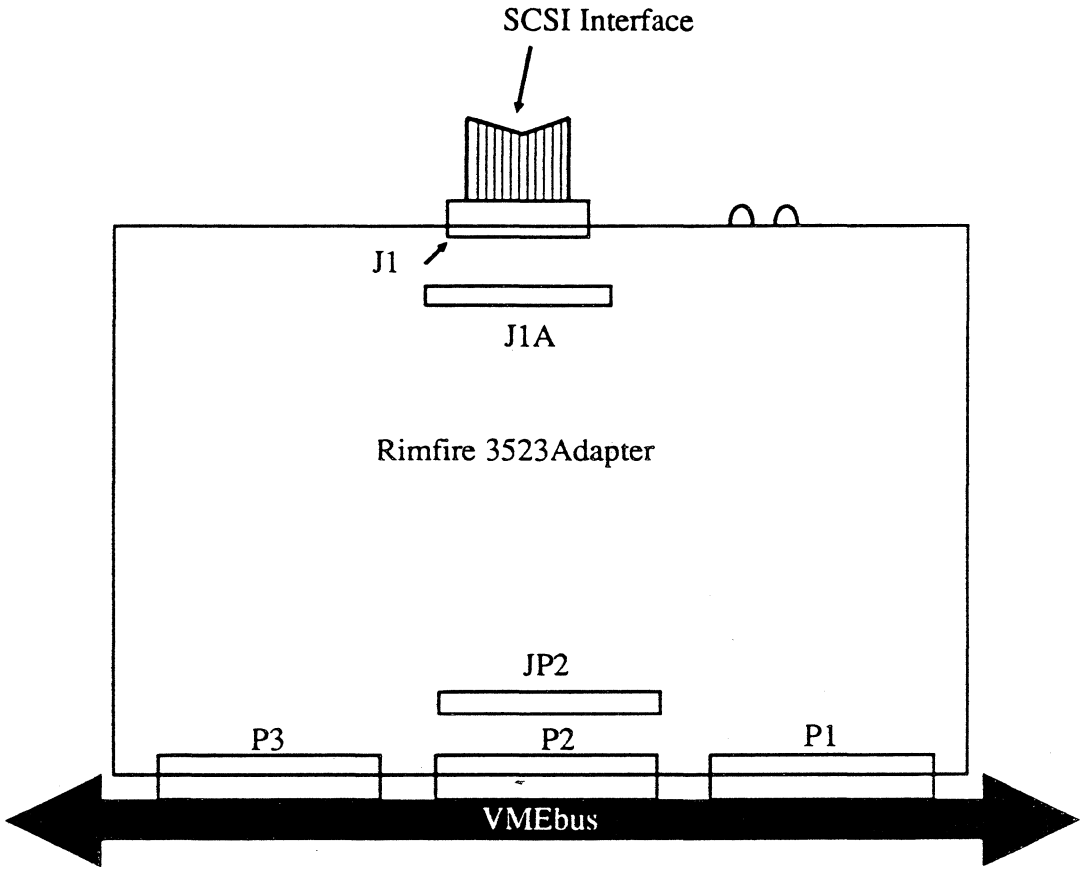


Figure F-3 Rimfire 3523 Interconnection Diagram

The SCSI interface on the Rimfire 3523 adapter can be through P2. You should use a sheilded cable from the Rimfire 3523 adapter to a metal cabinet containing the peripheral devices. Inside the cabinet, an unsheilded , flat-ribbon cable allows the cable to be daisey-chained more easily. The cumulative length of the cables is six meters. If the cabinet is made so the cable exits to another cabinet, but no cabinet is attached, terminate the SCSI cable. An external terminator is available from Methode Electronics, Inc., part number DM-900.

Tables F-2, F-3, and F-4 list cable parts for the Rimfire 3523 bootable adapter.

Table F-2 Rimfire 3523 P2 SCSI On-board Cable

Quantity	Description	Suggested Parts
As Required	15-inch, 50-Conductor, Flat-Ribbon Cable	3M 3365-50
2	50-Pin Connector, Socket, Without Strain Relief	3M 3425-6000

Table F-3 Rimfire 3523 Inside Cabinet Cable (Unshielded)

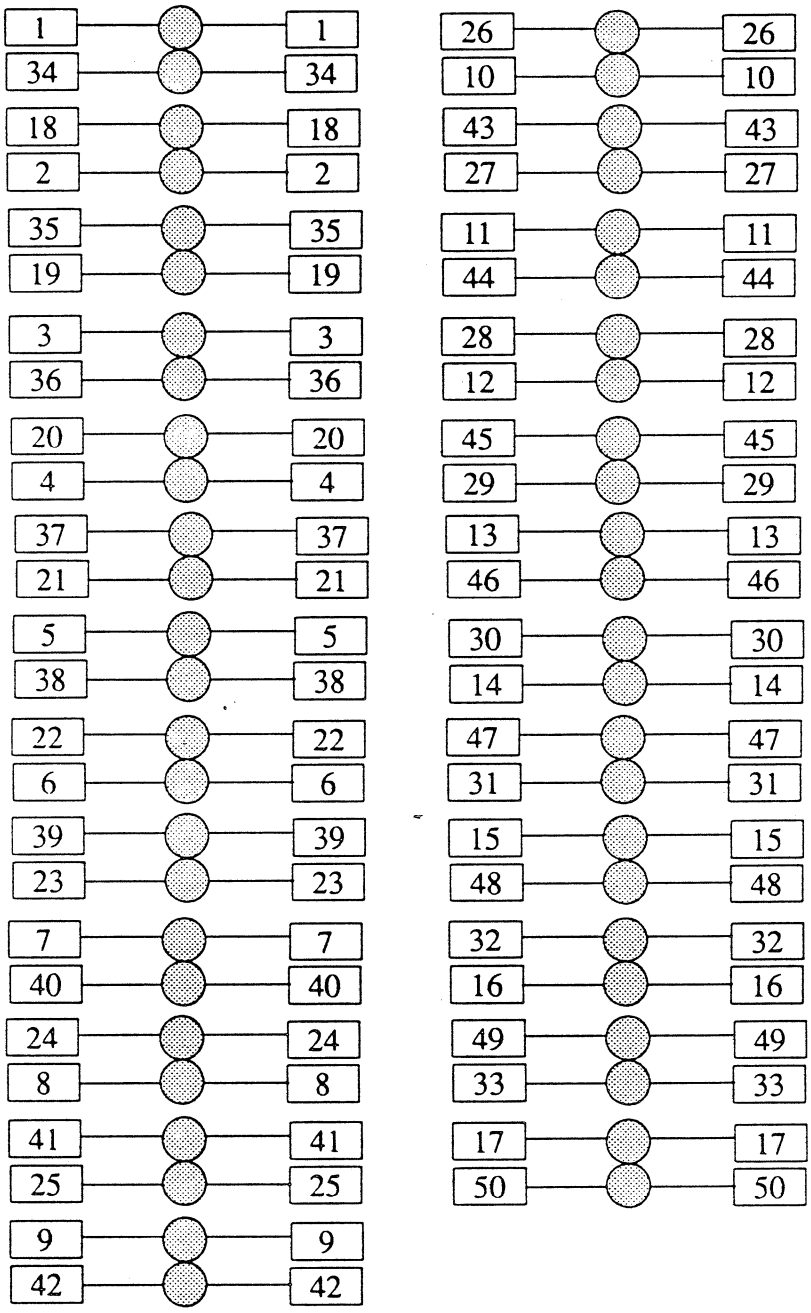
Quantity	Description	Suggested Parts
As Required	50-Conductor, Flat-Ribbon Cable	3M 3365-50
1 (2 if in-out)	50-Pin Subminiature D IDC Connector	T&B Ansley 609-50S-M
1 per device	50-Pin , Connector, Socket, without strain relief	3M 3425-6000

Table F-4 Rimfire 3523 J1 Shielded Cable

Quantity	Description	Suggested Parts
As Required	25-pair, Shielded, 100-OHM Impedance, SCSI-compatible	
2	Connector Body	AMP 205212-3
100	Connector Pins	AMP 66507-9
2	Metal Backshell	AMP 745175-3

AMP is a trademark of AMP, Inc.

Following are the 50-pin D-subminiature suggested pin pairings:



Tables F-5 thru F-10 list connector pin assignments for the Rimfire 3500 series and the Rimfire 3523 adapters.

Table F-5 J1 - SCSI Single Ended Interface

3523 Pin	3500 Pin	Signal Name
34	2	-DATA BUS (0)
2	4	-DATA BUS (1)
19	6	-DATA BUS (2)
36	8	-DATA BUS (3)
4	10	-DATA BUS (4)
21	12	-DATA BUS (5)
38	14	-DATA BUS (6)
6	16	-DATA BUS (7)
23	18	-DATA BUS (P)
40	20	GROUND
8	22	GROUND
25	24	GROUND
42	26	TERMINATION POWER
10	28	GROUND
27	30	GROUND
44	32	-ATTENTION
12	34	GROUND
29	36	-BUSY
46	38	-ACKNOWLEDGE
14	40	-RESET
31	42	-MESSAGE
48	44	-SELECT
16	46	-CONTROL/DATA
33	48	-REQUEST
50	50	-INPUT/OUTPUT

➽ *NOTE: A minus sign (-) next to a Signal Name indicates the signal is active low.*

All odd numbered pins except 25 are connected to ground. Pin 25 is left open. Some SCSI products that were designed before the SCSI standard have pin 25 connected to ground.

On the 3523, other numbered pins except 9 are connected to ground. Pin 9 (pin 25 on the SCSI flat-ribbon header) is left open. Some SCSI products were designed before the SCSI standard connected this to ground.

Table F-6 J1 - SCSI Differential Interface

Pin	Signal Name	Pin	Signal Name
1	SHIELD GROUND	2	GROUND
3	+DATA BUS (0)	4	-DATA BUS (0)
5	+DATA BUS (1)	6	-DATA BUS (1)
7	+DATA BUS (2)	8	-DATA BUS (0)
9	+DATA BUS (3)	10	-DATA BUS (1)
11	+DATA BUS (4)	12	-DATA BUS (0)
13	+DATA BUS (5)	14	-DATA BUS (1)
15	+DATA BUS (6)	16	-DATA BUS (0)
17	+DATA BUS (7)	18	-DATA BUS (1)
19	+DATA BUS (8)	20	-DATA BUS (P)
21	DIFFERENTIAL SENSE	22	GROUND
23	GROUND	24	GROUND
25	TERMINATION POWER	26	TERMINATION POWER
27	GROUND	28	GROUND
29	+ATTENTION	30	-ATTENTION
31	GROUND	32	GROUND
33	+BUSY	34	-BUSY
35	+ACKNOWLEDGE	36	-ACKNOWLEDGE
37	+RESET	38	-RESET
39	+MESSAGE	40	-MESSAGE
41	+SELECT	42	-SELECT
43	+CONTROL/DATA	44	-CONTROL/DATA
45	+REQUEST	46	-REQUEST
47	+INPUT/OUTPUT	48	-INPUT/OUTPUT
49	GROUND	50	GROUND

➽ *NOTE: A plus sign (+) next to a signal name indicates the positive line of a differential signal pair. A minus sign (-) next to a signal name indicates the negative line of a differential signal pair.*

Table F-7 J2 - Floppy Disk Interface

Pin	Signal Name
2	ACTIVE READ FILTER
4	HEAD LOAD
6	DRIVE SELECT 3
8	INDEX/SECTOR
10	DRIVE SELECT 0
12	DRIVE SELECT 1
14	DRIVE SELECT 2
16	MOTOR ON
18	DIRECTION
20	STEP
22	WRITE DATA
24	WRITE GATE
26	TRACK 0
28	WRITE PROTECTED
30	READ DATA
32	SIDE ONE SELECT
34	READY

➳ *NOTE: All odd numbered pins are connected to ground.*

The ACTIVE READ FILTER signal (Pin 2) may have different designations, depending on the particular drive used (i.e., REDUCED WRITE CURRENT SELECT or HIGH DENSITY SELECT on some AT compatibles).

Table F-8 P1 - VMEbus Pin Assignments

VME Pin	Row A	Row B	Row C
1	+D00	-BBSY	+D08
2	+D01	-BCLR	+D09
3	+D02	-ACFAIL	+D10
4	+D03	-BG0IN	+D11
5	+D04	-BG0OUT	+D12
6	+D05	-BG1IN	+D13
7	+D06	-BG1OUT	+D14
8	+D07	-BG2IN	+D15
9	GND	-BG2OUT	GND
10	+SYSCLK	-BG3IN	-SYSFAIL
11	GND	-BG3OUT	-BERR
12	-DS1	-BR0	-SYSRESET
13	-DS0	-BR1	-LWORD
14	-WRITE	-BR2	+AM5
15	GND	-BR3	+A23
16	-DTACK	+AM0	+A22
17	GND	+AM1	+A21
18	-AS	+AM2	+A20
19	GND	+AM3	+A19
20	-IACK	GND	+A18
21	-IACKIN	SERCLK *	+A17
22	-IACKOUT	SERDAT *	+A16
23	+AM4	GND	+A15
24	+A07	-IRQ7	+A14
25	+A06	-IRQ6	+A13
26	+A05	-IRQ5	+A12
27	+A04	-IRQ4	+A11
28	+A03	-IRQ3	+A10
29	+A02	-IRQ2	+A09
30	+A01	-1RQ1	+A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

* *Indicates unused pin assignments.*

Table F-9 P2 - Rimfire 3500 VMEbus/SCSI Pin Assignments

Pin	All	Rimfire 3517 Only		Rimfire 3518 Only	
	Row B (VMEbus)	Row A (SCSI)	Row C (SCSI)	Row A (SCSI)	Row C (SCSI)
1	+5V	-DATA BUS (0)	GROUND	GROUND	SHIELD GROUND
2	GND	-DATA BUS (1)	GROUND	-DATA BUS (0)	+DATA BUS (0)
3	Res.	-DATA BUS (2)	GROUND	-DATA BUS (1)	+DATA BUS (1)
4	+A24	-DATA BUS (3)	GROUND	-DATA BUS (2)	+DATA BUS (2)
5	+A25	-DATA BUS (4)	GROUND	-DATA BUS (3)	+DATA BUS (3)
6	+A26	-DATA BUS (5)	GROUND	-DATA BUS (4)	+DATA BUS (4)
7	+A27	-DATA BUS (6)	GROUND	-DATA BUS (5)	+DATA BUS (5)
8	+A28	-DATA BUS (7)	GROUND	-DATA BUS (6)	+DATA BUS (6)
9	+A29	-DATA BUS (P)	GROUND	-DATA BUS (7)	+DATA BUS (7)
10	+A30	GROUND	GROUND	-DATA BUS (P)	+DATA BUS (P)
11	+A31	GROUND	GROUND	GROUND	DIFFERENTIAL SENSE
12	GND	GROUND	GROUND	GROUND	GROUND
13	+5V	TERMINATION POWER	**	TERMINATION POWER	TERMINATION POWER
14	+D16	GROUND	GROUND	GROUND	GROUND
15	+D17	GROUND	GROUND	-ATTENTION	+ATTENTION
16	+D18	-ATTENTION	GROUND	GROUND	GROUND
17	+D19	GROUND	GROUND	-BUSY	+BUSY
18	+D20	-BUSY	GROUND	-ACKNOWLEDGE	+ACKNOWLEDGE
19	+D21	-ACKNOWLEDGE	GROUND	-RESET	+RESET
20	+D22	-RESET	GROUND	-MESSAGE	+MESSAGE
21	+D23	-MESSAGE	GROUND	-SELECT	+SELECT
22	GND	-SELECT	GROUND	-CONTROL/DATA	+CONTROL/DATA
23	+D24	-CONTROL/DATA	GROUND	-REQUEST	+REQUEST
24	+D25	-REQUEST	GROUND	-INPUT/OUTPUT	+INPUT/OUTPUT
25	+D26	-INPUT/OUTPUT	GROUND	GROUND	GROUND
26	+D27	*	*	*	*
27	+D28	*	*	*	*
28	+D29	*	*	*	*
29	+D30	*	*	*	*
30	+D31	*	*	*	*
31	GND	*	*	*	*
32	+5V	*	*	*	*

* *Indicates unused pin assignments.*

** *Pin C13 (pin 25 on the SCSI flat ribbon header) is left open. Some SCSI products were designed before the SCSI Standard connected this to ground.*

Rimfire 3517 *- A minus sign (-) next to a signal name indicates the signal is active low.*

Rimfire 3518 *- A plus sign (+) next to the signal name indicates a positive line of a differential pair. A minus sign (-) indicates a negative line of a differential pair.*

Table F-10 P2 - Rimfire 3523 VMEbus/SCSI Pin Assignments

Pin	All	Rimfire 3523	Rimfire 3523
	Row B (VMEbus)	Row A (SCSI)	Row A (SCSI)
1	+5V	*	*
2	GND	*	*
3	Res.	*	*
4	+A24	*	*
5	+A25	*	*
6	+A26	*	*
7	+A27	*	*
8	+A28	GROUND	-INPUT/OUTPUT
9	+A29	GROUND	-REQUEST
10	+A30	GROUND	-CONTROL/DATA
11	+A31	GROUND	-SELECT
12	GND	GROUND	-MESSAGE
13	+5V	GROUND	-RESET
14	+D16	GROUND	-ACKNOWLEDGE
15	+D17	GROUND	-BUSY
16	+D18	GROUND	GROUND
17	+D19	GROUND	-ATTENTION
18	+D20	GROUND	GROUND
19	+D21	GROUND	GROUND
20	+D22	**	TERMINATION POWER
21	+D23	GROUND	GROUND
22	GND	GROUND	GROUND
23	+D24	GROUND	GROUND
24	+D25	GROUND	-DATA BUS (P)
25	+D26	GROUND	-DATA BUS (7)
26	+D27	GROUND	-DATA BUS (6)
27	+D28	GROUND	-DATA BUS (5)
28	+D29	GROUND	-DATA BUS (4)
29	+D30	GROUND	-DATA BUS (3)
30	+D31	GROUND	-DATA BUS (2)
31	GND	GROUND	-DATA BUS (1)
32	+5V	GROUND	-DATA BUS (0)

* *Indicates unused pin assignments.*

** *Pin A20 (pin 25 on the SCSI flat ribbon header) is left open. Some SCSI products were designed before the SCSI Standard connected this to ground.*

Appendix G - Manual Driver Installation

Perform the procedures in this chapter to manually install the driver for the Ciprico Rimfire 3500 adapter. It is distributed on 1/4-inch and 1/2-inch tape. Contained on the tapes are a number of files: *cs35.c, cs35lib.c, cs35if.h, cs35prm.h, cs35io.h, cs35err.h, cs35int.h, cs35flp.h, cs35ut.c, cs35mk.c, Makefile,* and *README.*

Driver Installation - SunOS 3.5 or Earlier

1. Enter the following commands to create a directory called */sys/CIPRICO:*

   ```
   mkdir  /sys/CIPRICO
   ```

   Enter the following command to switch to the directory you created:

   ```
   cd  /sys/CIPRICO
   ```

2. Use the *tar* command to copy all files to the */sys/CIPRICO directory. If you are using 1/4 inch tape, enter the following tar* command:

   ```
   tar xvbf 2Ø /dev/rst8
   ```

   If you are using 1/2 inch tape, enter the following *tar* command:

   ```
   tar xvbf 126 /dev/rmtØ
   ```

➽ *NOTE: Throughout this procedure you are asked to copy files to a file with the same name and a .nocf suffix. Doing so allows you to use the upgrade and uninstall programs of the installation script.*

For the following steps, replace SunX with the designation for the Sun system you are using (i.e., Sun3 for Sun- 3 Workstations, Sun4 for Sun-4 Workstations, Sun3x for 68030 Workstations,etc.).

3. Enter the following commands to change to the /sys/conf directory and copy the current system configuration file. If this is a new installation, the file will be named GENERIC. Otherwise, consult the System Administrator for the correct file name. Enter a command similar to the following to copy the configuration file to the new file name (*RIMFIRE*):

```
cd /sys/conf
cp GENERIC RIMFIRE
```

Enter this command to copy the *RIMFIRE* file to *RIMFIRE.nocf*:

```
cp RIMFIRE RIMFIRE.nocf
```

4. Enter the following command to access the *RIMFIRE* file with the *vi* editor:

```
vi RIMFIRE
```

Search the *RIMFIRE* file for the following *config* line:

```
config vmunix swap generic
```

You can specify the root and swap devices now, or leave the setting at generic, allowing root and swap device specification at boot.

If the new Rimfire controlled disk is used as the root and swap device, edit the config line to look like this:

```
config vmunix root on cf0 swap on cf0
```

➳ *NOTE: If the driver was installed with different root and swap locations, consult your system administrator before changing root and swap locations.*

For each adapter installed, you will need to add an adapter line to the *RIMFIRE* file. The following example illustrates a typical adapter line:

```
controller cfc# at vme16d32 ? csr 0xc#### priority 2 vector cfintr 0xF#
```

Variables specifying your particular system configuration (indicated in **bold** print) are as follows:

- ***cfc#*** indicates the Rimfire adapter being installed. This variable is incremented for each adapter in the system. *cfc#* entries and their respective adapter distinctions are as follows:

Adapter	*cfc#*
1st	*cfcØ*
2nd	*cfc1*
3rd	*cfc2*
4th	*cfc3*

- ***Øx####*** indicates the board address. During hardware installation (see page 2- 2), you were instructed to check the Board Address jumper settings for proper addressing. Enter the address currently set on the board address jumpers (A15-A9).
- ***ØxF#*** indicates the interrupt vector. The interrupt vector can be assigned any unique, single byte value. Each adapter in your system should have a unique interrupt vector value. Typically, *ØxFE* is used for the first Rimfire adapter in the system and *ØxFA* is used for the second Rimfire adapter in the system.

For example, suppose you are installing two Rimfire adapters. The Board Address jumpers (A15-A9) on the first Rimfire adapter are set for an address of *Øx5ØØØ*. The Board Address jumpers (A15-A9) on the second Rimfire adapter are set for an address of *Øx55ØØ*. The following adapter lines would be added to the *RIMFIRE* file:

```
controller cfcØ at vmel6d32 ? csr Øx5ØØØ priority 2 vector cfintr ØxFE
controller cfcl at vmel6d32 ? csr Øx55ØØ priority 2 vector cfintr ØxFA
```

5. You will also need to add references for each disk or tape drive in your system. The following lines illustrate typical tape drive and disk drive references:

```
tape cf# at cfc# drive Ø flags Ø
disk cf# at cfc# drive Ø flags Ø
```

Variables specifying your particular system configuration (indicated in **bold** print) are as follows:

- *cf#* represents the Sun logical unit value to assign to the drive (corresponding to the device index number in the *rfdinfo* array). The *cf#* reference is strictly sequential.
- *cfc#* indicates the adapter to which the drive is attached. This variable is incremented for each adapter in the system. *cfc#* entries and their respective adapter distinctions are as follows:

Adapter	*cfc#*
1st	*cfcØ*
2nd	*cfc1*
3rd	*cfc2*
4th	*cfc3*

When adding drive references, it is a good idea to precede each reference with a comment indicating the drive type. The following example illustrates drive references for a seven-drive system with two adapters:

```
#A Hard Disk (Miniscribe)
disk cfØ at cfcØ drive Ø flags Ø
#A Hard Disk (WrenV)
disk cf1 at cfcØ drive Ø flags Ø
#A Hard Disk (Micropolis)
disk cf2 at cfcØ drive Ø flags Ø
#A Hard Disk (Micropolis)
disk cf3 at cfc1 drive Ø flags Ø
#A SCSI Tape - (Archive)
tape cf4 at cfc1 drive Ø flags Ø
#A SCSI Tape - (Wangtek)
tape cf5 at cfc1 drive Ø flags Ø
#A Dummy device
disk cf6 at cfc1 drive Ø flags Ø
```

6. Move the cursor to the xy entries. The xy entries are similar to the following lines:

```
controller xycØ at vme16d16 ? csr ØxEE4Ø priority 2 vector xyintr Øx48
controller xyc1 at vme16d16 ? csr ØxEE48 priority 2 vector xyintr Øx49
disk xyØ at xycØ drive Ø
disk xy1 at xycØ drive 1
disk xy2 at xyc1 drive Ø
disk xy3 at xyc1 drive 1
```

The disk boot emulation controller must be addressed at *ØxEE4Ø*. Other controllers cannot be present at this address. You must comment out any xy entry with a *csr* address equaling the *ØxEE4Ø* boot emulation address. Do this by adding the comment symbol (#) to the beginning of the appropriate line.

For example, if you are installing the Rimfire 3523 controller as the primary boot controller, the boot emulation address for the controller is *ØxEE4Ø*. In such a case, you must comment out the first xy entry (*xycØ)* and its associated disk entries (*xyØ* and *xyl*). Use your editor to add the comment symbol (#), as illustrated in this example:

```
#controller xycØ at vme16d16 ? csr ØxEE4Ø priority 2 vector xyintr Øx48
controller xycl at vme16d16 ? csr ØxEE48 priority 2 vector xyintr Øx49
#disk xyØ at xycØ drive Ø
#disk xyl at xycØ drive 1
disk xy2 at xycl drive Ø
disk xy3 at xycl drive 1
```

7. Move the cursor to the xt entries. The xt entries are similar to these lines:

```
controller xtcØ at vme16d16 ? csr ØxEE6Ø priority 2 vector xtintr Øx64
controller xtcl at vme16d16 ? csr ØxEE68 priority 2 vector xtintr Øx65
```

The tape boot emulation controller must be addressed at *ØxEE6Ø*. Other controllers cannot be at this address. You must comment out any xt entries with a *csr* address equaling the boot emulation address of the controller you are installing by adding the comment symbol (#) to the beginning of the appropriate line.

For example, if you are installing the Rimfire 3523 controller as the bootable tape controller, the boot emulation address for the controller is *ØxEE6Ø*. In such a case, you must comment out the first xt entry (*xtcØ)* and its associated tape entry (*xtØ*). Use your editor to add the comment symbol (#), as illustrated in this example:

```
#controller xtcØ at vme16d16 ? csr ØxEE6Ø priority 2 vector xtintr Øx64
controller xtcl at vme16d16 ? csr ØxEE68 priority 2 vector xtintr Øx65
#tape xtØ at xtcØ drive 1
tape xtl at xtcØ drive 1
```

8. Also, you must add device-specific information for the drive(s) to the *rfdinfo* array. The *rfdinfo* array is in the *cs35int.h* file (located in the */sys/CIPRICO/cf* directory).

   Enter the following commands to change directories to */sys/CIPRICO/cf* and access *cs35int.h* with the *vi* editor:

   ```
   cd /sys/CIPRICO/cf/sundev
   vi cs35int.h
   ```

   Add the necessary device-specific information.

The following example illustrates the *rfdinfo* array:

```
struct device_word rfdinfo[] = {
/* device      dev        target     logical      unit    partion */
/* index       id           id         unit       opts    number  */
   {0,        DIR_ACC,       0,          0,     Miniscribe,0},   /* minor 0 */
   {0,        DIR_ACC,       0,          0,     Miniscribe,1},   /* minor 1 */
   {0,        DIR_ACC,       0,          0,     Miniscribe,2},   /* minor 2 */
   {0,        DIR_ACC,       0,          0,     Miniscribe,3},   /* minor 3 */
   {0,        DIR_ACC,       0,          0,     Miniscribe,4},   /* minor 4 */
   {0,        DIR_ACC,       0,          0,     Miniscribe,5},   /* minor 5 */
   {0,        DIR_ACC,       0,          0,     Miniscribe,6},   /* minor 6 */
   {0,        DIR_ACC,       0,          0,     Miniscribe,7},   /* minor 7 */
   {1,        DIR_ACC,       1,          0,     WrenV,     0},   /* minor 8 */
   {1,        DIR_ACC,       1,          0,     WrenV,     1},   /* minor 9 */
   {1,        DIR_ACC,       1,          0,     WrenV,     2},   /* minor 10 */
   {1,        DIR_ACC,       1,          0,     WrenV,     3},   /* minor 11 */
   {1,        DIR_ACC,       1,          0,     WrenV,     4},   /* minor 12 */
   {1,        DIR_ACC,       1,          0,     WrenV,     5},   /* minor 13 */
   {1,        DIR_ACC,       1,          0,     WrenV,     6},   /* minor 14 */
   {1,        DIR_ACC,       1,          0,     WrenV,     7},   /* minor 15 */
   {2,        DIR_ACC,       2,          0,     Micropolis,0},   /* minor 16 */
   {2,        DIR_ACC,       2,          0,     Micropolis,1},   /* minor 17 */
   {2,        DIR_ACC,       2,          0,     Micropolis,2},   /* minor 18 */
   {2,        DIR_ACC,       2,          0,     Micropolis,3},   /* minor 19 */
   {2,        DIR_ACC,       2,          0,     Micropolis,4},   /* minor 20 */
   {2,        DIR_ACC,       2,          0,     Micropolis,5},   /* minor 21 */
   {2,        DIR_ACC,       2,          0,     Micropolis,6},   /* minor 22 */
   {2,        DIR_ACC,       2,          0,     Micropolis,7},   /* minor 23 */
   {3,        DIR_ACC,       3,          0,     Micropolis,0},   /* minor 24 */
   {3,        DIR_ACC,       3,          0,     Micropolis,1},   /* minor 25 */
   {3,        DIR_ACC,       3,          0,     Micropolis,2},   /* minor 26 */
   {3,        DIR_ACC,       3,          0,     Micropolis,3},   /* minor 27 */
   {3,        DIR_ACC,       3,          0,     Micropolis,4},   /* minor 28 */
   {3,        DIR_ACC,       3,          0,     Micropolis,5},   /* minor 29 */
   {3,        DIR_ACC,       3,          0,     Micropolis,6},   /* minor 30 */
   {3,        DIR_ACC,       3,          0,     Micropolis,7},   /* minor 31 */
   {4,        SEQ_ACC,       4,          0,     Archive,0},      /* minor 32 */
   {4,        SEQ_ACC,       4,          0,     Archivenr,0},    /* minor 33 */
   {5,        SEQ_ACC,       5,          0,     WANGTEK,0},      /* minor 34 */
   {5,        SEQ_ACC,       5,          0,     WANGTEKnr,0},    /* minor 35 */
   {6,        DUMMY,     NOT_USED,   NOT_USED,  NOT_USED,0},     /* minor 36 */
};
```

➽ *Note:* *If you modify the rfdinfo array, the cs35mk.c make node utility must be recompiled before using it to make the new device nodes. See Appendix H for information about using the make node utility.*

Variables in the *rfdinfo* array are as follows:

- ***device index*** indexes the SCSI device as set up in the Sun *config* file (sys/conf/RIMFIRE). For example; Sun configuration *cfØ* will have a device index of *Ø*, *cf1* will have a device index of *1*, etc.
- ***dev id*** specifies the SCSI device type. For example; *SEQ_ACC* specifies tape, *DIR_ACC* specifies hard disk, *FLOPPY* specifies floppy, and *DUMMY* specifies dummy device.

➠ *Note:* *A dummy device ID was created to work with the RF3500 utility (cs35ut). A dummy device can be opened even if a device does not exist. You can issue a command (for example, debug control or identify controller) from the restricted command menu for dummy devices. Therefore, if a device fails to open, debug can be used. To create a dummy device during installation, add an entry to the Sun config file (RIMFIRE, see page G-4), and an entry in the rfdinfo array (shown in the foregoing example). The make node utility (cs35mk) creates the device node as rrdX (rrd6 in the example), where X represents the device index from the rfdinfo array.*

- ***target id*** gives the SCSI Target ID number that is set on the drive.
- ***logical unit*** specifies the SCSI logical unit. If the device does not support logical units this field is Ø.
- ***unit opts*** indexes defines for device unit options (synchronous, not synchronous, no read capacity, etc). The defines for the various devices (such as Exabyte and Maxtor) can be found in the *cs35prm.h* file. If a define does not exist for your device, you must create one. For a detailed description of the unit options and procedures for creating defines, see Appendix C.
- ***partition number*** specifies the Device Partition number. If the device does not have partitions this field is Ø.
- ***minor device number*** appear in the last column of the *rfdinfo* array. They are for reference in creating device nodes, and they must be sequential.

Also located in the *cs35int.h* header file, is the *coninfo* array. The *coninfo* array contains an entry (Target ID) for each possible adapter configured in your system. All Ciprico adapters are assigned a default Target ID of *6*.

The following example illustrates the *coninfo* array:

```
static int coninfo[MAXBOARDS] = {
        6,
        6,
        6
};
```

SCSI devices configured in your system must have a Target ID other than *6*. If a particular SCSI device requires a Target ID of *6*, you will need to change Target IDs in the *coninfo* array to a value other than *6*.

9. Enter the following command to access the *cs35if.h* file with the *vi* editor:

```
vi cs35if.h
```

The *cs35if.h* file has define statements specifying the level of SunOS you are running. Use *vi* to search the *cs35if.h* file for the following lines:

```
/*#define SunOS3 /* define for SunOS 3.2, 3.4, and 3.5 systems */
/*#define SunOS4 /* define for SunOS 4.0 systems */
```

Remove the first two characters (/*) from the line specifying the SunOS level you are using.

For example, if you are using SunOS 3.5, the first line of the previous example should be modified to read as follows:

```
#define SunOS3 /* define for SunOS 3.2, 3.4, and 3.5 systems */
```

Enter this command to write the changes to the *cs35if.h* file and exit the editor:

```
:wq!
```

10. Enter the following command to copy the driver files from the */sys/CIPRICO/cf* directory to the */sys/sundev* directory:

```
cp cs35*.* /sys/sundev
```

11. Enter the following command to copy the driver header files from the /sys/CIPRICO/cf directory to the /usr/include/sundev directory:

```
cp cs35*.h /usr/include/sundev
```

12. Enter the following command to switch to the */sys/sun* directory:

```
cd /sys/sun
```

Enter this command to copy the *conf.c* file to *conf.c.nocf*:

```
cp conf.c conf.c.nocf
```

Access the *conf.c* file with the *vi* editor by entering the following command:

```
vi conf.c
```

Edit the *conf.c* file, adding the following references for *cf* to the include section of the file:

```
#include "cf.h"
#if NCF > Ø
extern int cfopen(), cfclose(), cfstrategy(), cfread();
extern int cfwrite(), cfdump(), cfioctl(), cfsize();
#else
#define cfopen          nodev
#define cfclose         nodev
#define cfstrategy      nodev
#define cfread          nodev
#define cfwrite         nodev
#define cfdump          nodev
#define cfioctl         nodev
#define cfsize          Ø
#endif
```

Locate the *bdevsw* structure and add the following reference to the end of the structure. Increment the Block Device Major number (represented by *XX*), and make a note of the new number for later use:

```
{ cfopen,   cfclose,   cfstrategy,   cfdump,   /*XX*/
  cfsize,   Ø},
```

Locate the *cdevsw* structure and add the following reference to the end of the structure. Increment the Character Device Major number (represented by *YY*), and make note of the new number for later use.

```
{
   cfopen,    cfclose,    cfread,    cfwrite,    /*YY*/
   cfioctl,   nodev,      nulldev,   Ø,
   seltrue,   Ø,            Ø,
},
```

Write the changes to the *conf.c* file.

13. Enter this command to copy the *swapgeneric.c* file to *swapgeneric.c.nocf*:

```
cp swapgeneric.c swapgeneric.c.nocf
```

In the *swapgeneric.c* file, locate these lines for the *xy adapter:*

```
#include "xy.h"
#if NXY > Ø
extern struct mb_driver xycdriver;
#endif
```

Add these lines after the xy adapter lines:

```
#include "cf.h"
#if NCF > Ø
extern struct mb_driver cfcdriver;
#endif
```

Locate these lines:

```
#if NXY > Ø
        {"xy", &xycdriver, makedev (XX,Ø)},
#endif
```

In the foregoing lines, XX represents the block device major number assigned in the *bdevsw* structure in the *conf.c* file.

Add these lines:

```
#if NCF > Ø
        {"cf", &cfcdriver, makedev (XX,Ø)},
#endif
```

In the foregoing lines, XX represents the block device major number assigned in the *bdevsw* structure in the *conf.c* file.

Then locate the following lines (XX represents the the block device major number, and YY represents the character device major number):

```
#if NXD > Ø
      {"xd", &xdcdriver, makedev (YY, Ø), makedev (XX, Ø)},
#endif
```

Then, add these lines following them:

```
#if NCF > Ø
      {"cf", &cfcdriver, makedev (YY, Ø), makedev (XX, Ø)},
#endif
```

14. Enter this command to write the changes to the *swapgeneric.c* file and exit the editor:

```
:wq!
```

15. Enter the following command to switch to the */sys/conf* directory:

```
cd /sys/conf
```

Enter this command to copy the *devices* file to *devices.nocf*:

```
cp devices devices.nocf
```

Enter the following command to access the *devices* file:

```
vi devices
```

➽ *NOTE: For Sun-3 systems running SunOS 3.X, this file is called "devices.sun3". In which case, you will need to enter "vi devices.sun3" to access the appropriate file.*

Edit the *devices* (or *devices.sun3*) file. Add the following reference line for the Rimfire adapter. Use the incremented number from the *bdevsw* structure (/sys/sun/conf.c) as the reference number (represented by *XX*).

```
cf XX
```

Write the changes to the *devices* (or *devices.sun3*) file.

16. Enter this command to copy the *files* file to *files.nocf*:

```
cp files files.nocf
```

Enter this command to access the *files* (or *files.sun3*) file with the *vi* editor:

```
vi files
```

➳ *NOTE: For Sun-3 systems running SunOS 3.X, this file is called "files.sun3". In which case, you will need to enter "vi files.sun3" to access the appropriate file.*

Locate the *xy* reference and add the following reference lines for the optional *cs35* device-driver:

```
sundev/cs35.c     optional cf device-driver
sundev/cs35lib.c  optional cf device-driver
```

➳ *NOTE: The hyphen must be included or the system will not read the information correctly.*

Write the changes to the *files.sun3* or *files* file.

17. While still in the */sys/conf* directory, use the following *config* command to add the new devices in the configuration:

```
config RIMFIRE
```

This creates a new subdirectory with the same name as the new configuration file. It will also create the object and header files and a makefile.

➳ *Note: If the config fails, look in /sys/RIMFIRE/makedeperrs for information.*

18. Enter the following commands to change to your new configuration directory (*RIMFIRE*) and run the *make* command:

```
cd ../RIMFIRE
make
```

This will build a new UNIX kernel, including the new adapter and drives.

When the *make* command has completed, enter the following command to copy the new *vmunix* to the root directory:

```
cp vmunix /test
```

The new *vmunix* is copied to a name other than *vmunix* for test purposes. Once it is tested and proven to work, enter the following command to overwrite the original *vmunix* with the new version:

```
mv /vmunix /vmunix.nocf
   mv /test /vmunix
```

19. Create the device nodes using the make node utility (*cs35mk*). Refer to Appendix H for a description of the utility.

➳ *NOTE: The following step is necessary only if you are installing a bootable device.*

20. Enter this command to change to the /etc directory:

```
cd /etc
```

Enter this command to copy the *fstab* file to *fstab.nocf:*

```
cp fstab fstab.nocf
```

Copying this file allows you to use the automatic update and deinstallation programs.

If the adapter you are installing is used to boot, edit the *fstab* file to change all xy references to rs.

Enter this command to write these changes to disk:

```
:wq!
```

21. Now the system can be shut down and rebooted with the new UNIX kernel.

STOP

This concludes the installation procedure for systems running SunOS 3.5 or earlier. For information on formatting and verifying drives, see Appendix B.

Driver Installation - SunOS 4.0 or later

1. Enter the following commands to create a directory path called */sys/CIPRICO/cf*:

```
mkdir /sys/CIPRICO
```

   Enter the following command to switch to the */sys/CIPRICO/cf* directory:

```
cd /sys/CIPRICO
```

2. Use the *tar* command to copy all files to the */sys/CIPRICO/cf* directory. If you are using 1/4 inch tape, enter the following *tar* command:

```
tar xvbf 126 /dev/rst8
```

   If you are using 1/2 inch tape, enter the following *tar* command:

```
tar xvbf 2Ø /dev/rmtØ
```

➳ *NOTE: For the following steps, replace SunX with the designation for the Sun system you are using (i.e., Sun3 for Sun- 3 Workstations, Sun4 for Sun-4 Workstations, Sun3x for 68030 Workstations,etc.).*

Throughout this procedure, you are asked to copy files to a file with the same name but a .nocf suffix. Doing so allows you to use the upgrade and deinstallation programs.

3. Enter these commands to change to the /sys/sunX/conf directory and make a copy of the current system configuration file. If this is a new installation, the file will be called *GENERIC*; otherwise, consult the System Administrator for the correct file name. Enter a command similar to the following to copy the configuration file to the new file name (*RIMFIRE*):

```
cd /sys/sunx/conf
cp GENERIC RIMFIRE
```

   Enter this command to copy the *RIMFIRE* file to *RIMFIRE.nocf*:

```
cp RIMFIRE RIMFIRE.nocf
```

4. Enter the following command to access the *RIMFIRE* file with the *vi* editor:

```
vi RIMFIRE
```

Search the *RIMFIRE* file for the following *config* line:

```
config vmunix swap generic
```

You can specify the root and swap devices now, or leave the setting at generic, allowing root and swap device specification at boot.

If the new Rimfire controlled disk is used as the root and swap device, edit the config line to look like this:

```
config vmunix root on cf0 swap on cf0
```

➳ *NOTE: If the driver was installed with different root and swap locations, consult your system administrator before changing root and swap locations.*

For each adapter installed, you will need to add a adapter line to the *RIMFIRE* file. The following example illustrates a typical adapter line:

```
controller cfc# at vme16d32 ? csr 0x#### priority 2 vector cfintr 0xF#
```

Variables specifying your particular system configuration (indicated in **bold** print) are as follows:

- ***cfc#*** indicates the Rimfire adapter being installed. This variable is incremented for each adapter in the system. *cfc#* entries and their respective adapter distinctions are as follows:

Adapter	*cfc#*
1st	*cfcØ*
2nd	*cfc1*
3rd	*cfc2*
4th	*cfc3*

- *Øx####* indicates the board address. During Hardware Installation procedures (see page 2-2), you were instructed to check the Board Address jumper settings for proper addressing. Enter the address currently set on the Board Address jumpers (A15-A9).
- *ØxF#* indicates the interrupt vector. The interrupt vector can be assigned any unique, single byte value. Each adapter in your system should have a unique interrupt vector value. Typically, *ØxF8* is used for the first Rimfire adapter in the system and *ØxFA* is used for the second Rimfire adapter in the system.

For example, suppose you are installing two Rimfire adapters. The Board Address jumpers (A15-A9) on the first Rimfire adapter are set for an address of *Øx5ØØØ*. The Board Address jumpers (A15-A9) on the second Rimfire adapter are set for an address of *Øx55ØØ*. The following adapter lines would be added to the *RIMFIRE* file:

```
controller cfcØ at vmel6d32 ? csr Øx5ØØØ priority 2 vector cfintr ØxFE
controller cfcl at vmel6d32 ? csr Øx55ØØ priority 2 vector cfintr ØxFA
```

5. You will also need to add references for each disk or tape drive in your system. The following lines illustrate typical tape drive and disk drive references:

```
tape cf# at cfc# drive Ø flags Ø
disk cf# at cfc# drive Ø flags Ø
```

Variables specifying your particular system configuration (indicated in **bold** print) are as follows:

- *cf#* represents the Sun logical unit value to assign to the drive (corresponding to the device index number in the *rfdinfo* array). The *cf#* reference is strictly sequential.
- *cfc#* indicates the adapter to which the drive is attached. This variable is incremented for each adapter in the system. *cfc#* entries and their respective adapter distinctions are as follows:

Adapter	*cfc#*
1st	*cfcØ*
2nd	*cfc1*
3rd	*cfc2*
4th	*cfc3*

When adding drive references, it is a good idea to proceed each reference with a comment indicating the drive type. The following example illustrates drive references for a seven-drive system with two adapters:

```
#A Hard Disk (Miniscribe)
disk cfØ at cfcØ drive Ø flags Ø
#A Hard Disk (WrenV)
disk cf1 at cfcØ drive Ø flags Ø
#A Hard Disk (Micropolis)
disk cf2 at cfcØ drive Ø flags Ø
#A Hard Disk (Micropolis)
disk cf3 at cfc1 drive Ø flags Ø
#A SCSI Tape - (Archive)
tape cf4 at cfc1 drive Ø flags Ø
#A SCSI Tape - (Wangtek)
tape cf5 at cfc1 drive Ø flags Ø
#A Dummy device
disk cf6 at cfc1 drive Ø flags Ø
```

6. Move the cursor to the xy entries. The xy entries are similar to the following lines:

```
controller xycØ at vme16d16 ? csr ØxEE4Ø priority 2 vector xyintr Øx48
controller xyc1 at vme16d16 ? csr ØxEE48 priority 2 vector xyintr Øx49
disk xyØ at xycØ drive Ø
disk xy1 at xycØ drive 1
disk xy2 at xyc1 drive Ø
disk xy3 at xyc1 drive 1
```

The disk boot emulation controller must be addressed at *ØxEE4Ø*. Other controllers cannot be present at this address. You must comment out the xy entry with a *csr ØxEE4Ø* address. Do this by adding the comment symbol (#) to the beginning of the appropriate line.

For example, if you are installing the Rimfire 3523 controller as the primary boot controller, the boot emulation address for the controller is *ØxEE4Ø*. In such a case, you must comment out the first xy entry (*xycØ*). Use your editor to add the comment symbol (#), as illustrated in this example:

```
#controller xycØ at vme16d16 ? csr ØxEE4Ø priority 2 vector xyintr Øx48
controller xyc1 at vme16d16 ? csr ØxEE48 priority 2 vector xyintr Øx49
#disk xyØ at xycØ drive Ø
#disk xy1 at xycØ drive 1
disk xy2 at xyc1 drive Ø
disk xy3 at xyc1 drive 1
```

7. Move the cursor to the xt entries. The xt entries are similar to these lines:

```
controller xtcØ at vme16d16 ? csr ØxEE6Ø priority 2 vector xtintr Øx64
controller xtc1 at vme16d16 ? csr ØxEE68 priority 2 vector xtintr Øx65
```

The tape boot emulation controller must be addressed at *ØxEE6Ø*. Other controllers cannot be at this address. You must comment out the xt entries with a *csr ØxEE6Ø* address by adding the comment symbol (#) to the beginning of the appropriate line.

For example, if you are installing the Rimfire 3523 controller as the bootable tape controller, the boot emulation address for the controller is *ØxEE6Ø*. In such a case, you must comment out the xt entry (*xtcØ*). Use your editor to add the comment symbol (#), as illustrated in this example:

```
#controller xtcØ at vme16d16 ? csr ØxEE6Ø priority 2 vector xtintr Øx64
controller xtc1 at vme16d16 ? csr ØxEE68 priority 2 vector xtintr Øx65
#tape xtØ at xtcØ drive 1
tape xt1 at xtcØ drive 1
```

8. You will also need to add device specific information for the drive(s) to the *rfdinfo* array. The *rfdinfo* array is in the *cs35int.h* file (located in the */sys/CIPRICO/cf* directory). Enter the following commands to change directories to */sys/CIPRICO/cf* and access *cs35int.h* with the *vi* editor:

```
cd /sys/CIPRICO/cf/sundev
vi cs35int.h
```

Add the necessary device-specific information. The following example illustrates the *rfdinfo* array:

```
struct device_word rfdinfo[] = {
/* device      dev        target     logical      unit    partion */
/* index       id           id         unit       opts    number  */
   {0,        DIR_ACC,       0,         0,      Miniscribe,0},  /* minor 0 */
   {0,        DIR_ACC,       0,         0,      Miniscribe,1},  /* minor 1 */
   {0,        DIR_ACC,       0,         0,      Miniscribe,2},  /* minor 2 */
   {0,        DIR_ACC,       0,         0,      Miniscribe,3},  /* minor 3 */
   {0,        DIR_ACC,       0,         0,      Miniscribe,4},  /* minor 4 */
   {0,        DIR_ACC,       0,         0,      Miniscribe,5},  /* minor 5 */
   {0,        DIR_ACC,       0,         0,      Miniscribe,6},  /* minor 6 */
   {0,        DIR_ACC,       0,         0,      Miniscribe,7},  /* minor 7 */
   {1,        DIR_ACC,       1,         0,      WrenV,     0},  /* minor 8 */
   {1,        DIR_ACC,       1,         0,      WrenV,     1},  /* minor 9 */
   {1,        DIR_ACC,       1,         0,      WrenV,     2},  /* minor 10 */
   {1,        DIR_ACC,       1,         0,      WrenV,     3},  /* minor 11 */
   {1,        DIR_ACC,       1,         0,      WrenV,     4},  /* minor 12 */
   {1,        DIR_ACC,       1,         0,      WrenV,     5},  /* minor 13 */
   {1,        DIR_ACC,       1,         0,      WrenV,     6},  /* minor 14 */
   {1,        DIR_ACC,       1,         0,      WrenV,     7},  /* minor 15 */
   {2,        DIR_ACC,       2,         0,      Micropolis,0},  /* minor 16 */
   {2,        DIR_ACC,       2,         0,      Micropolis,1},  /* minor 17 */
   {2,        DIR_ACC,       2,         0,      Micropolis,2},  /* minor 18 */
   {2,        DIR_ACC,       2,         0,      Micropolis,3},  /* minor 19 */
   {2,        DIR_ACC,       2,         0,      Micropolis,4},  /* minor 20 */
   {2,        DIR_ACC,       2,         0,      Micropolis,5},  /* minor 21 */
   {2,        DIR_ACC,       2,         0,      Micropolis,6},  /* minor 22 */
   {2,        DIR_ACC,       2,         0,      Micropolis,7},  /* minor 23 */
   {3,        DIR_ACC,       3,         0,      Micropolis,0},  /* minor 24 */
   {3,        DIR_ACC,       3,         0,      Micropolis,1},  /* minor 25 */
   {3,        DIR_ACC,       3,         0,      Micropolis,2},  /* minor 26 */
   {3,        DIR_ACC,       3,         0,      Micropolis,3},  /* minor 27 */
   {3,        DIR_ACC,       3,         0,      Micropolis,4},  /* minor 28 */
   {3,        DIR_ACC,       3,         0,      Micropolis,5},  /* minor 29 */
   {3,        DIR_ACC,       3,         0,      Micropolis,6},  /* minor 30 */
   {3,        DIR_ACC,       3,         0,      Micropolis,7},  /* minor 31 */
   {4,        SEQ_ACC,       4,         0,      Archive,0},     /* minor 32 */
   {4,        SEQ_ACC,       4,         0,      Archivenr,0},   /* minor 33 */
   {5,        SEQ_ACC,       5,         0,      WANGTEK,0},     /* minor 34 */
   {5,        SEQ_ACC,       5,         0,      WANGTEKnr,0},   /* minor 35 */
   {6,        DUMMY,     NOT_USED,  NOT_USED,  NOT_USED,0},    /* minor 36 */
};
```

➽ *Note:* *If you modify the rfdinfo array, the cs35mk.c make node utility must be recompiled before using it to make the new device nodes. See Appendix H for information about using the make node utility.*

Variables in the *rfdinfo* array are as follows:

- ***device index*** indexes the SCSI device as set up in the Sun *config* file (/sys/sunX/conf/RIMFIRE). For example, Sun configuration *cfØ* will have a device index of *Ø*, *cf1* will have a device index of *1*, etc.
- ***dev id*** specifies the SCSI device type. For example; *SEQ_ACC* specifies tape, *DIR_ACC* specifies hard disk, *FLOPPY* specifies floppy, and *DUMMY* specifies dummy device.

➳ *NOTE: A dummy device ID was created to work with the RF3500 utility (cs35ut). A dummy device can be opened even if a device does not exist. You can issue a command (for example, debug control or identify controller) from the restricted command menu for dummy devices. Therefore, if a device fails to open, debug can be used. To create a dummy device during installation, add an entry to the Sun config file (RIMFIRE, see page G-17), and an entry in the rfdinfo array (shown in the foregoing example). The make node utility (cs35mk) creates the device node as rrdX, where X represents the device index from the rfdinfo array.*

- ***target id*** gives the SCSI Target ID number that is set on the drive.
- ***logical unit*** specifies the SCSI logical unit. If the device does not support logical units this field is Ø.
- ***unit opts*** indexes defines for device unit options (synchronous, not synchronous, no read capacity, etc). The defines for the various devices (such as Exabyte and Maxtor) can be found in the *cs35prm.h* header file. If a define does not exist for your device, you must create one. For a detailed description of the unit options and procedures for creating additional defines, see Appendix C .
- ***partition number*** specifies the Device Partition number. If the device does not have partitions this field is Ø.
- ***minor device number*** appear in the last column of the *rfdinfo* array. They are for reference in creating device nodes, and they must be sequential.

Also located in the *cs35int.h* header file, is the *coninfo* array. The *coninfo* array contains an entry (Target ID) for each possible adapter configured in your system. All Ciprico adapters are assigned a default Target ID of *6*.

The following example illustrates the *coninfo* array:

```
static int coninfo[MAXBOARDS] = {
        6,
        6,
        6
};
```

SCSI devices configured in your system must have a Target ID other than *6*. If a particular SCSI device requires a Target ID of *6*, you will need to change Target IDs in the *coninfo* array to a value other than *6*.

9. Enter the following command to access the *cs35if.h* file with the *vi* editor:

```
vi cs35if.h
```

The *cs35if.h* file has define statements specifying the level of SunOS you are running. Use the *vi* editor to search for the following lines:

```
/*#define SunOS3 /* define for SunOS 3.2, 3.4, and 3.5 systems */
/*#define SunOS4 /* define for SunOS 4.0 systems */
```

Remove the first two characters (/*) from the line specifying the SunOS level you are using. For example, if you are using SunOS 4.0, the second line of the above example should be modified to read as follows:

```
#define SunOS4 /* define for SunOS 4.0 systems */
```

Enter this command to write the changes to the cs35if.h file and exit the editor:

```
:wq!
```

10. Enter the following command to copy the driver files from the */sys/CIPRICO/cf* directory to the */sys/sundev* directory:

```
cp cs35*.* /sys/sundev
```

11. Enter the following command to copy the driver header files from the /sys/CIPRICO/cf directory to the /usr/include/sundev directory:

```
cp cs35*.h /usr/include/sundev
```

12. Enter the following command to change directories to */sys/sun*:

```
cd /sys/sun
```

Enter this command to copy the *conf.c* file to *conf.c.nocf*:

```
cp conf.c conf.c.nocf
```

Edit the *conf.c* file, adding the following references for *cf* to the include section of the file:

```
#include "cf.h"
#if NCF > Ø
extern int cfopen(), cfclose(), cfstrategy(), cfread();
extern int cfwrite(), cfdump(), cfioctl(), cfsize();
#else
#define cfopen          nodev
#define cfclose         nodev
#define cfstrategy      nodev
#define cfread          nodev
#define cfwrite         nodev
#define cfdump          nodev
#define cfioctl         nodev
#define cfsize           Ø
#endif
```

Locate the *bdevsw* structure and add the following reference to the end of the structure. Increment the Block Device Major number (represented by *XX*), and make a note of the new number for later use.

```
{  cfopen,    cfclose,    cfstrategy,    cfdump,    /*XX*/
   cfsize,    Ø },
```

Locate the *cdevsw* structure and add the following reference to the end of the structure. Increment the Character Device Major number (represented by *YY*), and make note of the new number for later use.

```
{
   cfopen,     cfclose,     cfread,     cfwrite,     /*YY*/
   cfioctl,    nulldev,     seltrue,    Ø,
   Ø,
},
```

Write the changes to the *conf.c* file.

13. Enter this command to copy the *swapgeneric.c* file to *swapgeneric.c.nocf*:

```
cp swapgeneric.c swapgeneric.c.nocf
```

In the *swapgeneric.c* file, locate these lines for the *xy adapter:*

```
#include "xy.h"
#if NXY > Ø
extern struct mb_driver xycdriver;
#endif
```

Add these lines after the xy adapter lines:

```
#include "cf.h"
#if NCF > Ø
extern struct mb_driver cfcdriver;
#endif
```

Locate these lines:

```
#if NXY > Ø
        {"xy", &xycdriver, makedev (XX,Ø)},
#endif
```

In the foregoing lines, XX represents the block device major number assigned in the *bdevsw* structure in the *conf.c* file.

Add these lines:

```
#if NCF > Ø
        {"cf", &cfcdriver, makedev (XX,Ø)},
#endif
```

In the foregoing lines, XX represents the block device major number assigned in the *bdevsw* structure in the *conf.c* file.

Then locate the following lines (XX represents the the block device major number, and YY represents the character device major number):

```
#if NXD > Ø
      {"xd", &xdcdriver, makedev (YY, Ø), makedev (XX, Ø)},
#endif
```

Then, add these lines following them:

```
#if NCF > Ø
      {"cf", &cfcdriver, makedev (YY, Ø), makedev (XX, Ø)},
#endif
```

14. Enter this command to write the changes to the *swapgeneric.c* file and exit the editor:

```
:wq!
```

15. Enter the following command to change directories to */sys/SunX/conf*:

```
cd /sys/SunX/conf
```

Enter this command to copy the *devices* file to *devices.nocf*:

```
cp devices devices.nocf
```

Enter the following command to access the *devices* file with the *vi* editor:

```
vi devices
```

Edit the *devices* file. Add the following reference line for the Rimfire adapter. Use the incremented number from the *bdevsw* structure (/sys/sun/conf.c) as the reference number (represented by *XX*).

```
cf XX
```

Write the changes to the *devices* file.

16. Enter this command to copy the *files* file to *files.nocf*:

```
cp files files.nocf
```

Enter the following command to access the *files* file with the *vi* editor:

```
vi files
```

Locate the *xy* reference and add the following reference lines for the optional *cs35* device driver. The hyphen must be included or the system will not read the information correctly.

```
sundev/cs35.c     optional cf device-driver
sundev/cs35lib.c  optional cf device-driver
```

Write the changes to the *files* file.

17. While still in the */sys/sunX/conf* directory, use the following *config* command to add the new devices in the configuration:

```
config RIMFIRE
```

This creates a new subdirectory with the same name as the new configuration file. It also creates the object and header files and a makefile.

➳ *Note: If the config fails, look in /sys/RIMFIRE/makedeperrs for information.*

18. Enter the following commands to change to your new configuration directory (*RIMFIRE*) and run the *make* command:

```
cd ../RIMFIRE
make
```

This will build a new UNIX kernel, including the new adapter and drives.

When the *make* command has completed, enter the following command to copy the new *vmunix* to the root directory:

```
cp vmunix /test
```

The new *vmunix* is copied to a name other than *vmunix* for test purposes. Once it is tested and proven to work, enter the following command to overwrite the original *vmunix* with the new version:

```
mv /vmunix /vmunix.nocf
   mv /test /vmunix
```

19. Create the device nodes using the make node utility (*cs35mk*). Refer to Appendix H for a description of it.

➳ *NOTE: The following step is necessary only if you are installing a bootable device.*

20. Enter this command to change to the /etc directory:

```
cd /etc
```

Enter this command to copy the *fstab* file to *fstab.nocf:*

```
cp fstab fstab.nocf
```

Copying this file allows you to use the automatic update and deinstallation programs.

If the adapter you are installing is used to boot, edit the *fstab* file to change all xy references to rs.

Enter this command to write these changes to disk:

```
:wq!
```

21. Now, the system can be shut down and rebooted with the new UNIX kernel.

STOP

This concludes the installation procedure for systems running SunOS 4.0 or later. For information on formatting and verifying drives, see Appendix B.

Appendix H - Using the Make Node Utility

Follow the steps in this procedure to use the make node utility, *cs35mk.c*.

1. Enter the following command to change to the /sys/CIPRICO/cf directory:

```
cd /sys/CIPRICO/cf/cs35makedev
```

2. Compile the make node utility (*cs35mk.c*) by entering the following command:

```
make
```

The Make Node utility (*cs35mk*) is used to make the device nodes. This utility reads device specific information from the *rfdinfo* array and creates the device nodes. You must know the current block major (*XX*) and character major (*YY*) device numbers to run correctly the *cs35mk* utility. (These are the entries in the bdevsw and cdevsw tables in the *conf.c* file located in the /sys/sun directory.

Table H-1 illustrates the *cs35mk* options and respective commands for creating nodes.

Table H-1 cs35mk Commands

Option	*cs35mk* Command
Individually recreate nodes for the default path (*/dev*)	*cs35mk*
Recreate all nodes for the default path (*/dev*)	*cs35mk y*
Individually recreate nodes for a specified path (indicated by *[path]*)	*cs35mk n [path]*
Recreate all nodes for a specified path (indicated by *[path]*)	*cs35mk y [path]*

The command you enter to run the *cs35mk* utility will vary with the number of nodes recreated and the path name for creating the node. The standard command to run the *cs35mk* utility is as follows:

```
cs35mk
```

If a node exists with the same name as the node you are creating, a request appears asking if you want to delete the existing node and create a new one. Entering *y* creates the new node.

If you would rather recreate all nodes, enter a *y* as the first argument of the command line, as illustrated in the following example:

```
cs35mk y
```

The *cs35* utility uses */dev* as the default path name for making nodes. If a different path is desired, enter that path as the second argument of the command line. The specified path must be for an existing directory. Directories can be created with the *mkdir* command (i.e., *mkdir /dev/35*) For example, the following command would be entered to recreate all nodes for */dev/35*:

```
cs35mk y /dev/35
```

While the following command would be entered to recreate nodes for */dev/35* individually:

```
cs35mk n /dev/35
```

When you enter the *cs35mk* command, the following message is displayed:

```
You will need to know the major numbers for both block and
character devices.  If you do not have the information; you
should stop this program and obtain the required information.
Do you wish to continue?
```

If you enter *n*, the program will exit to the system prompt.

If you enter *y*, you are prompted for the Block Device Major number and Character Device Major number, as illustrated in the following lines:

```
What is your block device major number?
What is your character device major number?
```

3. Enter the current Block Device and Character Device major numbers.

   If you elected to recreate all nodes (by entering *cs35mk y* or *cs35mk y [path]*), device nodes are created automatically.

   If you elected to individually recreate device nodes (by entering *cs35mk* or *cs35mk n [path]*), lines similar to the following are displayed for each existing device node:

   ```
   /dev/rrsØa already exists,
   Do you want to remake it?  y/n or q
   ```

   If you wish to delete the node and create a node with the same name, enter *y*. The program will recreate the device node and continue making nodes.

   If you do not wish to remake the node, enter *n*. The program will continue making nodes.

4. To quit making device nodes, enter *q*. The program exits to the system prompt.

   If an error occurs while creating a node, a message (similar to the following example) appears:

   ```
   Can't make node for /dev/rrsØa
   ```

   You also have the option of manually creating the device nodes using the *mknod* command. Appendix D gives procedures for manually making the device nodes.

5. After making the nodes, shut down the system and reboot with the new UNIX kernel.

Appendix I - Adding a Device Manufacturer to the Installation Script

If the Device Type menu does not include a device manufacturer you need, you can add it to the menu before installation. You need a manual for the device you want to add. Follow the steps in this procedure to add a manufacturer to the installation script. These examples are for the Maxtor disk drive in asynchronous mode.

1. Enter this command to change to the /sys/CIPRICO/cf/sundev directory:

   ```
   cd /sys/CIPRICO/cf/sundev
   ```

   Enter this command to edit the *cs35prm.h* file:

   ```
   vi cs35prm.h
   ```

2. Create a define statement for the device you want to add, using Appendix C of this manual and the appropriate device manual. In Appendix C, ignore the example of the define used in the rfdinfo array; it only applies to manual installation of the device driver.

3. Each device type in the Device Type Menu has a file that contains information the install script needs to generate an entry in the *rfdinfo* array. Enter this command to change directories to /sys/CIPRICO/cf/install/proto:

   ```
   cd /sys/CIPRICO/cf/install/proto
   ```

   The device type files all have a file name like this: *cf.dev.name*. The file you create must use this format for the install script to find it. *cf.dev* is a constant prefix, and *.name* can be anything, but it should be some sort of abbreviation of the manufacturers name.

   For example, the Maxtor Asynchronous disk drive file name is *cf.dev.maxtor*, and the Maxtor synchronous disk drive file name is *cf.dev.maxtors*. Enter the following command to create a file for your device; *NAME* is an abbreviation of the device manufactures name:

   ```
   vi cf.dev.NAME
   ```

The contents of this file are:

```
Device description as to show up in menu
SCSI device type (DIR_ACC for disk ... )
Unit options define as created above
```

The device description should contain the device manufacturer, the device type (disk, tape), and anything else you may want to appear in the Device Type menu.

The SCSI device types are defined in the *cs35prm.h* header file. The device type entry in the file must match one of these types:

Device Type	Description
DIR_ACC	Disk
SEQ_ACC	Tape
PRN	Printer
PROC	Processor
WORM	Write once/read multiple - optical disk
RDONLYDIR_ACC	Read only direct access device - optical disk
FLOPPY	Floppy device
DUMMY	Dummy device

The unit options define is the define you created in step 2. This field should contain a define for each configuration needed to operate the device. For example, most tape files contain a define for the standard device, and one for the no-rewind device because both configurations are needed for the device.

Following are examples of device type files for the Maxtor Asyncronous disk drive, the Maxtor Syncronous disk drive, and the Wangtek tape drive:

Maxtor asyncronous disk drive

```
Maxtor Asyncronous Disk Drive
DIR_ACC
Maxtor
```

Maxtor syncronous disk drive

```
Maxtor Syncronous Disk Drive
DIR_ACC
MaxtorS
```

Wangtek tape drive

```
Wangtek QIC Tape Drive
SEQ_ACC
WANGTEK WANGTEKnr
```

Index

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

```
#ifndef lint
#static char Sccsrid[] = "@(#)README  (75222005)  Copyright 1987, 1989 Ciprico,
#endif
```

The following information applies to revision 2.0 or higher releases of the RimFire 3500 Unix Sun OS driver. A number of enhancements and alterations have been made to the RF3500 driver with the release of version 2.0:

1) Ciprico has a new device driver nameing system which changes the driver file names from rfsd to cs35. The "c" represents Ciprico, the "s" is the system code, which is Sun in this case, the "3" is the bus designator as used in the products themselves, which is vme, and the "5" is for controller type which is SCSI. The driver prefix is changing from rfsd to cf. The "c" represents Ciprico, and the "f" is the substitute for 5 for the 3500. Because of the way the Sun config file is parsed, numbers could not be used in the driver prefix.

   Ciprico Driver Prefix Letter Substitutes:
   ```
       Ciprico TM3000 = 0 = a
                        1 = b
       Ciprico RF3200 = 2 = c
                        3 = d
       Ciprico RF3400 = 4 = e
       Ciprico RF3500 = 5 = f
   ```

2) The RF3500 driver now supports multiple RF3500 adapters in the same system.

3) The Minor device number system used to identify the various SCSI devices has been greatly simplified and is contained in one master array. (rfdinfo located in the header file cs35int.h)

4) The driver no longer uses the drive and flag fields of the Sun config file. This information can now be found in the rfdinfo array located in the header file cs35int.h.

5) A new source code file for standard Ciprico and SCSI functions has been added. (This is known as cs35lib.c)

6) The header file cs35reg.h has been renamed to cs35if.h and has been expanded to contain additional code structures.

Because this driver has been renamed, it is strongly recommended by Ciprico to remove the old driver, when updating to 75222005. When reinstalling the new driver, use the latest installation manual (21017200). Below is a list of steps to take to remove the old driver from the kernel. If you do not have a copy of the latest manual, call Ciprico Customer Support.

INSTRUCTIONS TO REMOVE THE RF3500 DRIVER FROM KERNEL:

1. Change to the "/sys/conf/" directory and edit the configuration file that contains the Rimfire 3500 controller and device references. This file will be refered to as "RIMFIRE" throughout these instructions. Remove the reference lines for the controller and any drives that have been configured in. Examples are as shown below:


```
controller rfsdc0 at vme16d32 ? csr 0x5000 priority 2 vector rfsdintr 0xF8
controller rfsdc1 at vme16d32 ? csr 0x5500 priority 2 vector rfsdintr 0xFA
#A SCSI Tape (Exabyte) Target ID=5
tape rfsd0 at rfsdc0 drive 40 flags 0x211
#A Hard Disk (Micropolis) Target ID = 0
disk rfsd1 at rfsdc0 drive 0 flags 0
#A Hard Disk (Maxtor) Target ID = 1
disk rfsd2 at rfsdc0 drive 8 flags 0x180
#A SCSI Tape (Wangtek) Target ID = 2
tape rfsd3 at rfsdc1 drive 16 flags 0x11e
```

2. Edit the file "files.sun3" or "files" which is also in the "/sys/conf" directory. Locate and remove the "rfsd" reference.

   ```
   sundev/rfsd.c     optional rfsd device-driver
   ```

   Write the changes to the "files.sun3" or "files" file.

3. Change directories to "/sys/sun" and edit the "conf.c" file. Remove the following references to "rfsd" in the include section of the file:

   ```
        #include "rfsd.h"
        #if NRFSD > 0
        extern int rfsdopen(), rfsdclose(), rfsdstrategy(), rfsdread();
        extern int rfsdwrite(), rfsdioctl(), rfsdsize();
        #else
        #define rfsdopen        nodev
        #define rfsdclose       nodev
        #define rfsdstrategy    nodev
        #define rfsdread        nodev
        #define rfsdwrite       nodev
        #define rfsdioctl       nodev
        #define rfsdsize          0
        #endif
   ```

   Locate the "bdevsw" structure and remove the following reference near the end of the structure.

   ```
        {rfsdopen,    rfsdclose,    rfsdstrategy,    nulldev,    /*XX*/
        rfsdsize,    0},
   ```

   Locate the cdevsw structure and remove the following reference near the end of the structure.

   ```
        {
           rfsdopen,     rfsdclose,     rfsdread,     rfsdwrite,     /*YY*/
           rfsdioctl,    nodev,         nulldev,      0,
           seltrue,      0,             0,
        },
   ```

   Write the changes to the "conf.c" file.

4. Change directories to "/sys/conf" and edit the "devices.sun3" or "devices" file. Remove the following reference line for the Rimfire controller.


   ```
   rfsd XX
   ```

   Write the changes to the "devices.sun3" or "devices" file.

5. If you have the old driver on tape, change directories to /sys/sundev and remove the driver from this directory with the following commands:

   ```
   cd /sys/sundev
   rm rfsd*
   ```



```
/*****************************************************************************
******************************************************************************
****                                                                      ****
****                Copyright (C) Ciprico Incorporated 1987.              ****
****                                                                      ****
****                        Ciprico Incorporated                          ****
****                          2955 Xenium Lane                            ****
****                         Plymouth, MN 55441                           ****
****                           (612) 559-2034                             ****
****                                                                      ****
****    Module Name:    cs35.c                                            ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus.    ****
****    Module Rev:     $Revision$                              ****
****    Date:           $Date$              ****
****                                                                      ****
****    Subroutines:    cfattach, cfopen, cfclose, cfstrategy,            ****
****                    cfread, cfwrite, cfioctl, cfprint,                ****
****                    sglcmd, rfsdcmd                                   ****
****                                                                      ****
****    Description:                                                      ****
****            This driver handles UNIX I/O requests for the Ciprico     ****
****            Rimfire 3500.                                             ****
****                                                                      ****
******************************************************************************
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                           *
*                              Revision History                             *
*                                                                           *
*****************************************************************************

    Revision    Date            Author
        Description of Change
-----------------------------------------------------------------------------

    1.0         07/14/86        Umesh Gupta
        Initial Release.

    1.1         02/02/87        D. A. Dickey
        Initial B Release.
    1.1         08/20/87        D. A. Dickey
        Initial A Release.

    1.2         ??/??/??        ???? ????

    1.3         02/88 John Lind
        a. Add Sun 4 compatibility (include files and VME address modifier)
           [rfsd.c, rfsdparam.c]
        b. Fix retry reset for SEQ_ACC devices [rfsd.c]
        c. Add the EXABYTE vendor unique parameter to handle Busy Status
                   Jody Martin
        d. Added the following changes made by PURDUE.
                1. Add RFIOCGBLKSIZE and RFIOCSBLKSIZE to get/set block size for
                   exabyte drive.
                2. Add the FIXEDBLK flag in the configuration for the exabyte.
                3. Add global variable "rfsd_pr_sense = 0", more debug.
                4. Added a uonEOF flag to keep track of a filemark status,
                   when actual data on tape is smaller than requested before
                   the filemark, the FM was getting lost.
                5. The driver was changed to not do "Test unit ready" command
                   during the slave routine if device is SEQ_ACC.
                6. Change to not skip to filemark on no-rew close after read cmd
                7. Changed code for the Exabyte LEOT error.  LEOT will now be
                   invisable.
                8. Made corrections to the Berkley "mt" ioctls so the mt command
                03/01/88        K. J. Hopps
        f. Mods to allow 1k block sizes (especially for WORMs)
                03/29/88        Jody Martin
        g. If the device is a "tape", device initialization will not be done
           until the "open" routine.
        i. There was a problem with the Restore command and some Sequential
           Access devices.  The Residual count (amount of bytes not transfered)
           was set incorrectly if reading data in block mode, when end of file
           was crossed and the drive responded to a request sense with the Valid
           bit of the class byte true.  The Restore command would fail, not
           reqesting the next tape, but displaying an error.
        k. A "Set General Board Options" command will now be the 1st command
           issued to the controller.
        n. When issueing a Mode Select cmd, the Vme xfer count was set to 0xff
           and the Parameter count was 12.  This caused the command to hang if
```



```
           the exact burst jumper was in.  The vme xfer count was changed to 12.

    1.4
                04/29/88        Jody Martin
        a. A problem existed in the way B_WANTED flag was cleared.
        b. turned on disconect/reselect feature in the Unit options command.
        c. Added support for new Request Extended Sense bytes.  See the flag
           definitions in the rfsdparam.h file.
                05/24/88        Dick Taylor
        d. Corrected status block handling in the rfsdintr routine for extended
           sense bytes.  Also added a number of small changes to print the
           target ID in a status block, observe a programmable delay after
           reset, and show that a retry corrected an error.
        e. Changes were made to support the new "Ignor SCSI soft errors"
           feature in the Unit Options command.
           [rfsd.c] [rfsdparam.h]
                06/15/88        Jody Martin
        f.  A problem existed in the way iomem was de-allocated in the slave
            routine.
        h.  A "b" error was occurring with the General Options command in the
            "slave" routine because the Extended Parameter block was not being
            zeroed out correctly.
        k.  Added the changes so the driver would run on Sun OS3 or OS4.

        $Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35.c_v  $

      Rev 2.0   09 May 1989 11:40:40   JMartin
   Initial revision.

    2.0         04/06/89 Jody Martin

        1. Ciprico has a new device driver nameing system which changes the
           driver prefix to cs35 from rfsd.  The "c" represents Ciprico, the
           "s" is the system code, which is Sun in this case, the "3" is the
           bus designator as used in the products themselves, which is vme,
           and the "5" is for controller type which is SCSI.
        2. The RF3500 driver now supports multiple RF3500 adapters in the same
           system.
        3. The Minor device number system used to identify the various SCSI
           devices has been greatly simplified and is contained in one master
           array. (rfdinfo located in the header file cs35int.h)
        4. The driver no longer uses the drive and flag fields of the Sun
           config file.  This information can now be found in the rfdinfo
           array located in the header file cs35int.h.
        5. A new source code file for standard Ciprico and SCSI functions has
           been added. (This is known as cs35lib.c)
        6. The header file cs35reg.h has been renamed to cs35if.h and has been
           expanded to contain additional code structures.
        7. During cmds other than read/write cmds, the driver was sleeping on
           iomem when waiting for command complete.  When running multiple
           devices at the same time, this was a problem because they would
           sometimes have the same address for iomem, thus commands were
           being woken up before they were actully complete.

$Log$
```



```
    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523 as follows:

           - Made changes to support standalone utility.
           - Made many changes to the cs35ut utility, and the driver ioctl
             calls.
           - Made changes to support the new install script.
           - The format disk command now supports including the defect list
             during the disk format.
           - Changed the command codes to match as close as possible to the
             Rimfire 32XX rfutil.
           - Added support for selecting a dummy device to change debug
             value through the utility if you can't open another disk.
           - Added various features to existing commands.

*****************************************************************************/
```



```
/*
        The major points of interest within this file are ordered as follows:
                Include the include files
                Subroutine definitions
                Variables UNIX needs
                RF 3500 Driver internal structs
                cfattach      - Initialization routine
                cfopen        - Open routine
                cfclose       - Close routine
                cfstrategy    - Strategy routine
                cfintr        - Interrupt service routine
                cfread        - Raw read routine
                cfwrite       - Raw write routine
                cfioctl       - Ioctl interface routine
                cfprint       - Print out a message
                sglcmd        - Send a type 0 command to the controller
                rfsdcmd       - Send a type 1 command to the controller

*****************************************************************************

        The subroutines interact as follows.  The interaction shown is only for
        those subroutines within this driver, unix subroutines are not shown.

        ROUTINE or SYSTEM               CALLS

        I/O system              ==>     cfattach, cfopen, cfclose,
                                        cfstrategy, cfread, cfwrite,
                                        cfioctl

        Interrupt Mechanism     ==>     cfintr

        cfattach                ==>     sglcmd
        cfopen                  ==>     rfsdcmd
        cfclose                 ==>     rfsdcmd
        cfstrategy              ==>     (nothing)
        cfintr                  ==>     (nothing)
        cfread                  ==>     cfstrategy (via physio)
        cfwrite                 ==>     cfstrategy (via physio)
        cfioctl                 ==>     rfsdcmd
        cfprint                 ==>     (nothing)
        sglcmd                  ==>     (nothing)
        rfsdcmd                 ==>     (nothing)

*****************************************************************************


*****************************************************************************
*/
```

```
#ifndef lint
static  char Sccsid[] = "@(#)cs35.c   (75222006)  Copyright 1987, 1989 Ciprico,
#endif

/* The include file rfsd.h is created during the configuration process on Sun
 * machines.  Its name itself is dependent on the configuration process.
 * Basically, what is needed in the include file are two defines:
 *      NC5     - Defines how many drives are hooked up to RF3500's
 *      NC5C    - Defines how many RF 3500's are plugged into the system
 */
#include "cf.h"


/* Don't bother compiling if there aren't any drives. */
#if NCF > 0

/* Include whatever system include files are needed here. */
#include <sys/param.h>
#include <sys/dir.h>
#include <sys/user.h>
#include <sys/buf.h>
#include <sys/systm.h>
#include <sys/kernel.h>
#include <sys/map.h>
#include <sys/ioctl.h>
#include <sys/file.h>
#include <sys/dk.h>
#include <sys/mtio.h>
#include <sys/uio.h>

#include "../sun/dkio.h"
#include "../sun/dklabel.h"
#include "../machine/psl.h"
#include "../sundev/mbvar.h"

#include "../sundev/cs35if.h"
#include "../sundev/cs35io.h"
#include "../sundev/cs35err.h"
#include "../sundev/cs35prm.h"
#include "../sundev/cs35flp.h"
#include "../sundev/cs35int.h"

#define b_saddr         b_mbinfo

/* Get a Direct Virtual Memory Address (DVMA) from a virtual kernel address */
#define RF_ADDR(x) ((int)((int)(x) & 0xFFFFF))  /* DVMA address (20 bits) */

/* For those routines in the driver which are visible to the outside world,
 * define what they return.
 */

int cfopen(), cfclose(), cfintr();
int cfioctl(), cfread(), cfwrite(), cfdump();
void cfstrategy();
daddr_t cfsize();
```



```
static int cfprobe(), cfattach(), cfslave();
static caddr_t safealloc();
static void safefree();

/* Define the structures and storage needed for configuration and runtime */
struct mb_ctlr *cfcontroller_info[NCFC]; /* per controller */
static struct mb_device *cfdrive_info[NCF];     /* per drive */

/* the driver structure */
struct mb_driver cfcdriver = {
        cfprobe,                /* the probe routine */
        cfslave,                /* the slave routine */
        cfattach,               /* the attach routine */
        0,                        /* there is no mdr_go routine */
        0,                        /* there is no mdr_done routine */
        cfintr,                 /* the intr routine */
        0,                        /* no memory used */
        "cf",                   /* the device name */
        cfdrive_info,           /* info per drive */
        "cfc",                  /* the controller name */
        cfcontroller_info,      /* info per controller */
        MDR_BIODMA,               /* we do DMA transfers */
        0                         /* interrupt routine linked list */
};

/* General Options Flag.  Putting the options into a variable allows a user */
/* to change the options with adb (so they don't have to recompile). */
/* NOTE: Suns DO NOT support the BMT option. */
static int      rfsdoptions     = DIS;

/***********************************************
 * Debug macro - execute code if debug level is set
 ***********************************************/

#define DEBUG(level, code)      if (rfsddbg & level) {code};

/*  Set the debug flag to turn off debugging or turn on debugging */
static dword rfsddbg = 0x0;

        /* fixed/var default mode - should be per drive */
static dword rfsd_ex_fixed = 0; /* set to run in fixed blocksize */
static dword rfsd_pr_sense = 0; /* set to 0 to shutoff, 1 for sense,2 verbose */
static dword rfsd_pr_check = 0; /* set to 0 to shutoff, 1 for sense,2 verbose */

/*  NOTE:  The following flags are Exabyte options only.  The 1st five flags  *
 *  control vender unique parameters for the mode select command for Exabyte  *
 *  only.  These flags will only be monitored for devices with the Exabyte    *
 *  flag set in unit options field of the rfdinfo array.  The next four flags *
 *  allow selection of firmware revision, writing short filemarks, and contol *
 *  of what gets printed to the screen.                                       */

static dword rfsd_ex_optional1 = 0xe;   /* patchable mode sel 1st vendor byte */
                                        /* busy stat, evn byte dis, _______   */
static dword rfsd_ex_cart = 1;          /* 1 = P5 (European) (pad count ???)  */
static dword rfsd_ex_motion = 0x80; /*motion thresh, (1-0xf7) default is 0x80 */
static dword rfsd_ex_recon = 0xA0; /* reconnect thresh, (1-0xf7) default 0xA0 */
```



```
static dword rfsd_ex_gap = 0x0;         /* gap thresh, (0-0x7) default is 0x7 */

static dword rfsd_ex_MX4_23 = 1;    /* set if all drives >= MX4$23 firmware    */
                                    /* allow mode select of motion, recon, gap */
static dword rfsd_ex_write_short_fm = 0;            /*use Exabyte short filemarks */
static dword rfsd_pr_errlog =  0;   /* 01 - uprintf, 02 - printf, 0x8 debug */
static dword rfsd_pr_leot = 0;  /* set to 0 to shutoff, 1 for sense,2 verbose */
                                /* dont set this if Exabyte has MX 4$23 fmwr  */

/* The devtype variable will be set equal to md_dk of mb_device structure in
 * the slave routine.  This will be used in the attach routine to determine if
 * the device is a tape or a disk.  I had problems trying to reference this
 * md_dk field of the structure once in the attach routine.  ??? 1.3   */
short devtype[NCF];

/* Local buffer header used for executing all the commands
 * except system read/write request
 */
struct buf rfsdbuf[NCF];

struct buf rrfsdbuf[NCF];       /* Buffers for physical reads & writes. */

#define CREGBUG         /* comment out this line if you don't have the register

#ifdef CREGBUG
# define REGISTER
#else
# define REGISTER register
#endif

/* iomem_count keeps track of how many times the driver has iomem allocated at
 * the present time.  At times, when multiple devices are running at once,
 * errors would occur when calling rmalloc for iomem.  This counter will be
 * incremented when a succesful rmaolloc occurs for iomem, and decremented when
 * the rmfree is called.  If no memory is available when calling rmalloc, and
 * iomem_count is not zero, the driver will sleep, waiting for memory to be
 * free.  If iomem_count is zero, an actual memory shortage has occured that
 * wasn't caused by our driver, and the driver will return an error.
 * "iomem_wanted" is set to TRUE if NULL is returned from rmalloc when trying
 * to aquire memory from iomem, and sleep() is called.  IF iomem_wanted is TRUE
 * when iomem is freed, it is set to FALSE, and wakeup() is called.
 */

static int iomem_count;
static byte iomem_wanted;

/* how long (looping) until controller reset should complete */
/* This is MUCH too cpu dependent to really be considered a second. */
/* Adjust this as needed. */
#define SECONDS         600000          /* approx. one second in loop */

#ifdef STANDALONE
#define MAXWAIT         (2700*SECONDS)  /* wait up to45 min for format cmd */
#else
#define MAXWAIT         (15*SECONDS)    /* wait few seconds */
#endif
```



```
/* The Rimfire 3500 uses a circular buffer structure for passing
 * parameter blocks to the controller and returning status blocks
 * from the controller.  This structure must be physically contiguous.
 */
static CMDLIST  *cmdq[NCFC];            /* one per controller */

/* This is the BSY bit of the Rimfire 3500's Status Port as the controller
 * pictures it.  It should be set to zero on a controller reset and toggle
 * for each type zero channel attention.
 */
static int      bsybit[NCFC];

/* This is the controller type of each controller.  The value is read from
 * the status register of the board after a reset.  See rfprobe().
 */
static int      rfsdctype[NCFC];

/* This is the blksiz of the device that gets filled in during the attach
 * routine after a mode sense command is issued.  The open routine uses this
 * value when it reads the disk label */
static int      blksiz[NCF];

/* To keep track of all the parameter blocks in the RF3500, the ID field
 * of the standard parameter block is actually a pointer to the associated
 * UNIX buffer.
 * Thus, when we receive a status block from the controller, we can
 * use its identifier to determine which UNIX buffer to free.
 * We also need to keep the return value of mbsetup (mbinfo) so that
 * we can release the entries in the kernel memory map via mbrelse.
 * Since we don't use the available list pointers in the buf structure
 * we will abuse them by saving mbinfo as a pointer to struct buf.
 */
#define b_mbinfo av_forw        /* from struct buf */
#define b_md_hd  av_back        /* from struct buf */

/* the geometry for each of the drives */
static struct dk_geom geometry[NCF];

/* the partition information for each of the drives */
static struct dk_map partitions[NCF][NDKMAP];

/* Bit arrays that indicate the state of the devices */
static byte     ureserved[NCF];  /* A bit for each logical unit. */
static byte     uwritten[NCF];
static byte     uonfmk[NCF];
static byte     uonEOF[NCF]; /* set if drive hit FM,but user hadn't seen it */
static byte     ugeneral[NCF];
static word     uopen[64 * NCFC]; /* uopen is an array of 16 bit words with */
/* each word representing a device on the scsi bus, and each bit of the word  */
/* representing a partition on the device. Sun presently supports 8 partions  */
/* on a disk device, but will be supporting 16 partitions in the future.  The */
/* array size is 64 (number of possible target id's multiplied by the number  */
/* of possible logical units attached to that target id) multiplied by the    */
/* number of controllers configured in the system. */
```

```
/* The following defines manipulate the state arrays above.  This is done
 * as follows:
 *   For all arrays, for each physical unit (up to NCF of them), use
 *   bit 0 for logical unit zero, bit 1 for logical unit one, etc.
 *   One exception to this is for disks (including floppies) uopen array.  In
 *   this case, the lunit is actually (lunit + 1); with lunit == 0 specifying
 *   that at least one partition is still open on the disk.  The following
 *   exception for floppies still applies.
 *   The physical unit comes from the minor device number for all device types.
 *      SEQ_ACC & FLOPPIES:
 *              Use the logical unit as specified in the device_target array
 *      OTHERS:
 *              Use the logical unit as specified in the minor device number
 */
#define uflagon(ary, punt, miner)       if(rfdinfo[miner].dev_id == SEQ_ACC || \
                                rfdinfo[miner].tag_id == 0xFE) \
                                ary[punt] |= (1 << rfdinfo[miner].log_unit); \
                                else \
                                ary[punt] |= (1 << rfdinfo[miner].partition)
#define uflagoff(ary, punt, miner)      if(rfdinfo[miner].dev_id == SEQ_ACC || \
                                rfdinfo[miner].tag_id == 0xFE) \
                                ary[punt] &= ~(1 << rfdinfo[miner].log_unit); \
                                else \
                                ary[punt] &= ~(1 << rfdinfo[miner].partition)
#define uflagtst(ary, punt, miner)      ((rfdinfo[miner].dev_id == SEQ_ACC || \
                                rfdinfo[miner].tag_id == 0xFE) ? \
                                ary[punt] & (1 << rfdinfo[miner].log_unit) : \
                                ary[punt] & (1 << rfdinfo[miner].partition))

#define uopenon(openmsk, miner, index) \
                        openmsk[index] |= (1 << rfdinfo[miner].partition)
#define uopenoff(openmsk, miner, index) \
                        openmsk[index] &= ~(1 << rfdinfo[miner].partition)
#define uopentst(openmsk, miner, index) \
                        openmsk[index] & (1 << rfdinfo[miner].partition)


/* What are the record sizes on the device...and how many are there. */
rec_info record_info[NCF];

/* Set this flag when command list full and some process
 * is waiting for a parameter block
 */
dword   pb_wanted[NCFC];

/* 'Identify' status */
RETID   idstat[NCFC];

dword   kb_xfer[NCF];           /* Kbytes xferred */
byte    lastop[NCF];            /* last operation (for errlog) */
#define UNDEF   0               /* last operation undefined */
#define READ    1               /* last operation was a READ */
#define WRITE   2               /* last operation was a WRITE */

/* Rename some of the entries in flags field of buffer header */
```



```
#define b_bn    b_resid         /* block number */

/* Addressing macros */
#define realaddr(n)     (RF_ADDR((dword) (n)))

static int sglcmd();

/* Some defines that should not change */
#define RF3500_ID       0xFF    /* Target ID for general board commands */

/* Mode select user selected values - these will be set to the default
 * values in the cfattach routine.  They will can be changed thru the
 * mode select command in the utility, and won't be reset to the
 * default values again until the next time the system is rebooted,
 * or the user chooses the default values. */
mode_sel mdsel_val[NCF];

/* Mode select tape default values */
mode_sel mdsel_tp_def = {
        0,      /* reserved */
        0,      /* medium type */
        0x10,   /* byte2 */
        8,      /* block descriptor length */
        0,      /* density code */
        0,0,0,  /* Number of blocks (MSB) - (LSB) */
        0,      /* reserved */
        0,2,0   /* Block length = 200 (MSB) - (LSB) */
};

/* Mode select disk default values */
mode_sel mdsel_dsk_def = {
        0,      /* reserved */
        0,      /* medium type */
        0,      /* byte2 */
        8,      /* block descriptor length */
        0,      /* density code */
        0,0,0,  /* Number of blocks (MSB) - (LSB) */
        0,      /* reserved */
        0,2,0   /* Block length = 200 (MSB) - (LSB) */
};

/* Defect list formats - to be used with the format command */
byte def_lst_fmt[] = {
        0,      /* Block Format */
        0x4,    /* Index Format */
        0x5     /* Physical Sector Format */
};
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             cfprobe                                      *
*                                                                            *
*       Calling Sequence:       cfprobe()                                    *
*                                                                            *
*       Called by:              UNIX initialization                          *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:                                                     *
*                               timer - for reset timeout                    *
*                               xpb - pointer to extended parameter block    *
*                               setup - setup command list parameter block   *
*                               iocgpb - set iocg parameter block            *
*                               gopt - set general board options             *
*                                                                            *
*       Calls Subroutines:      pokec, sglcmd                                *
*                                                                            *
*       Public/Global Variables:                                             *
*                                                                            *
*       Description:                                                         *
*               This routine is called at startup time to probe for a        *
*               controller, to reset it, and to configure it for any options *
*                                                                            *
*****************************************************************************/

static
cfprobe(rf, ctlr)
REGISTER RF35REG *rf;
{
        dword timer;

        DEBUG(0x1,printf("cfprobe: controller %d at %x\n", ctlr, rf);)
        if (ctlr >= NCFC) {
                DEBUG(0x1, printf("cfprobe: bad controller number %d\n", ctlr);)
                return(0);
        }

        /* Reset controller and check if it exists */
        if (pokec((char *)&rf->reset, 1) != 0) {
                /* its not there */
                DEBUG(0x1, printf("cfprobe: no controller present at %x\n", rf);
                return(0);
        }
        bsybit[ctlr] = 0;

        /* Wait for reset to complete */
        timer = 0;
        while ((rf->status & SWAB(STATRST)) != SWAB(RESETDONE))
#ifdef MAXWAIT
                if (++timer > MAXWAIT) {
                        printf("cfprobe: Cannot reset controller: status %x\n",
                        return(0);
                }
```



```
#else MAXWAIT
                ;
#endif MAXWAIT
        rfsdctype[ctlr] = rf->status & STATCTYPE;

        DEBUG(0x1, printf("Controller %d is a ", ctlr);)
        DEBUG(0x1, printf("%s\n", rfsdctype[ctlr] == STATRF3500 ? "3500" : "Unkn

        if (RFSD_RESET_DELAY) {

#ifndef STANDALONE
                printf("Waiting %d second%s for SCSI bus reset completion.\n", R
#endif
                DELAY(RFSD_RESET_DELAY * 1000000);
        }
        if (rfsdctype[ctlr] != STATRF3500){
                return(0);
        }
        return(sizeof(RF35REG));
}
```

```
/*****************************************************************************
*                                                                            *
*       Subroutine:             cfslave                                      *
*                                                                            *
*       Calling Sequence:       cfslave(dev, rf)                             *
*                                                                            *
*       Called by:              UNIX configuration startup                   *
*                                                                            *
*       Calling Parameters:     dev - pointer to device structure            *
*                               rf - controllers I/O address                 *
*                                                                            *
*       Local Variables:        xpb - and extended parameter block           *
*                               pb - standard parameter block                *
*                                                                            *
*       Calls Subroutines:      sglcmd                                       *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*               This routine checks for the existence of the specified drive.*
*               We do this by issuing a drive interrogation command.  If the *
*               controller fails this command, then the drive is not usable. *
*                                                                            *
*****************************************************************************/

/*ARGSUSED*/
static
cfslave(dev, rf)
REGISTER struct mb_device *dev;
REGISTER RF35REG *rf;
{
        REGISTER EXTPB *xpb;
        REGISTER PARMBLK *pb;
        REGISTER UOPPB *upb;
        REGISTER GOPTPB *gopt;
        int timer;
        int dtype;
        int btarget, bunit;
        short unit;
        char *iomem;
        inq_data *inqdat;
        int error, index;
        int found = 0;
        int options;

        /* Get the index into the device_word structure for the device type,
           target id, logical unit number, and unit options.  This is actually
           the minor device number array, but the slave and attach routines
           don't know about minor device numbers. */

        for (index = 0; index <= MAXMINOR; index++) {
                if (dev->md_unit == rfdinfo[index].dev_index) {
                        unit = rfdinfo[index].dev_index; /* get unit number */
                        found = 1; /* found index into device_word array */
                        break;
```



```
                }
        }

        if (!found) {
                printf("cfslave: config file doesn't match device_word array --c
                return(0);
        }

        /* If configured device is dummy, return without error */
        if (rfdinfo[index].dev_id == DUMMY)
        {
                DEBUG(0x100, printf("cfslave: found DUMMY device\n");)
                return(1);
        }

        /* options is the unit options in the rfdinfo array for this device */
        if(rfdinfo[index].tag_id == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[index].unit_ops;
        }

        /* Set the mode select values to the defaults during boot.  These
         * values can be changes after a device is opened with the
         * cs35ut utility.
         */
        if(rfdinfo[index].dev_id == DIR_ACC)
        {
                mdsel_val[unit] = mdsel_dsk_def;
        } else
        {
                mdsel_val[unit] = mdsel_tp_def;
        }

        /* Lets do error checking on the configuration since the device type
           has to be entered in both the config file (GENERIC) and the minor
           device number table (rfdinfo[]) */

        if(dev->md_dk) {
                if(rfdinfo[index].dev_id == SEQ_ACC) {
                        printf("devtype in config file does not match dev type i
                        return(0);
                }
        } else {
                if(rfdinfo[index].dev_id != SEQ_ACC) {
                        printf("devtype in config file does not match dev type i
                        return(0);
                }
        }

        /* Floppies are always there.....sort of. */
        if (rfdinfo[index].tag_id == 0xFE){
                return(1);
        }

        iomem = (char *)safealloc(MEMSIZE);
```



```
        if (iomem == (char *) NULL) {
                DEBUG(0x100, printf("cfslave: Can't allocate memory for iomem\n"
                return(0);
        }

        /* get an extended parameter block */
        xpb = (EXTPB *)safealloc(sizeof(EXTPB));
        if (xpb == (EXTPB *) NULL) {
                printf("cfslave: Cannot allocate memory for extended PB.\n");
                safefree((caddr_t) iomem);
                return(0);
        }


        /* Set the General Board Options */
        DEBUG(0x101, printf("cfslave: Setting General Board Options\n");)

        gopt = (GOPTPB *)&xpb->pb;
        bzero((char *)gopt, sizeof(EXTPB));
        gopt->id = 0;
        gopt->optflags = rfsdoptions;
        gopt->throttle = THROTTLE;
        gopt->ownid = coninfo[dev->md_ctlr];
        gopt->targetid = RF3500_ID;
        gopt->command = C_OPTION;

        if (sglcmd(xpb, cfcontroller_info[dev->md_ctlr])) {
                printf("cfslave: Cannot set General Board Options!\n");
                blkpr(&xpb->sb, sizeof(STATBLK));
                safefree((caddr_t)xpb);
                safefree((caddr_t) iomem);
                return(0);
        }

        /* Issue a 'unit options' command to set the unit up
         */
        bzero((char *)xpb, sizeof(EXTPB));
        upb = (UOPPB *) &xpb->pb;
        upb->distimeout = 100000; /* */
        upb->unitid = rfdinfo[index].tag_id; /* SCSI target id */
        upb->targetid = RF3500_ID;
        upb->seltimeout = 256;  /* Default is 0 */

        if (rfdinfo[index].dev_id != SEQ_ACC)   /* if device is not a tape */
        {
                upb->retrycntrl = RCRBE | RCRCE | RCRPE | RCISB | SCINT;
                upb->retrylimit = 3;
        } else
        {
                upb->retrycntrl = 0;
                upb->retrylimit = 0;
        }
        upb->uflags = 0;

        /* set the request sense, byte count.  This is set by the user in */
        /* the flags field of the config file for each device 1.4b*/
```



```
        if(options & REQLENHI) {
                if(options & REQLENLO) {
                        upb->reqlength = 32;
                } else {
                        upb->reqlength = 24;
                }
        } else {
                if(options & REQLENLO) {
                        upb->reqlength = 16;
                } else {
                        upb->reqlength = 0; /* set to 0 to select default (8) */
                                            /* for firmware compatablility */
                }
        }

        if (options & NORETRYSOFT) {
                upb->uflags |= UF_ISE;
        }
        if (options & SYNCHRONOUS) {
                upb->uflags |= UF_SYN;
        }
        upb->command = C_UNITOPT;
        DEBUG(0x201,printf("cfslave: Setting unit options for target id 0x%x\n",
        if (sglcmd(xpb, cfcontroller_info[dev->md_ctlr])) {
                DEBUG(0x201, printf("cfslave: cannot access device\n");)
                safefree((caddr_t)xpb);
                safefree((caddr_t)iomem);
                return(0);
        }

        /* If Seq. Access device (tape), don't check for device rdy till open*/
        /* Try once to see if the unit is ready, or wait up to 3 minutes */
        /* if LONGRDYWAIT is set in the flags. */
        if (rfdinfo[index].dev_id != SEQ_ACC) {  /* if device is not a tape */

                timer = (options & LONGRDYWAIT) ? 180 : 1;

                while (timer) {
                        /* Issue a 'test unit ready' command to check if the
                        * device is present
                        */
                        bzero((char *)xpb, sizeof(EXTPB));
                        xpb->pb.scdb.cmd = SC_READY;
                        xpb->pb.targetid = rfdinfo[index].tag_id;
                        xpb->pb.scdb.byte1 = rfdinfo[index].log_unit << 5;
                        DEBUG(0x201,printf("cfslave: Testing for device existenc

                        if ((error = sglcmd(xpb, cfcontroller_info[dev->md_ctlr]

                                if (sglcmd(xpb, cfcontroller_info[dev->md_ctlr])
                                        if (--timer) {
                                                DELAY(1000000); /* Wait 1 second
                                                continue;
                                        }
                                        DEBUG(0x201, printf("cfslave: cannot acc
                                        safefree((caddr_t)xpb);
```

```
                                        safefree((caddr_t)iomem);
                                        return(0);
                                } else
                                        break;
                        } else
                                break;
                }

        }
        /* success */
        safefree((caddr_t)xpb);
        safefree((caddr_t)iomem);
        return(1);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             cfattach                                     *
*                                                                            *
*       Calling Sequence:       cfattach(device)                             *
*                                                                            *
*       Called by:              UNIX configuration startup                   *
*                                                                            *
*       Calling Parameters:     device - the device to do the housework for  *
*                                                                            *
*       Local Variables:        rf - the controller address                  *
*                               xpb - extended parameter block               *
*                               label - disk label structure                 *
*                               cpb - configuration parameter block          *
*                               pb - standard parameter block                *
*                               usable - usable number of disk blocks        *
*                               physical - physical number of disk blocks    *
*                               gp - pointer to disk geometry                *
*                                                                            *
*       Calls Subroutines:      sglcmd                                       *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*               This function does preliminary setup work for a              *
*               specified slave number.  If the drive is found, the label is *
*               read in.                                                     *
*                                                                            *
*****************************************************************************/

static
cfattach(device)
REGISTER struct mb_device *device;
{
        REGISTER EXTPB *xpb;
        REGISTER CMDLIST *cmdptr;
        register int btarget, bunit;
        register int unit;
        register int dtype;
        REGISTER mode_sel *ms;
        REGISTER page_5 *p5;
        REGISTER page_4 *p4;
        REGISTER page_3 *p3;
        REGISTER page_2 *p2;
        REGISTER page_1 *p1;
        REGISTER page_20 *p20;
        struct dk_label *label;
        PARMBLK *pb;
        int usable, physical;
        struct dk_geom *gp;
        int i;
        unsigned short *wp;
        unsigned short cksum;
        char *iomem;
        int mxbsize, mnbsize, bsize;
```



```
        read_cap *rc;
        inq_data *inqdat;
        blk_lim *blklim;
        int offset;
        SETUPPB *setup;
        GOPTPB *gopt;
        UOPPB *upb;
        struct vec *vp;
        int res, index;
        int goodlabel = 0;
        int found = 0;
        int options;

        /* Get the index into the device_word structure for the device type,
           target id, logical unit number, and unit options.  This is actually
           the minor device number array, but the slave and attach routines
           don't know about minor device numbers. */

        for (index = 0; index <= MAXMINOR; index++) {
                if (device->md_unit == rfdinfo[index].dev_index) {
                        found = 1;
                        unit = rfdinfo[index].dev_index;
                        break;
                }
        }
        if (!found) {
                printf("cfattach: config file doesn't match device_word array ch
        }

        /* Is the unit value ok? */
        if (unit >= NCF) {
                printf("cfattach: Illegal unit=0x%x\n", unit);
                return(0);
        }

        /* If configured device is dummy, return without error */
        if (rfdinfo[index].dev_id == DUMMY)
        {
                DEBUG(0x100, printf("cfattach: attached DUMMY device\n");)
                return(1);
        }

        /* blksiz[unit] is filled in a global variable that is filled in during
         * the attach routine, and used in the open routine when reading the
         * label. */
        blksiz[unit] = 0; /* pre-init to bad value */

        /* allocate space for extended parameter block and command list(s) */
        xpb = (EXTPB *)safealloc(sizeof(EXTPB));
        if (xpb == (EXTPB *) NULL) {
                DEBUG(0x100, printf("cfattach: Can't allocate memory for xpb\n")
                return(0);
        }
        DEBUG(0x2, printf("cfattach: allocated EXTPB at %x\n", xpb);)

        /* allocate space for miscellaneous command information. */
```



```
        iomem = (char *)safealloc(MEMSIZE);
        if (iomem == (char *) NULL) {
                DEBUG(0x100, printf("cfattach: Can't allocate memory for iomem\n
                safefree((caddr_t)xpb);
                return(0);
        }
        DEBUG(0x2, printf("cfattach: allocated iomem at %x\n", iomem);)

        btarget = rfdinfo[index].tag_id;  /* btarget = scsi target id */
        bunit = rfdinfo[index].log_unit;  /* bunit = scsi logical unit # */
        dtype = rfdinfo[index].dev_id;    /* dtype = device type- DIR_ACC etc */
        if(rfdinfo[index].tag_id == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[index].unit_ops;   /* the unit options */
        }

        cmdptr = cmdq[device->md_ctlr];
        /* Allocate a command queue for each controller. */
        if (cmdptr == (CMDLIST *) NULL) {
                cmdq[device->md_ctlr] = (CMDLIST *)safealloc(sizeof(CMDLIST));
                DEBUG(0x2, printf("cfattach: allocated cmdlist at %x", cmdq[devi
                DEBUG(0x2, printf(" for controller %d\n", device->md_ctlr);)
                cmdptr = cmdq[device->md_ctlr];
                if (cmdptr == (CMDLIST *) NULL) {
                        printf("cfattach: Cannot allocate memory for command lis
                        safefree((caddr_t)xpb);
                        return(0);
                }

                /* Initialize the command list buffer:
                 *      set IN and OUT pointers to zero
                 *      set the list size fields
                 */
                cmdptr->pbin = 0;
                cmdptr->pbout = 0;
                cmdptr->sbin = 0;
                cmdptr->sbout = 0;
                cmdptr->pbsize = NPB;
                cmdptr->sbsize = NSB;
                cmdptr->resv[0] = cmdptr->resv[1] = 0;

                /* Use an extended parameter block to issue type 0 commands
                 * disable interrupts and wait for these commands to complete.
                 */
                xpb->intr  = 0;
                xpb->resv0 = 0;
                xpb->resv1 = 0;

                /* Issue a 'Start command list' command
                 * Let the controller know where the command list buffer is
                 */
                DEBUG(0x100, printf("cfattach: Issuing 'Start command list' comm

                setup = (SETUPPB *)&xpb->pb;
                bzero((char *)setup, sizeof(EXTPB));
```

```
		setup->id = 0;
		setup->addrmod = VME_ADD_MOD;
		setup->targetid = RF3500_ID;
		setup->mmeaddr = realaddr(cmdptr);
		vp = device->md_mc->mc_intr;
		setup->intr = (device->md_mc->mc_intpri << 8) | vp->v_vec;
		setup->command = C_STARTCL;

		if (sglcmd(xpb, device->md_mc)) {
			printf("cfattach: Cannot setup command list!\n");
			blkpr(&xpb->sb, sizeof(STATBLK));
			safefree((caddr_t)cmdptr);
			safefree((caddr_t)xpb);
			return(0);
		}

		/* Issue 'identify' command to the adapter */
		DEBUG(0x101, printf("cfattach: Issuing 'identify' command to the
		bzero((char *)&xpb->pb, sizeof(EXTPB));
		xpb->pb.targetid = RF3500_ID;
		xpb->pb.scdb.cmd = C_IDENTIFY;
		if (!sglcmd(xpb, device->md_mc))
			idstat[device->md_ctlr] = *(RETID *) &xpb->sb;
	}

	/* Issue 'Inquiry' command...is this device block or direct access? */
	/* or what?  If this is a tape, do not do an Inquiry command on boot*/
	/* because some tape drives (EXABYTE) require a tape to be loaded,  */
	/* for the command to complete with any sort of expected response.  */
	/* When the tape is not loaded, a hang condition was happening if Inq */
	/* command was issued.  */
	if(dtype != SEQ_ACC){
		bzero((char *)&xpb->pb, sizeof(EXTPB));
		xpb->pb.targetid = btarget;
		xpb->pb.addrmod = VME_ADD_MOD;
		xpb->pb.vmeaddr = RF_ADDR(iomem);
		xpb->pb.count = sizeof (inq_data);
		xpb->pb.scdb.cmd = SC_INQUIRY;
		xpb->pb.scdb.byte1 = bunit << 5;
		xpb->pb.scdb.byte4 = sizeof (inq_data);
		inqdat = (inq_data *) iomem;
		bzero((char *) inqdat, sizeof (inq_data));
		DEBUG(0x201,printf("cfattach: Issuing 'inquiry' command to unit
		if (sglcmd(xpb, device->md_mc)) {
			DEBUG(0x201, printf("cfattach: cannot inquire device\n")
			safefree((caddr_t)xpb);
			safefree((caddr_t)iomem);
			return(0);
		}

		utype[unit] = inqdat->dtype;
		DEBUG(0x201, printf("cfattach: device is a %x utype\n", dtype);)
		if (dtype == LUN_NOTPRES) {
			DEBUG(0x201, printf("cfattach: device not present\n");)
			safefree((caddr_t)xpb);
			safefree((caddr_t)iomem);
```



```
			return(0);
		}
		if ((btarget != 0xFE) && (utype[device->md_unit] != rfdinfo[inde
			printf("cfattach: Inquiry information does not match the
			safefree((caddr_t)xpb);
			safefree((caddr_t)iomem);
			return(0);
		}
	}

	switch (dtype) {
		case DIR_ACC:
		case WORM:
		case RDONLYDIR_ACC:
			break;
		default:
			if(dtype != SEQ_ACC) {
				printf("cfattach: Unimplemented device used: %s
				safefree((caddr_t)xpb);
				safefree((caddr_t)iomem);
				return(0);
			}
	}

	if (dtype == DIR_ACC || dtype == WORM || dtype == RDONLYDIR_ACC) {
		if (btarget != 0xFE) {
			/* Issue a mode sense for page 2. */
			bzero((char *)&xpb->pb, sizeof(EXTPB));
			pb = &xpb->pb;
			pb->id = 0;
			pb->targetid = btarget;
			pb->addrmod = VME_ADD_MOD;
			pb->vmeaddr = RF_ADDR(iomem);
			pb->count = MEMSIZE;
			pb->scdb.cmd = SC_SENMODE;
			pb->scdb.byte1 = bunit << 5;
			pb->scdb.byte2 = 0;
			pb->scdb.byte4 = 24;

			ms = (mode_sel *) iomem;
			DEBUG(0x201,printf("cfattach: Getting buffer parameters.
			if ((res = sglcmd(xpb, device->md_mc)) != 0)
				res = sglcmd(xpb, device->md_mc);/*Try again. */

			if (res == 0) {
				blksiz[unit] =  ms->blklen[0]<<16
					      | ms->blklen[1]<<8
					      | ms->blklen[2];
			}

		}
		if (btarget == 0xFE) {  /* Floppy */
			bzero((char *)&xpb->pb, sizeof(EXTPB));
			bzero((char *)iomem, sizeof(mode_sel));
			ms = (mode_sel *) iomem;
			xpb->pb.targetid = btarget;
```



```
			xpb->pb.addrmod = VME_ADD_MOD;
			xpb->pb.vmeaddr = RF_ADDR(iomem);
			xpb->pb.count = 44; /* used to be 36 */
			xpb->pb.scdb.cmd = SC_SELMODE;
			xpb->pb.scdb.byte1 = bunit << 5;
			xpb->pb.scdb.byte4 = 44; /* used to be 36 */
			ms->medium_type = FL_MEDIA(rfdinfo[index].unit_ops);
			ms->blk_des_len = 8;
			ms->density_code = 0;
			ms->nblk[0] = 0;
			ms->nblk[1] = 0;
			ms->nblk[2] = 0;
			ms->blklen[0] =BYTE2(FL_SECTS(rfdinfo[index].unit_ops));
			ms->blklen[1] =BYTE1(FL_SECTS(rfdinfo[index].unit_ops));
			ms->blklen[2] =BYTE0(FL_SECTS(rfdinfo[index].unit_ops));
			p5 = (page_5 *)ms->vend_uniq;
			bzero((char *)p5, sizeof (page_5));
			p5->page_code = 5;
			p5->page_length = 22;   /* Don't count length or code */
			p5->dsr = 3000;
			p5->hd_st_dly = 1;
			p5->on_dly = 10;
			p5->off_dly = 40;
			p5->hd_ld_dly = 1;
			/* The variable parameters */
			if ((FL_MEDIA(rfdinfo[index].unit_ops) == 0x1A) || (FL_M
				p5->xfer_rate = 0x1F4;  /* 500 kbit/s */
			else
				p5->xfer_rate = 0xFA;   /* 250 kbit/s */
			p5->nheads = 2 - (FL_MEDIA(rfdinfo[index].unit_ops) & 1)
			if (FL_SECTS(rfdinfo[index].unit_ops) <= 0) {
				printf("cfattach: Error in floppy block size con
				safefree((caddr_t)xpb);
				safefree((caddr_t)iomem);
				return(0);
			}
			p5->nbps = (word) FL_SECTS(rfdinfo[index].unit_ops);
			p5->spt = FL_SPT(rfdinfo[index].unit_ops);
			if (FL_MEDIA(rfdinfo[index].unit_ops) == 0x16 ||
			    FL_MEDIA(rfdinfo[index].unit_ops) == 0x1A ||
			    FL_MEDIA(rfdinfo[index].unit_ops) == 0x1E)
				p5->ncyls = 80;
			else if(FL_MEDIA(rfdinfo[index].unit_ops) <= 0xA) /*8"*/
				p5->ncyls = 76;
			else
				p5->ncyls = 40;
			/* Now adjust cylinders for the number of steps. */
			if (FL_RDSTP(rfdinfo[index].unit_ops) > 0)
				p5->ncyls /= (FL_RDSTP(rfdinfo[index].unit_ops)
			if (FL_MEDIA(rfdinfo[index].unit_ops) <= 0xA)  /* 8" */
				p5->s_wpre = 43;
			else
				p5->s_wpre = 255;
		/* Reduce Write Current same as Write Precomp for
		    all non-AT (high density, 0x1A) floppies.  */
			if (FL_MEDIA(rfdinfo[index].unit_ops) == 0x1A)
```



```
				p5->s_rwc = 255;
			if (!(rfdinfo[index].unit_ops & OS9FLOPPY)) {
				p5->ssn_s0 = 1;
				p5->ssn_s1 = 1;
			}

			/* Now do page 20... */
			p20 = (page_20 *) &p5[1];
			bzero((char *) p20, sizeof (page_20));
			p20->page_code = 0x20; /* Fixed value*/
			p20->page_length = 6;   /* bytes */
		/* If 0, length is 33bytes (single density) or 62 bytes (double den) */
			p20->post_index = 0;
		/* If 0, length is 33bytes (single density) or 62 bytes (double den) */
			p20->inter_sector = 0;
			p20->tverify = 0; /* Don't verify seeks */
			p20->tsteps = (FL_RDSTP(rfdinfo[index].unit_ops) + 1);

			DEBUG(0x201,printf("cfattach: Setting floppy parameters.
			if (sglcmd(xpb, device->md_mc)) {
				DEBUG(0x201, printf("cfattach: cannot set floppy
				safefree((caddr_t)xpb);
				safefree((caddr_t)iomem);
				return(0);
			}
			goto skiplabel;

		}

		label = (struct dk_label *)iomem;

		/* set up a read command for the first sector */
		pb = &xpb->pb;
		pb->id = 0;
		pb->targetid = btarget;
		pb->addrmod = VME_ADD_MOD;
		pb->vmeaddr = RF_ADDR(label);
		pb->count = blksiz[unit];
		pb->scdb.cmd = SC_READ;
		pb->scdb.byte1 = bunit << 5;
		pb->scdb.byte2 = 0;
		pb->scdb.byte3 = 0;     /* Sector # to read */
		pb->scdb.byte4 = 1;     /* # of sectors to read */

		if (sglcmd(xpb, device->md_mc)) {
		/* issue the read command and check for errors */
			/* Try again...some drives take two tries on the first t
			if (sglcmd(xpb, device->md_mc)) {
				/* the read failed, the drive is probably not fo
				printf("The disk may not exist or be formatted.\

				/* release memory used for disk label */
/*
*/
				safefree((caddr_t)label);

				/* Zero the geometry. */
				bzero(&geometry[device->md_unit], sizeof (struct
```

```
                        goto getinfo;
                }

        }
        /* read worked, see if there is a valid label on this sector */
        /* Compute what the checksum should be... */
        cksum = 0;
        for (i = 0, wp = (unsigned short *)label; i < sizeof(struct dk_l
                cksum ^= *wp++;
        if (cksum != label->dkl_cksum || label->dkl_magic != DKL_MAGIC)
#ifndef STANDALONE
                if (DKL_MAGIC != label->dkl_magic) {
                        if (btarget != 0xFE)
                                printf("cfattach: No label on unit %d\n"
                } else {
                        if (btarget != 0xFE)
                                printf("cfattach: Bad checksum in label
                }
#endif

                /* release memory used for disk label */
/*              safefree((caddr_t)label);
*/
skiplabel:


                /* Issue a mode sense for the pages of geometry. */
                bzero((char *)&xpb->pb, sizeof(EXTPB));
                pb = &xpb->pb;
                pb->id = 0;
                pb->targetid = btarget;
                pb->addrmod = VME_ADD_MOD;
                pb->vmeaddr = RF_ADDR(iomem);
                pb->count = MEMSIZE;
                pb->scdb.cmd = SC_SENMODE;
                pb->scdb.byte1 = bunit << 5;
                if (btarget == 0xFE) {  /* Floppy?  Use page 5. */
                        pb->scdb.byte2 = 5;
                        pb->scdb.byte4 = 34;
                } else {
                        pb->scdb.byte2 = 4;     /* Get page 4 first */
                        pb->scdb.byte4 = 30;
                }

                if (!sglcmd(xpb, device->md_mc)) {  /* This means it wor

                        ms = (mode_sel *) iomem;
                        /* Pull the geometry out of the pages... */
                        if (btarget == 0xFE) {  /* Floppy? */
                                offset = (ms->blk_des_len - 8);
                                p5 = (page_5 *) &ms->vend_uniq[offset];

                                gp = &geometry[device->md_unit];
                                gp->dkg_ncyl = p5->ncyls;
                                gp->dkg_acyl = 0;
                                gp->dkg_nhead = p5->nheads;
```



```
                                gp->dkg_bhead = 0;
                                gp->dkg_nsect = p5->spt;
                                gp->dkg_intrlv = 0;
                                gp->dkg_gap1 = 0;
                                gp->dkg_gap2 = 0;
                                goto getinfo;
                        }

                        gp = &geometry[device->md_unit];

                        offset = (ms->blk_des_len - 8);
                        p4 = (page_4 *) &(ms->vend_uniq[offset]);
                        gp->dkg_ncyl = p4->ncyl_b1;
                        gp->dkg_ncyl = (gp->dkg_ncyl << 8) | p4->ncyl_b2
                        gp->dkg_nhead = p4->nhead;

                        bzero((char *)&xpb->pb, sizeof(EXTPB));
                        pb = &xpb->pb;
                        pb->id = 0;
                        pb->targetid = btarget;
                        pb->addrmod = VME_ADD_MOD;
                        pb->vmeaddr = RF_ADDR(iomem);
                        pb->count = MEMSIZE;
                        pb->scdb.cmd = SC_SENMODE;
                        pb->scdb.byte1 = bunit << 5;
                        pb->scdb.byte2 = 3;
                        pb->scdb.byte4 = 34;

                        gp->dkg_gap1 = 0;
                        gp->dkg_gap2 = 0;
                        gp->dkg_bhead = 0;

                        if (sglcmd(xpb, device->md_mc)) {
                                /* Couldn't get page 3. */
                                /* keep minimal config */
                                gp->dkg_ncyl = 1;
                                gp->dkg_apc = 0;
                                gp->dkg_nhead = 1;
                                gp->dkg_nsect = 1;
                                gp->dkg_intrlv = 1;
                                goto getinfo;
                        }
                        offset = (ms->blk_des_len - 8);
                        p3 = (page_3 *) &(ms->vend_uniq[offset]);
                        if (p3->alttpzone != 0)
                                gp->dkg_apc = p3->altspzone / p3->alttpz
                        else
                                gp->dkg_apc = 0;
                        gp->dkg_apc = gp->dkg_apc * gp->dkg_nhead;
                        gp->dkg_nsect = p3->spt;
                        gp->dkg_intrlv = p3->interleave;
                        if(p3->alttpvol)
                                gp->dkg_acyl = p3->alttpvol / gp->dkg_nh
                        else
                                gp->dkg_acyl = 0;
```



```
                        } else {
                                /* keep minimal config */
                                gp = &geometry[device->md_unit];

                                gp->dkg_ncyl = 1;
                                gp->dkg_acyl = 0;
                                gp->dkg_apc = 0;
                                gp->dkg_nhead = 1;
                                gp->dkg_bhead = 0;
                                gp->dkg_nsect = 1;
                                gp->dkg_intrlv = 1;
                                gp->dkg_gap1 = 0;
                                gp->dkg_gap2 = 0;
                        }
                        if (btarget != 0xFE)
                                printf("%s%d: <Assuming: cyl %d alt %d hd %d sec
                                        gp->dkg_ncyl, gp->dkg_acyl, gp->dkg_nhea

                        goto getinfo;
                }

                goodlabel = 1;
                /* this disk has a valid label, lets see what it says */
                /* set the disk partition map according to the label */
                for (i = 0; i < NDKMAP; i++)
                        partitions[device->md_unit][i] = label->dkl_map[i];

                /* set the geometry according to the label */
                gp = &geometry[device->md_unit];
                gp->dkg_ncyl = label->dkl_ncyl;
                gp->dkg_acyl = label->dkl_acyl;
                gp->dkg_nhead = label->dkl_nhead;
                gp->dkg_bhead = label->dkl_bhead;
                gp->dkg_nsect = label->dkl_nsect;
                gp->dkg_intrlv = label->dkl_intrlv;
                gp->dkg_gap1 = label->dkl_gap1;
                gp->dkg_gap2 = label->dkl_gap2;
                gp->dkg_apc = label->dkl_apc;

                /* determine usable and physical sizes */
                usable = gp->dkg_ncyl * gp->dkg_nsect * gp->dkg_nhead;
                physical = usable + gp->dkg_acyl * gp->dkg_nsect * gp->dkg_nhead
                physical += gp->dkg_apc * (gp->dkg_acyl + gp->dkg_ncyl);
#ifndef STANDALONE
                printf("%s%d: <%s(%d/%d)>\n", cfcdriver.mdr_dname, device->md_un
                        label->dkl_asciilabel, usable, physical);
#endif

                /* release the label structure */
/*              rmfree(iopbmap, sizeof(struct dk_label), (caddr_t)label);*/

        } else if (rfdinfo[index].dev_id == SEQ_ACC) /* if device is tape */
        {
                uflagon(ugeneral, unit, index);
                /* release resources */
                safefree((caddr_t)xpb);
```



```
                safefree((caddr_t)iomem);
                return(1);
        }

getinfo:
        /* set up a read capacity command to find the record information */
        pb = &xpb->pb;
        bzero((char *)pb, sizeof(EXTPB));
        pb->id = 0;
        pb->targetid = btarget;
        pb->addrmod = VME_ADD_MOD;
        pb->vmeaddr = RF_ADDR(iomem);
        pb->count = sizeof (read_cap);
        pb->scdb.cmd = SC_RDCAP;
        pb->scdb.byte1 = bunit << 5;

        rc = (read_cap *)iomem;
        bzero(rc, sizeof (read_cap));
        DEBUG(0x1, printf("cfattach: Issuing 'read capacity'.\n");)
        /* issue the read capacity command and check for errors */
        if (sglcmd(xpb, device->md_mc)) {
                /* the read failed */
                DEBUG(0x1, printf("cfattach: Read capacity failed....\n");)
                printf("cfattach: Read capacity failed....\n");
        } else {
                /* ok...now copy the relevant part. */
                record_info[device->md_unit].bsize =
                        (rc->blklen[0] << 24) |
                        (rc->blklen[1] << 16) |
                        (rc->blklen[2] << 8) |
                        (rc->blklen[3]);
                record_info[device->md_unit].nblk =
                        (rc->nblk[0] << 24) |
                        (rc->nblk[1] << 16) |
                        (rc->nblk[2] << 8) |
                        (rc->nblk[3]);
                record_info[device->md_unit].nblk++;
                if (utype[device->md_unit] != SEQ_ACC && !goodlabel) {
                        partitions[device->md_unit][0].dkl_nblk = record_info[de
                        partitions[device->md_unit][0].dkl_cylno = 0;
                }
                DEBUG(0x1, printf("cfattach: bsize=%x nblks=%d\n",
                        record_info[device->md_unit].bsize,
                        record_info[device->md_unit].nblk);)
        }
        /* release resources and return */
        safefree((caddr_t)xpb);
        safefree((caddr_t)iomem);
        return(1);

}
```

```
/*****************************************************************************
*                                                                            *
*       Subroutine:     cfopen                                               *
*                                                                            *
*       Calling Sequence:       cfopen(dev, flag)                            *
*                                                                            *
*       Called By:      UNIX I/O System.                                     *
*                                                                            *
*       Calling Parameters:                                                  *
*               dev:    Major & minor number of device.                      *
*               flag:   specifies what kind of open mode (Read, Write, both..) *
*                                                                            *
*       Local Variables:                                                     *
*               unit:   device unit... form dev                              *
*               partno: partition number for disk - 0 for all other          *
*               bp:     pointer to buffer header                             *
*               PB:     adapter parameter block                              *
*               ms:     pointer to mode_sel structure                        *
*               rc:     pointer to read_cap structure                        *
*                                                                            *
*       Calls Subroutines:      rfsdcmd()                                    *
*                                                                            *
*       Public/Global Variables:                                             *
*               rfsdreset:  1 if board reset                                 *
*               rfsdbuf:    local buffer header                              *
*                                                                            *
*       Description:                                                         *
*               This routine is called when the special file associated with *
*               this adapter is opened.                                      *
*                                                                            *
*****************************************************************************/

cfopen(dev, flag)
dev_t dev;
int flag;
{
        register int MINER;
        register int unit;
        REGISTER struct mb_device *device;
        REGISTER byte partno;
        REGISTER struct buf *bp;
        REGISTER PARMBLK PB;
        REGISTER mode_sel *ms;
        REGISTER read_cap *rc;
        REGISTER inq_data *inqdat;
        REGISTER blk_lim *blklim;
        register int bsize;
        REGISTER byte bunit, btarget;
        register int write_prot;
        register int ididreserve;
        register int open_index;
        register int errco = 0;
        register int l;
        REGISTER char *iomem;
        REGISTER page_5 *p5;
```



```
        REGISTER page_4 *p4;
        REGISTER page_3 *p3;
        REGISTER page_2 *p2;
        REGISTER page_1 *p1;
        REGISTER page_20 *p20;
        REGISTER struct dk_geom *gp;
        int i, usable, physical;
        int offset;
        int mxbsize, mnbsize;
        int options;

        DEBUG(0x200, printf("cfopen: dev=%x  flag=%x\n", dev, flag);)

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        device = cfdrive_info[unit];


        if (unit >= NCF) {
                printf("cfopen: Illegal unit=0x%x\n", unit);
                return(EINVAL);
        }

        DEBUG(0x2, printf("cfopen: device %x.\n", device);)

        /* Check here for DUMMY device.  If dummy device return with ok */
        if (rfdinfo[MINER].dev_id == DUMMY)
        {
                DEBUG(0x100, printf("cfopen: opened DUMMY device\n");)
                return(0);
        }

        if (device == (struct mb_device *) NULL || !device->md_alive) {
                DEBUG(0x2, printf("cfopen: Unit %d not alive.\n", unit);)
                printf("cfopen: Unit %d not alive.\n", unit);
                return(ENXIO);
        }

        /* Calculate the device index into the uopen array for this device */
        if (rfdinfo[MINER].tag_id != 0xFE) {
                open_index = ((64 * device->md_ctlr) + ((8 * rfdinfo[MINER].tag_
        } else {
                open_index = ((64 * device->md_ctlr) + (rfdinfo[MINER].log_unit)
        }

         /* If the device is TAPE we return error because */
         /* only one open of TAPE is allowed at a time.   */

        if((uopen[open_index]) && (rfdinfo[MINER].dev_id == SEQ_ACC)) {
                printf("cfopen: Permission denied, device is already open\n");
                return(EBUSY);
        }

        btarget = rfdinfo[MINER].tag_id;
        bunit = rfdinfo[MINER].log_unit;
        partno = rfdinfo[MINER].partition;
```



```
        if(btarget == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[MINER].unit_ops;
        }
        bsize = record_info[unit].bsize;

        if (btarget == 0xFE) {
                if (partno >= FLPFMT) {
                        printf("cfopen: Illegal floppy device=0x%x\n", dev);
                        return(EINVAL);
                }
        }

        /* Get the buffer for commands... */
        l = splx(pritospl(device->md_mc->mc_intpri));
        bp = &rfsdbuf[unit];
        buf_free(dev, bp);
        splx(l);
        bp->b_dev = dev;

        /* allocate space for iomem */
        while ((iomem = (char *)safealloc(MEMSIZE)) == (char *)NULL) {
                if(iomem_count) {
                        iomem_wanted = 1;
                        sleep((caddr_t)&iomem_wanted, PRIBIO);
                } else {
                        printf("cfopen: Can't allocate memory for iomem\n");
                        DEBUG(0x100, printf("cfopen: Can't allocate memory for i
                        wakeprocess(bp);
                        return(ENOMEM);
                }
        }

        iomem_count++;
        DEBUG(2, printf("cfopen: allocated iomem at %x\n", iomem);)

        bp->b_un.b_addr = (caddr_t)dev;

        /* Initialize the ididreserve so that if the device cannot be */
        /* opened we can release the unit before returning. */
        ididreserve = 0;

        /* Issue a test unit ready command to be sure drive is ready.  This */
        /* was added because the Test Unit Ready command will no longer be  */
        /* issued during boot (attach routine) for tape devices. If Test    */
        /* Unit Ready command completes with an error, issue command again  */
        /* to give it another chance.                                       */

        bzero((char *)&PB, sizeof(PARMBLK));
        PB.scdb.cmd = SC_READY;
        PB.targetid = btarget;
        PB.scdb.byte1 = bunit << 5;
        DEBUG(0x201,printf("cfopen: Testing for device existence\n");)

        if (rfsdcmd(&PB, device->md_mc, unit)) {
```



```
                if (rfsdcmd(&PB, device->md_mc, unit)) {
                        DEBUG(0x201, printf("cfopen: cannot access device\n");)
                        errco = ENXIO;
                        goto error;
                }
        }

        /* Issue 'Inquiry' command...is this device block or direct access? */
        /* or what?  We will have to issue this command again even though   */
        /* already done on boot because it was not done if the device is a  */
        /* tape.                                                            */

        bzero((char *)&PB, sizeof(PARMBLK));
        PB.targetid = btarget;
        PB.addrmod = VME_ADD_MOD;
        PB.vmeaddr = RF_ADDR(iomem);
        PB.count = sizeof (inq_data);
        PB.scdb.cmd = SC_INQUIRY;
        PB.scdb.byte1 = bunit << 5;
        PB.scdb.byte4 = sizeof (inq_data);
        inqdat = (inq_data *) iomem;
        bzero((char *) inqdat, sizeof (inq_data));
        DEBUG(0x201,printf("cfopen: Issuing 'inquiry' command to unit %d\n", uni
        if (rfsdcmd(&PB, device->md_mc, unit)) {
                DEBUG(0x201, printf("cfopen: Inquiry command failed\n", bp->b_de
                errco = ENXIO;
                goto error;
        }

        utype[unit] = inqdat->dtype;
        DEBUG(0x201, printf("cfopen: device is a %x utype\n", utype[device->md_u
        if (utype[unit] == LUN_NOTPRES) {
                DEBUG(0x201, printf("cfopen: device not present\n");)
                errco = ENXIO;
                goto error;
        }
        if ((btarget != 0xFE) && (utype[unit] != rfdinfo[MINER].dev_id)) {
                printf("cfopen: Inquiry information does not match the device_wo
                errco = ENXIO;
                goto error;
        }

        /* If the device is already open then nothing more needs to
         * be done. If the device is TAPE we return error because
         * only one open of TAPE is allowed at a time, otherwise
         * open if successful.
         * For uopen tests, map partno for all disks to partno 0.
         */
        if (((rfdinfo[MINER].dev_id == DIR_ACC || rfdinfo[MINER].dev_id == WORM
            || (uopen[open_index])) {
                if (rfdinfo[MINER].dev_id == SEQ_ACC || rfdinfo[MINER].dev_id ==
                        wakeprocess(bp);
                        safefree((caddr_t)iomem);
                        iomem_count--;
                        if(iomem_wanted) {
                                iomem_wanted = 0;
```

```
                        wakeup((caddr_t)&iomem_wanted);
                }
                return(EACCES);
        }

        wakeprocess(bp);
        safefree((caddr_t)iomem);
        if(iomem_wanted) {
                iomem_wanted = 0;
                wakeup((caddr_t)&iomem_wanted);
        }
        iomem_count--;
        return(0);
}

/* If we should reserve the unit and it's not already reserved... */
if (!(options & NORESERVE) && !ureserved[unit]) {
        bzero((char *)&PB, sizeof(PARMBLK));
        if(errco = SCSI_RESERVE(device,PB,&rfdinfo[MINER],MINER,bp)) {
                printf("error occurred on SCSI_RESERVE\n");
                goto error;
        }

        if (rfdinfo[MINER].dev_id == SEQ_ACC || rfdinfo[MINER].dev_id ==
                ureserved[unit] |= 1;
        else
                ureserved[unit] |= (1 << partno);
        ididreserve = 1;
}


write_prot = 0; /* Assume not write protected. */
/* Do device specific initialization */
/* The dev setup for SEQ_ACC devices used to be done in */
/* the cfattach routine and was moved to the open routine */
/* at level 1.3. */
switch (rfdinfo[MINER].dev_id) {
        case SEQ_ACC:

                /* Read the block limits. */
                bzero((char *)&PB, sizeof(PARMBLK));
                PB.targetid = btarget;
                PB.addrmod  = VME_ADD_MOD;
                PB.vmeaddr = RF_ADDR(iomem);
                PB.count = 501; /* Let's not get any more than this... *
                PB.scdb.cmd = SC_RDBLKLIM;
                PB.scdb.byte1 = bunit << 5;
                blklim = (blk_lim *)iomem;
                bzero((char *)blklim, sizeof(blk_lim));

                DEBUG(0x201,printf("cfopen: Issuing 'read block limits'

                if (rfsdcmd(&PB, device->md_mc, unit)) {
                        DEBUG(0x201, printf("cfopen: 'read block limits'
                        errco = ENXIO;
                        goto error;
```



```
                }

                /* Grab the block size in case there is one. */
                mxbsize = 0;
                mxbsize = (blklim->mxblklen[0] << 16) |
                          (blklim->mxblklen[1] <<  8) |
                          (blklim->mxblklen[2]);
                mnbsize = 0;
                mnbsize = (blklim->mnblklen[0] << 8) |
                          (blklim->mnblklen[1]);
                if (mnbsize == mxbsize)
                        bsize = mxbsize;
                else
                        bsize = 0;

                if (((options & EXABYTE) && rfsd_ex_fixed) || (options &
                        bsize = 1024;
                }

DEBUG(0x201,printf("cfopen: mnbsize=%x, mxbsize=%x\n", mnbsize, mxbsize);)


                record_info[device->md_unit].bsize = bsize;
                record_info[device->md_unit].nblk = (1<<28); /* Infinity
                partitions[device->md_unit][0].dkl_cylno = 0;
                partitions[device->md_unit][0].dkl_nblk = (1<<28);

                uflagoff(uwritten, unit, MINER);
                uflagoff(uonfmk, unit, MINER);
                /* Fall thru */

        case DIR_ACC:
        case WORM:
        case RDONLYDIR_ACC:

        DEBUG(0x201,printf("cfopen: ops=%x\n", options);)
                /* Now let's select our options... */
                if ((!(options & NOOPN_MDSEL)) &&
                        (!(options & GEN_MODE) || (uflagtst(ugeneral, un
                        if (btarget == 0xFE){   /* Floppy */
                                bzero((char *)&PB, sizeof(PARMBLK));
                                bzero((char *)iomem, sizeof(mode_sel));
                                ms = (mode_sel *) iomem;
                                PB.targetid = btarget;
                                PB.addrmod = VME_ADD_MOD;
                                PB.vmeaddr = RF_ADDR(iomem);
                                PB.count = 44;
                                PB.scdb.cmd = SC_SELMODE;
                                PB.scdb.byte1 = bunit << 5;
                                PB.scdb.byte4 = 44;
                                ms->medium_type = FL_MEDIA(rfdinfo[MINER
                                ms->blk_des_len = 8;
                                ms->blklen[0] =BYTE2(FL_SECTS(rfdinfo[MI
                                ms->blklen[1] =BYTE1(FL_SECTS(rfdinfo[MI
                                ms->blklen[2] =BYTE0(FL_SECTS(rfdinfo[MI
                                p5 = (page_5 *)ms->vend_uniq;
```



```
                                bzero((char *)p5, sizeof (page_5));
                                p5->page_code = 5;
                                p5->page_length = 22;   /* Don't count 1
                                p5->dsr = 3000;
                                p5->hd_st_dly = 1;
                                p5->on_dly = 10;
                                p5->off_dly = 40;
                                p5->hd_ld_dly = 1;
                                /* The variable parameters */
                                if ((FL_MEDIA(rfdinfo[MINER].unit_ops) =
                                        p5->xfer_rate = 0x1F4;  /* 500 k
                                else
                                        p5->xfer_rate = 0xFA;   /* 250 k
                                p5->nheads = 2 -  (FL_MEDIA(rfdinfo[MINER
                                if (FL_SECTS(rfdinfo[MINER].unit_ops) <=
                                        printf("cfopen: Error in floppy
                                        errco = EINVAL;
                                        goto error;
                                }
                                p5->nbps = (word) FL_SECTS(rfdinfo[MINER
                                p5->spt = FL_SPT(rfdinfo[MINER].unit_ops
                                if (FL_MEDIA(rfdinfo[MINER].unit_ops) ==
                                    FL_MEDIA(rfdinfo[MINER].unit_ops) ==
                                    FL_MEDIA(rfdinfo[MINER].unit_ops) ==
                                        p5->ncyls = 80;
                                else if(FL_MEDIA(rfdinfo[MINER].unit_ops
                                        p5->ncyls = 76;
                                else
                                        p5->ncyls = 40;
                                /* Now adjust cylinders for the number o
                                if (FL_RDSTP(rfdinfo[MINER].unit_ops) >
                                        p5->ncyls /= (FL_RDSTP(rfdinfo[M
                                if (FL_MEDIA(rfdinfo[MINER].unit_ops) <=
                                        p5->s_wpre = 43;
                                else
                                        p5->s_wpre = 255;
                                /* Reduce Write Current same as Write Pr
                                 * for all non-AT (high density, 0x1A) f
                                if (FL_MEDIA(rfdinfo[MINER].unit_ops) ==
                                        p5->s_rwc = 255;
                                if (!(rfdinfo[MINER].unit_ops & OS9FLOPP
                                        p5->ssn_s0 = 1;
                                        p5->ssn_s1 = 1;
                                }

                        /* Page 20H (Vendor unique) flopppy disk configuration *
                        /* Start page 20 at the end of page 5 */
                                p20 = (page_20 *) &p5[1];
                                bzero((char *) p20, sizeof (page_20));
                                p20->page_code = 0x20; /* Fixed value*/
                                p20->page_length = 6;   /* bytes */
        /* If 0, length is 33bytes (single density) or 62 bytes (double den) */
                                p20->post_index = 0;
        /* If 0, length is 33bytes (single density) or 62 bytes (double den) */
                                p20->inter_sector = 0;
                                p20->tverify = 0; /* Don't verify seeks
```



```
                                p20->tsteps = (FL_RDSTP(rfdinfo[MINER].u
                                DEBUG(0x201,printf("cfopen: Setting flop
                                if (rfsdcmd(&PB, device->md_mc, unit)) {
                                        DEBUG(0x201, printf("cfopen: can
                                        errco = ENXIO;
                                        goto error;
                                }

                                /* Issue a mode sense for the pages of g
                                bzero((char *)&PB, sizeof(PARMBLK));
                                PB.id = 0;
                                PB.targetid = btarget;
                                PB.addrmod = VME_ADD_MOD;
                                PB.vmeaddr = RF_ADDR(iomem);
                                PB.count = MEMSIZE;
                                PB.scdb.cmd = SC_SENMODE;
                                PB.scdb.byte1 = bunit << 5;
                                PB.scdb.byte2 = 5;
                                PB.scdb.byte4 = 34;

                                if (!rfsdcmd(&PB, device->md_mc, unit))

                                        ms = (mode_sel *) iomem;
                                        /* Pull the geometry out of the
                                        offset = (ms->blk_des_len - 8);
                                        p5 = (page_5 *) &ms->vend_uniq[o

                                        gp = &geometry[device->md_unit];
                                        gp->dkg_ncyl = p5->ncyls;
                                        gp->dkg_acyl = 0;
                                        gp->dkg_nhead = p5->nheads;
                                        gp->dkg_bhead = 0;
                                        gp->dkg_nsect = p5->spt;
                                        gp->dkg_intrlv = 0;
                                        gp->dkg_gap1 = 0;
                                        gp->dkg_gap2 = 0;
                                }
                        } else {
                                /* Issue 'mode sense' command
                                 * before issueing mode select command
                                 */
                                bzero((char *)&PB, sizeof(PARMBLK));
                                ms = (mode_sel *) iomem;
                                bzero((char *)ms, sizeof(mode_sel));
                                PB.targetid = btarget;
                                PB.addrmod  = VME_ADD_MOD;
                                PB.vmeaddr = RF_ADDR(ms);
                                PB.scdb.cmd = SC_SENMODE;
                                PB.scdb.byte1 = (bunit << 5);

                                if (options & EXABYTE) {
                                        /* 2 vendor unique parameter */
                                        PB.scdb.byte4 = 14;
                                        PB.count = 14;

                                        if (rfsd_ex_MX4_23) {
```

```
                                        /* five vend unique */
                                        PB.scdb.byte4 = 17;
                                        PB.count = 17;
                                }
                        } else {
                                PB.scdb.byte4 = 12;
                                PB.count = 12;
                        }

                        DEBUG(0x201,printf("cfopen: Issuing 'mod
                        if (rfsdcmd(&PB, device->md_mc, unit)) {
                                DEBUG(0x201, printf("cfopen: 'mo
                                errco = ENXIO;
                                goto error;
                        }

                        /* Issue 'mode select' command */
                        bzero((char *)&PB, sizeof(PARMBLK));
                        PB.targetid = btarget;
                        PB.addrmod  = VME_ADD_MOD;
                        PB.vmeaddr = RF_ADDR(iomem);
                        PB.scdb.cmd = SC_SELMODE;
                        PB.scdb.byte1 = (bunit << 5);

                        ms->medium_type = mdsel_val[unit].medium
                        if (options & EXABYTE) {
                                ms->vend_uniq[0] = rfsd_ex_optio
                                ms->vend_uniq[1] = rfsd_ex_cart;
                                ms->vend_uniq[2] = rfsd_ex_motio
                                ms->vend_uniq[3] = rfsd_ex_recon
                                ms->vend_uniq[4] = rfsd_ex_gap;
                                /* 2 vendor unique parameter */
                                PB.scdb.byte4 = 14;
                                PB.count = 14;

                                if (rfsd_ex_MX4_23) {
                                        /* five vend unique */
                                        PB.scdb.byte4 = 17;
                                        PB.count = 17;
                                }
                        } else {
                                PB.scdb.byte4 = 12;
                                PB.count = 12;
                        }

                        ms->byte0 = mdsel_val[unit].byte0;

                        if (rfdinfo[MINER].dev_id == SEQ_ACC) {
                                /* Buffered Mode */
                                ms->byte2 = mdsel_val[unit].byte
                        }
                        if (rfdinfo[MINER].dev_id == WORM) {
                                /* Enable Blank Checking */
                                ms->byte2 = 1;
                        }
                        ms->blk_des_len = mdsel_val[unit].blk_de
```



```
                        ms->density_code = mdsel_val[unit].densi
                        /* zero = all the blocks */
                        ms->nblk[0] = 0;
                        ms->nblk[1] = 0;
                        ms->nblk[2] = 0;
                        ms->blklen[0] = BYTE2(bsize);
                        ms->blklen[1] = BYTE1(bsize);
                        ms->blklen[2] = BYTE0(bsize);

                        DEBUG(0x201,printf("cfopen: Issuing 'mod
                        if (rfsdcmd(&PB, device->md_mc, unit)) {
                                DEBUG(0x201, printf("cfopen: 'mo
                DEBUG(0x201,printf("cfopen: ops=%x\n", options);)
                                errco = ENXIO;
                                goto error;
                        }
                        }
                }
                break;
        }


        if (write_prot && (flag & FWRITE)) {
                errco = EROFS;
                goto error;
        }
        /* The open was successful!  Set the uopen flag for this "device
           partition" for disk, or "device" for tape or floppy */
        uopenon(uopen, MINER, open_index);

error:
        if (errco && ididreserve) {
                if (rfdinfo[MINER].dev_id == SEQ_ACC || rfdinfo[MINER].dev_id ==
                        ureserved[unit] &= ~1;
                else
                        ureserved[unit] &= ~(1 << partno);
                if (!ureserved[unit]) {
                        DEBUG(0x201, printf("cfopen: release device %x\n", bp->b
                        /* Release the unit. */
                        bzero((char *)&PB, sizeof(PARMBLK));
                        if(SCSI_RELEASE(device,PB,&rfdinfo[MINER],MINER,bp)) {
                                printf("error occurred on SCSI_RESERVE\n");
                        }
                }
        }
        wakeprocess(bp);
        safefree((caddr_t)iomem);
        if(iomem_wanted) {
                iomem_wanted = 0;
                wakeup((caddr_t)&iomem_wanted);
        }
        iomem_count--;

        return(errco);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:     cfclose                                              *
*                                                                            *
*       Calling Sequence:       cfclose(dev, flag)                           *
*               dev:    Major & minor number of device.                      *
*               flag:   specifies what kind of open mode (Read, Write, both..) *
*                                                                            *
*       Called By:      UNIX I/O System.                                     *
*                                                                            *
*       Calling Parameters:                                                  *
*                                                                            *
*       Local Variables:                                                     *
*               unit:   device unit... form dev                              *
*               lunit:  logical unit                                         *
*               bp:     pointer to buffer header                             *
*               PB:     adapter parameter block                              *
*               ms:     pointer to mode_sel structure                        *
*                                                                            *
*       Calls Subroutines:      rfsdcmd()                                    *
*                                                                            *
*       Public/Global Variables:                                             *
*               rfsdbuf:  local buffer header                                *
*                                                                            *
*       Description:                                                         *
*               This routine is called when the special file associated with *
*       this adapter is closed. If a device is opened more than once at one  *
*       time, close routine is only on the last close to the device.         *
*                                                                            *
*****************************************************************************/

cfclose(dev, flag)
dev_t dev;
int flag;
{
        register int MINER;
        register int unit;
        REGISTER struct mb_device *device;
        REGISTER PARMBLK PB;
        REGISTER struct buf *bp;
        REGISTER byte partno;
        register int btarget, bunit;
        register int open_index;
        int genmode;
        int l;
        int options;
        long a,n;

        DEBUG(0x300, printf("cfclose: dev=%x  dflag=%x\n",dev,flag);)

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        device = cfdrive_info[unit];
        btarget = rfdinfo[MINER].tag_id;
        bunit = rfdinfo[MINER].log_unit;
```



```
        partno = rfdinfo[MINER].partition;
        if(btarget == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[MINER].unit_ops;
        }

        /* Check here for DUMMY device.  If dummy device return with ok */
        if (rfdinfo[MINER].dev_id == DUMMY)
        {
                DEBUG(0x100, printf("cfclose: opened DUMMY device\n");)
                return(0);
        }

        /* Calculate the device index into the uopen array for this device */
        if (rfdinfo[MINER].tag_id != 0xFE) {
                open_index = ((64 * device->md_ctlr) + ((8 * rfdinfo[MINER].tag_
        } else {
                open_index = ((64 * device->md_ctlr) + (rfdinfo[MINER].log_unit)
        }

        /*
         * Do errlog if we are going to be rewinding.
         * Have to do errlog here, since we have
         * buffer busy later, and errlog will
         * deadlock trying to get our own buffer.
         * This was put here instead of in case statement because rfsd_errlog
         * hangs sleeping for bp to not be free.
         */
        if ((rfdinfo[MINER].dev_id == SEQ_ACC) && !(options & NOREWIND))
                if (options & EXABYTE)
                        rfsd_errlog(dev);

        /* Wait for the buffer header to get free */
        bp = &rfsdbuf[unit];
        l = splx(pritospl(cfdrive_info[unit]->md_mc->mc_intpri));
        buf_free(dev, bp);
        splx(l);
        bp->b_dev = dev;
        bp->b_un.b_addr = (caddr_t)dev;

        switch (rfdinfo[MINER].dev_id) {
            case DIR_ACC:
            case WORM:
            case RDONLYDIR_ACC:
                /* mark the device as not open */
                break;

            case SEQ_ACC:

                /* write a filemark if tape openopen for writing or if tape
                 * is open for read/write and something got written
                 */
                if ((flag&FWRITE) && (uflagtst(uwritten, unit, MINER))) {
                        genmode = ((!(options & GEN_MODE)) || (uflagtst(ugeneral
                        /* write a filemark */
```

```
                        DEBUG(0x3, printf("cfclose: writing filemark\n");)
                        bzero((char *)&PB, sizeof(PARMBLK));
                        PB.targetid = btarget;
                        PB.scdb.cmd = SC_WFM;
                        PB.scdb.byte1 = bunit << 5;
                        if ((options & (ONEFILEMARK|ONEFM)) || !genmode)
                                PB.scdb.byte4 = 1;
                        else
                                PB.scdb.byte4 = 2;      /* 2 file marks */
                        /* allow use of "short" filemarks */
                        if ((options & EXABYTE) && rfsd_ex_write_short_fm)
                                PB.scdb.byte5 = 0x80;
                        if (rfsdcmd(&PB, device->md_mc, unit))  {
                                printf("cfclose: cannot write filemark\n");
                                goto error;
                        }
                        /* Do it again if... */

                        if (PB.scdb.byte4 == 1 &&
                           !(options & ONEFM) &&
                           rfsdcmd(&PB, device->md_mc, unit)) {
                                printf("cfclose: cannot write filemark2\n");
                                goto error;
                        }
                        if (!(options & GEN_MODE) &&
                           (options & NOREWIND) &&
                           !(options & ONEFM)) {

                                /* back up over one of them */
                                DEBUG(0x3, printf("cfclose: searching -filemark\
                                bzero((char *)&PB, sizeof(PARMBLK));
                                PB.targetid = btarget;
                                PB.scdb.cmd = SC_SPACE;
                                PB.scdb.byte1 = bunit << 5 | SFM;
                                PB.scdb.byte2 = BYTE2(-1);
                                PB.scdb.byte3 = BYTE1(-1);
                                PB.scdb.byte4 = BYTE0(-1);
                                if (rfsdcmd(&PB, device->md_mc, unit))  {
                                        printf("cfclose: cannot find -filemark\n
                                        goto error;
                                }
                        }
                }

                for(n = 1; n < 20000; n = n + 1) {  /* Delay for search filemark
                        a = n;
                }

                /* rewind the tape if no rewind option not set */

                if (!(options & NOREWIND))  {
                        uflagon(ugeneral, unit, MINER);
                        DELAY(1000000);
                        DEBUG(0x3, printf("cfclose: rewinding tape\n");)
                        bzero((char *)&PB, sizeof(PARMBLK));
                        PB.targetid = btarget;
```



```
                        PB.scdb.cmd = SC_REWIND;
                        PB.scdb.byte1 = bunit << 5;
                        if (rfsdcmd(&PB, device->md_mc, unit))  {
                                printf("cfclose: cannot rewind tape\n");
                                goto error;
                        }
                }
        }

error:

        if (rfdinfo[MINER].dev_id == SEQ_ACC || rfdinfo[MINER].dev_id == PRN ||
                ureserved[unit] &= ~1;
        else
                ureserved[unit] &= ~(1 << partno);
        if (!ureserved[unit] && !(options & NORESERVE)) {

                /* Release the unit */
                bzero((char *)&PB, sizeof(PARMBLK));
                if(SCSI_RELEASE(device,PB,&rfdinfo[MINER],MINER,bp)) {
                        DEBUG(0x201, printf("cfclose: release device %x failed\n
                }

        }
        /* release the system buffer and wakeup any process
         * waiting for the buffer
         */
        wakeprocess(bp);

        /* mark the device as not open */
        uopenoff(uopen, MINER, open_index);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:     cfstrategy                                           *
*                                                                            *
*       Calling Sequence:       cfstrategy(bp)                               *
*                                                                            *
*       Called By:      UNIX I/O System.                                     *
*                                                                            *
*       Calling Parameters:                                                  *
*               bp:     Pointer to I/O request.                              *
*                                                                            *
*       Local Variables:                                                     *
*               rfreg:  pointer to the special registers on the adapter      *
*               unit:   device unit... form dev                              *
*               lunit:  logical unit                                         *
*               bp:     pointer to buffer header                             *
*               PB:     adapter parameter block                              *
*               last:   last valid block on the device                       *
*               targetid: SCSI target ID                                     *
*               targetunit: unit number on the SCSI target                   *
*               nsectors: number of blocks for data transfer                 *
*               blklen: device block length                                  *
*               pb:     pointer to adapter parameter block                   *
*               ps:     process status                                       *
*                                                                            *
*       Calls Subroutines:      none                                         *
*                                                                            *
*       Public/Global Variables:                                             *
*               cmdq:   command list to issue commands to the adapter        *
*                                                                            *
*       Description:                                                         *
*                                                                            *
*****************************************************************************/

void
cfstrategy(bp)
REGISTER struct buf *bp;
{
        register int MINER;
        register int unit;
        REGISTER struct mb_device *device;
        REGISTER RF35REG *rfreg;
        REGISTER byte partno;
        REGISTER dword last;
        REGISTER byte btarget, bunit;
        REGISTER struct dk_map *partp;
        struct  dk_geom *gp;
        dword   nsectors;
        dword   blklen;
        PARMBLK *pb;
        dword   ps;
        int     mbinfo, mbwaitflag;
        int     options;

        DEBUG(0x400, printf("cfstrategy: dev=%x flags=%x  bn=%x cc=%x ma=%x\n",b
```



```
        MINER = minor(bp->b_dev);
        unit = rfdinfo[MINER].dev_index;
        device = cfdrive_info[unit];
        rfreg = (RF35REG *) device->md_mc->mc_addr;
        if(rfdinfo[MINER].tag_id == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[MINER].unit_ops;
        }

        if(rfdinfo[MINER].dev_id == DUMMY)
        {
                printf("cfstrategy: cf%x is a DUMMY device and cannot be written
        }

        if (bp->b_bcount <= 0) {
                iodone(bp);
                return;
        }

        partno = rfdinfo[MINER].partition;
        btarget = rfdinfo[MINER].tag_id;
        bunit = rfdinfo[MINER].log_unit;
        gp = &geometry[device->md_unit];

        if (btarget == 0xFE) {          /* FLOPPY */
                partno = 0;
                partp = &partitions[unit][partno];
                partp->dkl_nblk = record_info[unit].nblk;
        } else
                partp = &partitions[unit][partno];

        switch (rfdinfo[MINER].dev_id) {
                case DIR_ACC:
                case WORM:
                case RDONLYDIR_ACC:
                        blklen = record_info[unit].bsize;
                        last = record_info[unit].nblk;
                        bp->b_bn = bp->b_blkno;
                        bp->b_bn += (partp->dkl_cylno * gp->dkg_nsect * gp->dkg_
                        if (blklen) {
                                bp->b_bn = (bp->b_bn * DEV_BSIZE) / blklen;
                                /* Going past the end of the partition? */
                                if (bp->b_blkno >= partp->dkl_nblk ||
                                    (bp->b_blkno + (bp->b_bcount / blklen) > par
                                        printf("ERROR - PAST PARTITION\n");
                                        bp->b_flags |= B_ERROR;
                                        bp->b_error = EIO;
                                        iodone(bp);
                                        return;
                                }
                        }
                        break;
                case SEQ_ACC:
                        blklen = record_info[unit].bsize;
```

```
                        last = record_info[unit].nblk;
                        bp->b_bn = 0;
                        break;
                default:
                        goto error;
        }

        DEBUG(0x400,
                printf("cfstrategy: last=%d\n", last);
        )
        /* If trying to read next to the last block then
         * set residual count to transfer count and return.
         * If trying to do any other transfers outside the
         * valid block range then set the error and return.
         */
        if (bp->b_blkno < 0 || bp->b_blkno >= last) {
                if (bp->b_blkno == last && bp->b_flags & B_READ)
                        bp->b_resid = bp->b_bcount;
                else {
                        bp->b_resid = bp->b_bcount;
                        bp->b_flags |= B_ERROR;
                        bp->b_error = EIO;
                }
                iodone(bp);
                return;
        }

        /* Make sure the I/O count is multiple of device block size
         * and calculate the number of blocks to be transfered
         * This only applies if blklen is set.  Otherwise transfer size is
         * in bytes.  This is mainly for variable length tape blocks.
         */
        DEBUG(0x400,
                printf("cfstrategy: blklen=%d\n", blklen);
        )
        if (blklen) {
                if ((bp->b_bcount/blklen*blklen) != bp->b_bcount) {
                        printf("RF3500: dev %x, I/O count %x   device block size
                        printf("RF3500: dev %x, I/O not multiple of device block
                        goto error;
                }
                nsectors = bp->b_bcount / blklen;
        } else
                nsectors = bp->b_bcount;        /* really:  nbytes */

        /* return EOF if prev read was short read before FMK */
        if  (uflagtst(uonEOF, unit, MINER)) {
                uflagoff(uonEOF, unit, MINER);  /* clears flg on write */
                if ((options & EXABYTE) && (bp->b_flags & B_READ)) {
                        bp->b_resid = bp->b_bcount; /* zero byte read */
                        bp->b_error = 0;
                        iodone(bp);
                        return;
                }
        }
```



```
        /* Flag the tape as being written */
        if (rfdinfo[MINER].dev_id == SEQ_ACC) {
                uflagoff(ugeneral, unit, MINER);
                if ((bp->b_flags & B_READ) == 0) {
                        uflagon(uwritten, unit, MINER);
                } else {
                        uflagoff(uwritten, unit, MINER);
                }
        }

        if (bp != &rfsdbuf[unit]) {
                /* We'll get called on the interrupt service stack if called */
                /* from the network disk interface. */
#ifndef SunOS4
                mbwaitflag = intsvc() ? MB_CANTWAIT : 0;
                /* map memory to main bus */
                bp->b_saddr = 0;
                mbinfo = mbsetup(device->md_hd, bp, mbwaitflag);
                if (mbwaitflag && mbinfo == 0) {
                        timeout(cfstrategy, bp, hz);
                        return;
                }
#else
                bp->b_saddr = 0;
                mbinfo = mbsetup(device->md_hd, bp, 0);
#endif
                /* save mbinfo for release in rfintr */
                bp->b_mbinfo = (struct buf *)mbinfo;
                bp->b_md_hd = (struct buf *)device->md_hd;
        }

        ps = splx(pritospl(device->md_mc->mc_intpri));

        /* -Point to the parameter block in the command list
         * -Set up the parameter block
         * -Issue the command to the controller
         */
#if NPBMASK
        while (((cmdq[device->md_ctlr]->pbin + 1) & NPBMASK) == cmdq[device->md_
#else
        while ((cmdq[device->md_ctlr]->pbin + 1) % NPB == cmdq[device->md_ctlr]-
#endif NPBMASK
        {
                DEBUG(0x402, printf("cfstrategy: command list full\n");)
                pb_wanted[device->md_ctlr] = 1;
                sleep((caddr_t)&pb_wanted[device->md_ctlr], PRIBIO);
        }
        DEBUG(0x402, printf("cfstrategy: pbin 0x%x\n",cmdq[device->md_ctlr]->pbi
        pb = &cmdq[device->md_ctlr]->pblist[cmdq[device->md_ctlr]->pbin];

        bzero((char *)pb, sizeof(PARMBLK));
        pb->id = (dword)bp;
        pb->targetid = btarget;
        pb->addrmod  = VME_ADD_MOD;
        pb->vmeaddr  = MBI_ADDR(mbinfo);
        pb->count    = bp->b_bcount;
```



```
        pb->scdb.cmd = (bp->b_flags & B_READ) ? SC_READ : SC_WRITE;
        if (rfdinfo[MINER].dev_id == SEQ_ACC) {
                pb->scdb.byte1 = (bunit << 5) | (BYTE2(bp->b_bn) & 0x1F);
                if (blklen)
                        pb->scdb.byte1 |= 1;    /* Fixed */
                pb->scdb.byte2 = BYTE2(nsectors);
                pb->scdb.byte3 = BYTE1(nsectors);
                pb->scdb.byte4 = BYTE0(nsectors);
        } else {
                pb->scdb.byte1 = (bunit << 5) | (BYTE2(bp->b_bn) & 0x1F);
                pb->scdb.byte2 = BYTE1(bp->b_bn);
                pb->scdb.byte3 = BYTE0(bp->b_bn);
                pb->scdb.byte4 = nsectors;
        }

        DEBUG(0x403,
                blkpr(pb, sizeof(PARMBLK));
                printf("cfstrategy: issuing channel attention\n");
        )

#if NPBMASK
        cmdq[device->md_ctlr]->pbin = (cmdq[device->md_ctlr]->pbin + 1) & NPBMAS
#else
        cmdq[device->md_ctlr]->pbin = (cmdq[device->md_ctlr]->pbin + 1) % NPB;
#endif NPBMASK
        rfreg->attention = SWAB(1);
        /*printf("cfstrategy: dev=%x flags=%x  bn=%x cc=%x ma=%x bufsz=%x resid=

        splx(ps);
        return;

error:  /* Cannot process the request. Set the user error flag and return */
        bp->b_flags |= B_ERROR;
        bp->b_error = EIO;
        iodone(bp);
        return;
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:     cfintr                                          *
*                                                                            *
*       Calling Sequence:       cfintr()                                     *
*                                                                            *
*       Called By:      Unix interrupt service mechanism                     *
*                                                                            *
*       Calling Parameters:     none                                         *
*                                                                            *
*       Local Variables:                                                     *
*               sb:     pointer to the status block in 'cmdq'                *
*               bp:     pointer to the buffer header                         *
*               error:  error in command execution                           *
*                                                                            *
*       Calls Subroutines:      none                                         *
*                                                                            *
*       Public/Global Variables:                                             *
*               cmdq:   command list to issue commands to the adapter        *
*                                                                            *
*       Description:                                                         *
*                                                                            *
*****************************************************************************/

cfintr(ctlr)
{
        REGISTER STATBLK *sb;
        REGISTER struct buf *bp;
        REGISTER byte error = 0;
        register int s;
        REGISTER long resid;
        REGISTER struct mb_device *device;
        register int MINER;
        register int unit;
        int blklen;
        int leotflag = 0;
        int a;
        byte target_id;
        int cindex;
        int options;

        DEBUG(0x500, printf("cfintr:\n");)

        /* process interrupts on all controllers */
        for (cindex = 0; cindex < NCFC;) {

           error = 0;
           /* if no interrupt pending, go on to the next controller
            * by incrementing the controller index (cindex) */
           if ((cmdq[cindex] == (CMDLIST *) NULL) || (cmdq[cindex]->sbin == cmdq
              cindex++;
           } else {

              /* If some process is sleeping on command list wake him up */
              if (pb_wanted[cindex]) {
```

```
                pb_wanted[cindex] = 0;
                wakeup((caddr_t)&pb_wanted[cindex]);
             }

             /* Process all the status blocks in the status queue
              * one by one.
              */
             while (cmdq[cindex]->sbin != cmdq[cindex]->sbout) {
             /* point to the status block */
                sb = &cmdq[cindex]->sblist[cmdq[cindex]->sbout];

#if NSBMASK
                cmdq[cindex]->sbout = (cmdq[cindex]->sbout + 1) & NSBMASK;
#else
                cmdq[cindex]->sbout = (cmdq[cindex]->sbout + 1) % NSB;
#endif NSBMASK

                /* get buffer address from the id field of the status blk */
                bp = (struct buf *)sb->id;

                MINER = minor(bp->b_dev);
                unit = rfdinfo[MINER].dev_index;
                if(rfdinfo[MINER].tag_id == 0xFE) {
                   options = 0;
                } else {
                   options = rfdinfo[MINER].unit_ops;
                }

                if (PRINT_TARGET_ID) {
                   target_id = rfdinfo[MINER].tag_id;
                } else {
                   target_id = unit;
                }
                device = cfdrive_info[unit];
                blklen = record_info[device->md_unit].bsize;
                resid = 0;    /* Assume it all got there. */

                /* Errors!!! */
                error = sb->error;

                if (rfsd_pr_sense & 02) {
                   printf("cf: dbg: ");
                   blkpr(sb, sizeof(STATBLK));
                }

    /* If the continued flag isn't set, get the information out of the status
       block. This information can be overridden if we have retries at things,
       but since most of the critical information is for tape devices (the
       residual count, for example) the normal case is that a single extended
       status block will finish the operation. */

                if (!(sb->flags & ST_CON)) {

                /* If a soft error has occurred, scsi sense key = 1 */
                /* print out the status block. */
```



```
                if (sb->flags & ST_SOFT) {
                   printf("cfintr: rf3500 soft error\n");
                   blkpr(sb, sizeof(STATBLK));
                }

                DEBUG(0x501,
                printf("cfintr: sbout = %xH\n",cmdq[cindex]->sbout);
                blkpr(sb, sizeof(STATBLK));
                }

                if (error) {

                   if (rfsd_pr_sense & 01) {
                      printf("cf: dbge: ");
                      blkpr(sb, sizeof(STATBLK));
                   }
                   resid = bp->b_bcount;      /* Assume nothing got there. */
                   if (error == EE_SCSIERR) {
                   /* Look at the SCSI status */
                      if ((sb->scsistat & STATMASK) == CHECK_COND) {
                         switch (sb->scsiflags & SENSEMASK) {
                            case MISCOMPARE:
                               bp->b_flags |= B_ERROR;
                               if (sb->class & ADVALID){
                                  if (blklen) {
                                     resid = (((sb->infob3 << 24) | (sb->infob

                                  } else {
                                     resid = (sb->infob3 << 24) | (sb->infob4

                                  }
                               }
                               error = 0;
                               break;
                            case UNIT_ATTEN:
                               bp->b_flags |= B_ERROR;
                               /* tape was chaged, clear errlog */
                               kb_xfer[unit] = 0;
                               lastop[unit] = UNDEF;
                               error = 0;
                               break;
                            case RECOVERED:
                               if(PRINT_RECOV_ERRORS) {
                                      printf("RF3500: dev %x (unit %x) recover
                                      if (!PRINT_SB_HEADER) {
                                             blkpr(((byte *) sb) + 8, sizeof(
                                      } else {
                                             blkpr(sb, sizeof(STATBLK));
                                      }
                               }
                               resid = 0;
                               error = 0;
                               break;
                            case PROTECTED:
                               error = 0;
                               bp->b_error = EROFS;
```



```
                               bp->b_flags |= B_ERROR;
                               break;
                            case BLANK:
                               error = 0;
                               bp->b_flags |= B_ERROR;
                               bp->b_error = EIO;
                               break;
                            case NOSENSE:
                               if (sb->scsiflags & FM) {
                               /* only set EOF pending if rd-ignore fsr/bsr */
                                  if (bp == &rfsdbuf[unit]) {
                                     error = resid = 0;
                                     break;
                                  }
                                  uflagon(uonfmk, unit, MINER);
                                  /* another filemark flag to make reads work
                                  uflagon(uonEOF, unit, MINER);
                                  if (sb->class & ADVALID){
                                     if (blklen) {
                                        resid = (((sb->infob3 << 24) | (sb->in
                                     } else {
                                        resid = (sb->infob3 << 24) | (sb->info
                                     }
                                  }
                                  error = 0;
                                  break;
                               }
                               /* LEOT check  */
                               if (options & EXABYTE){
                                  if ((sb->scsiflags & EOM)) {
                                      /* maybe spaced to EOT or bksp to BOT */
                                      /*
                                       * another check like ILI will
                                       * also post EOM if after the
                                       * LEOT. ILI happens on reads,
                                       * LEOT only on writes.
                                       */
                                      if (bp != &rfsdbuf[unit] &&
                                        (sb->scsiflags & ILI) &&
                                        (bp->b_flags & B_READ))
                                              goto ili;
                                      /*
                                       * Note: we stuff SCSI cmd in b_resid
                                       * If special buf, we probably
                                       * spaced off end of tape.
                                       * allow WEOF to go thru..
                                       * It will be ok.. but will
                                       * take an LEOT check on every
                                       * OP, but EOF will be written.
                                       * for MX 4$23 and later. --ghg
                                       */
                                      if (bp == &rfsdbuf[unit]) {
                                         if (bp->b_resid == SC_WFM ||
                                             bp->b_resid == SC_SENSE)
                                            error = 0;
                                         else
```



```
                                            error = EIO;
                                         break;
                                      }
                                      if (sb->class & ADVALID) {
                                         if (blklen) {
                                            resid = (((sb->infob3 << 24) | (sb-
                                         } else {
                                            resid = (sb->infob3 << 24) | (sb->i
                                         }
                         /* handle "bug" in Exabyte resid at LEOT in var mode */
                                      } else {
                                         resid=0; /* prob var reclen */
                                /* Exabyte -
                                 * Drives built or upgraded around 4/21/88
                                 * appear to exhibit strange behavior when
                                 * hitting LEOT in var blk. Most of the time..
                                 * ADVALID is not set, and infob[3456] are
                                 * 0x00 0xff 0xff 0xff.  Record does not
                                 * need to be rewritten for this. Rarely,
                                 * ADVALID is not set and infob[3456] are
                                 * set to 0x00 0x00 0x00 0x00.. and the
                                 * just issued write needs to be issued
                                 * again. infob[3456] should be meaningless
                                 * if ADVALID is not set. --ghg 06/19/88.
                                 * This still happens on 01/07/89. MX 4$23
                                 * firmware. --ghg
                                 */
                                       if (blklen == 0 &&
                                       sb->infob3 == 0 &&
                                       sb->infob4 == 0 &&
                                       sb->infob5 == 0 &&
                                       sb->infob6 == 0)
                                            resid = bp->b_bcount;

                                       }
                                       if (rfsd_pr_leot)
                                          printf("exabyte:%d hit LEOT WARNING (2
                                          error=0;
                                          break;
                                       }
                                  }
                ili:
                                /* If we have an ILI bit, OR if this isn't a
                                   read/write and we have an EOM, clear the
                                   error info. */
                                if ((sb->scsiflags & ILI) || ((sb->id == (dword
                                   if (sb->class & ADVALID) {
                                      if (blklen) {
                                         resid = (((sb->infob3 << 24) | (sb->in

                                      } else {
                                         resid = (sb->infob3 << 24) | (sb->info
                                       /*
                                        * If reading in var mode, a record
                                        * longer than your buffer, you will
```

```
                                        * (correctly) take in ILI. Exabyte
                                        * Sets resid < 0 if record was longer
                                        * than user buffer. Correct Unix
                                        * operation is to silently truncate
                                        * the record.
                                        */
                                         if (resid < 0)
                                               resid = 0;
                                      }
                                   }
                                   error = 0;
                                   break;
                                }
                             default:
                                printf("RF3500: dev %x (unit %x), SCSI sense ke
                                break;
                             }
                             if(PRINT_RECOV_ERRORS || !((sb->scsiflags & SENSEM
                                blkpr(sb, sizeof(STATBLK));
                             }

                          } else if ((sb->scsistat & STATMASK) == DEVICE_BUSY)
                             printf("RF3500: dev %x (unit %x), SCSI device busy
                          } else {
                             printf("RF3500: dev %x (unit %x), unknown SCSI sta
                          }

                       }
                       else if (error < EE_FRMERR)
                          printf("RF3500: dev %x (unit %x), adapter error %x, %
                       else
                          printf("RF3500: dev %x (unit %x), adapter error %x, i
                       } else {
                          if (bp->b_error) { /* if old error */
                             printf("RF3500: dev %x (unit %x) error cleared by
                          }
                          bp->b_error = 0;
                          bp->b_flags &= ~B_ERROR;
                       }

                       /* print out the status block if error */
                       if (error) {
                          if (error == EE_SCSIERR && !PRINT_SB_HEADER) {
                             blkpr(((byte *) sb) + 8, sizeof(STATBLK) - 8);
                          } else {
                             blkpr(sb, sizeof(STATBLK));
                          }
                       }

                       if ((options & EXABYTE) && resid != bp->b_bcount) {
                          if (bp != &rfsdbuf[unit] &&
                             (resid < 0 || resid > bp->b_bcount)) {
                             printf("bogus resid %d\n", resid);
                             resid = bp->b_bcount;
                          }
                       }
```



```
                       if (bp != &rfsdbuf[unit] && resid == bp->b_bcount){
                          /* read filemark in a read by itself */
                          uflagoff(uonEOF, unit, MINER);
                       }


                       if (bp == &rfsdbuf[unit]) {
                          bp->b_error = error;
                       } else {
                          if (error) {
                             bp->b_resid = bp->b_bcount;
                             bp->b_error = EIO;
                             bp->b_flags |= B_ERROR;
                          } else {
                             bp->b_resid = resid;
                          }
                       }

                    } else {

/*       If this IS a continued block, just print it out, since the sense info
         has already been sent.  Note that the only time we should see this
         information is on an error or a retry, so no test is required to
         see if we have an error.

*/

                       if (sb->error == EE_SCSIERR && !PRINT_SB_HEADER) {
                          printf("RF3500: sense code %x from unit %x.  Sense data:
                          blkpr(((byte *) sb) + 8, sizeof(STATBLK) - 8);
                       } else {
                          printf("RF3500: additional sense bytes received for unit
                          blkpr(sb, sizeof(STATBLK));
                       }
                    }

                 /* If the command is now complete (ST_CC set), then do the
                    final processing.
                  */

                    if (sb->flags & ST_CC) {

                       if (bp == &rfsdbuf[unit]) {
                          wakeup(&bp->b_un.b_addr);
                       } else {
                          mbrelse(bp->b_md_hd, &bp->b_mbinfo);
                          iodone(bp);
                       }
                    }
                 }        /* end of while loop */
              }  /* end of if/else loop checking that sbin != sbout */
           }  /* end of for loop (for each controller in system) */
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:     cfread & cfwrite                                     *
*                                                                            *
*       Calling Sequence:       cfread(dev) or cfwrite(dev)                  *
*                                                                            *
*       Called By:      UNIX I/O System...this provides the Raw interface.   *
*                                                                            *
*       Calling Parameters:                                                  *
*               dev:    Major & Minor of device to perform I/O on.           *
*                                                                            *
*       Local Variables:                                                     *
*                                                                            *
*       Calls Subroutines:      cfstrategy (via physio)                 *
*                                                                            *
*       Public/Global Variables:                                             *
*                                                                            *
*       Description:                                                         *
*               We provide the Raw I/O interface for UNIX...still, UNIX does *
*               practically all of the work for us.  We do a sanity check on *
*               the drive unit.                                              *
*                                                                            *
*****************************************************************************/

cfread(dev, uio)
dev_t dev;
struct uio *uio;
{
        int MINER;
        int unit;

        int status, resid;
        int options;

        DEBUG(0x500, printf("cfread:\n");)

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        if(rfdinfo[MINER].tag_id == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[MINER].unit_ops;
        }

        /* If dummy device, return an error.  Dummy device can't be
         * read from.
         */
        if(rfdinfo[MINER].dev_id == DUMMY)
        {
                printf("read: cf%x is a DUMMY device and cannot be read from\n",
                return(ENXIO);
        }

        if (unit >= NCF)
                return(ENXIO);
```



```
        if (status = chk_uio(dev, uio)){
                return(status);
        }
        if (options & EXABYTE) {
                if (lastop[unit] == WRITE)
                        rfsd_errlog(dev);
                lastop[unit] = READ;
                resid = uio->uio_resid;
        }
        status=physio(cfstrategy, &rrfsdbuf[unit], dev, B_READ, minphys, uio);
        if (status == 0 && options & EXABYTE) {
                if (rfsd_pr_errlog & 0x8)
printf("cfread: %d kbytes read\n", (resid - uio->uio_resid + 1023) / 1024);
                kb_xfer[unit] += (resid - uio->uio_resid + 1023) / 1024;
        }
        return(status);
}

cfwrite(dev, uio)
dev_t dev;
struct uio *uio;
{
        register int MINER;
        register int unit;
        REGISTER struct mb_device *device;
        REGISTER struct buf *bp;
        int error;
        int status;
        int options;

        DEBUG(0x500, printf("cfwrite:\n");)

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        device = cfdrive_info[unit];
        if(rfdinfo[MINER].tag_id == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[MINER].unit_ops;
        }

        /* If dummy device, return an error.  Dummy device can't be
         * written to.
         */
        if(rfdinfo[MINER].dev_id == DUMMY)
        {
                printf("cfwrite: cf%x is a DUMMY device and cannot be written to
                return(ENXIO);
        }

        if (unit >= NCF) {
                printf("UNIT = %x, NCF = %x \n", unit, NCF);
                return(ENXIO);
        }
        if (status = chk_uio(dev, uio)){
                return(status);
```

```
        }

        if (options & EXABYTE) {
                if (lastop[unit] == READ)
                        rfsd_errlog(dev);
                lastop[unit] = WRITE;
                if (rfsd_pr_errlog & 0x8)
printf("cfwrite: %d kbytes written\n", (uio->uio_resid + 1023) / 1024);
                kb_xfer[unit] += (uio->uio_resid + 1023) / 1024;
        }

        /* kludge for Exabyte LEOT */
        error = physio(cfstrategy, &rrfsdbuf[unit], dev, B_WRITE, minphys, uio);
        /* stupid LEOT.. restart aborted write */
        if (options & EXABYTE) {
                bp = &rrfsdbuf[unit];
                if (error == 0 && bp->b_resid) {
                        uio->uio_iov->iov_base += (bp->b_bcount - bp->b_resid);
                        /* iov_len already adj in physio.. */
                        error = physio(cfstrategy, &rrfsdbuf[unit],
                           dev, B_WRITE, minphys, uio);
                }
        }
        return(error);
}

static int
chk_uio(dev, uio)
dev_t           dev;
struct uio      *uio;
{
        int     i, unit, blklen, extra;
        struct iovec    *iov;
        int MINER;

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;

        switch (utype[unit]) {
        case DIR_ACC:
        case WORM:
        case RDONLYDIR_ACC:
                blklen = record_info[unit].bsize;
                if(blklen == 0){
                        return(ENXIO);
                }
                if(uio->uio_offset % blklen) {
                        return(EINVAL); /* must start on block boundary */
                }
                extra = 0;
                for(iov = &uio->uio_iov[i = 0]; i < uio->uio_iovcnt; i++, iov++)
                        extra = (extra + iov->iov_len) % blklen;
                if(extra){
                        return(EINVAL); /* must end on block boundary */
                }
        }
```



```
        return(0); /* all other combinations are ok */
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:     cfioctl                                              *
*                                                                            *
*       Calling Sequence:       cfioctl(dev, cmd, addr, flag)                *
*                                                                            *
*       Called By:      UNIX I/O System                                      *
*                                                                            *
*       Calling Parameters:                                                  *
*                                                                            *
*       Local Variables:                                                     *
*               unit:   device unit... form dev                              *
*               lunit:  logical unit                                         *
*               bp:     pointer to buffer header                             *
*               pb:     pointer to adapter parameter block                   *
*               level:  debug level                                          *
*               interleave: format interleave                                *
*               lba:    logical block address                                *
*               ps:     process status                                       *
*                                                                            *
*       Calls Subroutines:      rfsdcmd                                      *
*                                                                            *
*       Public/Global Variables:                                             *
*               rfsddbg:        debug level                                  *
*               rfsdbuf:        local buffer header                          *
*                                                                            *
*       Description:                                                         *
*               This routine provides the capability of doing operations that *
*       are out of the ordinary.                                             *
*                                                                            *
*       NOTE: ioctl's are limited to the super user.                         *
*                                                                            *
*****************************************************************************/

cfioctl(dev, cmd, addr, flag)
dev_t dev;
caddr_t addr;
{
        register int MINER;
        register int unit;
        REGISTER struct mb_device *device;
        REGISTER PARMBLK PB, *pb;
        REGISTER struct buf *bp;
        REGISTER struct mtop *mtop;
        REGISTER struct mtget *mtgetp;
        REGISTER RF35REG *rf;
        REGISTER UOPPB *upb;
        REGISTER GOPTPB *gopt;
        REGISTER EXTPB *xpb;
        REGISTER CMDLIST *cmdptr;
        REGISTER format *fmt;
        REGISTER def_list *def_lst;
        REGISTER mode_sel *ms;
        REGISTER inq_data *inqdat;
        register int open_idx_base;
```



```
        SETUPPB *setup;
        page_3 *pg3;
        page_4 *pg4;
        dword level;
        dword interleave;
        dword lba;
        int ps;
        register int errco, offset;
        REGISTER caddr_t iomem;
        byte bunit;
        register int ctlr;
        struct dk_info information;
        struct dk_vfy *vreq;
        struct dk_mapr *mreq;
        struct defect_list *deflp;
        def_list *deflst;
        struct dk_map xpart;
        struct vec *vp;
        read_cap *rc;
        int partno;
        int options;
        int timer;
        int i;
        int good_deflist;
        byte def_list_byte = 0;

        DEBUG(0x700,
                printf("cfioctl: dev %x  cmd %c/%d  addr %x  flag %x\n",
                        dev, (cmd>>8) & 0xFF, cmd & 0xFF, addr, flag);
        )

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        mtop = (struct mtop *) addr;
        partno = rfdinfo[MINER].partition;
        bunit = rfdinfo[MINER].log_unit;

        if(rfdinfo[MINER].tag_id == 0xFE) {
                options = 0;
        } else {
                options = rfdinfo[MINER].unit_ops;
        }

        device = cfdrive_info[unit];
        if (device == (struct mb_device *) NULL || !device->md_alive) {
                printf("cfioctl: Unit %d not alive.\n", unit);
                return(ENXIO);
        }

        ctlr = device->md_ctlr;
        rf = (RF35REG *) device->md_mc->mc_addr;

        /* You may want to remove this test if you want to allow normal
         * users to do ioctl's on disks.
         */
        if (utype[unit] != SEQ_ACC && !suser()) {
```

```
        return(EPERM);
    }

    /* allocate space for miscellaneous command information. */
    while ((iomem = (char *)safealloc(MEMSIZE)) == (char *)NULL) {
        if(iomem_count) {
                iomem_wanted = 1;
                sleep((caddr_t)&iomem_wanted, PRIBIO);
        } else {
                DEBUG(0x100, printf("cfioctl: Can't allocate memory for
                return(ENOMEM);
        }
    }
    iomem_count++;
    DEBUG(0x2, printf("cfioctl: allocated iomem at %x\n", iomem);)

    bzero((char *)&PB, sizeof(PARMBLK));
    PB.id = (dword)&rfsdbuf[unit];

    /* rewind on target 1 didnt work without this --ghg */
    PB.targetid = rfdinfo[MINER].tag_id;

    /* Execute device independent commands */
    switch (cmd) {
        case RFIOCGTARGET:
            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            ((target *)addr)->id = rfdinfo[MINER].tag_id;
            ((target *)addr)->unit = rfdinfo[MINER].log_unit;
            return(0);
        case DKIOCGGEOM:    /* Get drive configuration */
            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            *(struct dk_geom *)addr = geometry[unit];
            return(0);

        case DKIOCINFO:             /* Get information */

            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            information.dki_ctlr = (int) (&cfcontroller_info[ctlr]->mc_addr)
            information.dki_unit = device->md_unit;
            information.dki_ctype = DKC_RF3500;
            information.dki_flags = DKI_FMTVOL;
```



```
            *(struct dk_info *)addr = information;
            return(0);
            break;

        case RFIOCSDEBUG:   /* set debug level */

            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            level = *(dword *)addr;
            rfsddbg = level;
            return(0);

        case RFIOCGDEBUG:   /* get debug level */

            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            *(dword *)addr = rfsddbg;
            return(0);

        case RFIOCGDEVID:   /* get device id (SEQ_ACC, DIR_ACC ...) */

            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            *(dword *)addr = rfdinfo[MINER].dev_id;
            return(0);

        case RFIOCGBLKSIZE: /* get blocksize */
            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            *(dword *)addr = record_info[unit].bsize;
            return(0);

        case RFIOCSBLKSIZE: /* set blocksize */
            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            record_info[unit].bsize = *(dword *)addr;
```



```
            return(0);

        case RFIOCIDENT:    /* identify */

            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }
            *(RETID *)addr = idstat[ctlr];
            return(0);

        case RFIOCRESET:    /* Reset the Adapter */

            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }

            /* Reset controller and check if it exists */
            if (pokec((char *)&rf->reset, 1) != 0) {
                    /* its not there */
                    DEBUG(0x1, printf("cfprobe: no controller present at %x\
                    return(EIO);
            }

            /* Wait for reset to complete */
            timer = 0;
            while ((rf->status & SWAB(STATRST)) != SWAB(RESETDONE))
#ifdef MAXWAIT
                    if (++timer > MAXWAIT) {
                            printf("cfioctl: Cannot reset controller: status
                            return(EIO);
                    }
#else MAXWAIT
                    ;
#endif MAXWAIT

            if (RFSD_RESET_DELAY) {

                    printf("Waiting %d second%s for SCSI bus reset completio
                    DELAY(RFSD_RESET_DELAY * 1000000);
            }
            if (rfsdctype[ctlr] != STATRF3500){
                    return(0);
            }

            printf("Clear ou flags that will no longer be valid\n");
            /* Clear out various flags that will no longer be valid */

            /* Calculate the device index base for the devices on this
             * controller only.
             */
```



```
#ifdef notyet
            open_idx_base = 64 * device->md_ctlr;
            bzero((char *)uopen[open_idx_base], sizeof(uopen)/NCFC);
            bzero((char *)ureserved[NCF], sizeof(byte));
            bzero((char *)uwritten[NCF], sizeof(byte));
            bzero((char *)uonfmk[NCF], sizeof(byte));
            bzero((char *)uonEOF[NCF], sizeof(byte));
#endif

            printf("RFIOCRESET: Return to cs35ut\n");
            return(0);
            break;

        case RFIOCGENOPT:

            printf("RFIOCGENOPT:\n");
            safefree((caddr_t)iomem);
            iomem_count--;
            if(iomem_wanted) {
                    iomem_wanted = 0;
                    wakeup((caddr_t)&iomem_wanted);
            }

            printf("RFIOCGENOPT: allocate mem for cmd list\n");
            /* allocate mem for ext pb for command list cmd */
            xpb = (EXTPB *)safealloc(sizeof(EXTPB));
            if (xpb == (EXTPB *) NULL) {
                    DEBUG(0x100, printf("cfioctl: Can't allocate memory for
                    return(EIO);
            }


            printf("Setting up PB for General Options cmd\n");
            /* Set the General Board Options */
            gopt = (GOPTPB *)&xpb->pb;
            bzero((char *)gopt, sizeof(EXTPB));
            gopt->id = 0;
            gopt->optflags = rfsdoptions;
            gopt->throttle = THROTTLE;
            gopt->ownid = coninfo[device->md_ctlr];
            gopt->targetid = RF3500_ID;
            gopt->command = C_OPTION;
            printf("RFIOCGENOPT: Issue General Options command\n");

            if (sglcmd(xpb, cfcontroller_info[device->md_ctlr])) {
                    printf("cfioctl: Cannot set General Board Options!\n");
                    blkpr(&xpb->sb, sizeof(STATBLK));
                    safefree((caddr_t)xpb);
                    return(EIO);
            }
            printf("RFIOCGENOPT: Return to cs35ut\n");
            return(0);
            break;

        case RFIOCUNITOPT:
```

```
		printf("RFIOCUNITOPT:\n");
		safefree((caddr_t)iomem);
		iomem_count--;
		if(iomem_wanted) {
			iomem_wanted = 0;
			wakeup((caddr_t)&iomem_wanted);
		}

		printf("RFIOCUNITOPT: allocate mem for cmd list\n");
		/* allocate mem for ext pb for command list cmd */
		xpb = (EXTPB *)safealloc(sizeof(EXTPB));
		if (xpb == (EXTPB *) NULL) {
			DEBUG(0x100, printf("cfioctl: Can't allocate memory for
			return(EIO);
		}

		/* Issue a 'unit options' command to set the unit up */
		bzero((char *)xpb, sizeof(EXTPB));
		upb = (UOPPB *)&xpb->pb;
		upb->distimeout = 100000; /* */
		upb->unitid = rfdinfo[MINER].tag_id; /* SCSI target id */
		upb->targetid = RF3500_ID;
		upb->seltimeout = 256;  /* Default is 0 */

		if (rfdinfo[MINER].dev_id != SEQ_ACC)   /* if dev isnt a tape */
		{
			upb->retrycntrl = RCRBE | RCRCE | RCRPE | RCISB | SCINT;
			upb->retrylimit = 3;
		} else
		{
			upb->retrycntrl = 0;
			upb->retrylimit = 0;
		}
		upb->uflags = 0;

		/* set the request sense, byte count.  This is set by the usr */
		/* the unit_ops field of the rfdinfo array for each device. */
		if(options & REQLENHI) {
			if(options & REQLENLO) {
				upb->reqlength = 32;
			} else {
				upb->reqlength = 24;
			}
		} else {
			if(options & REQLENLO) {
				upb->reqlength = 16;
			} else {
				upb->reqlength = 0; /* default (8) */
			}
		}

		if (options & NORETRYSOFT) {
			upb->uflags |= UF_ISE;
		}
		if (options & SYNCHRONOUS) {
			upb->uflags |= UF_SYN;
```



```
		}
		upb->command = C_UNITOPT;

		printf("RFIOCUNITOPT: Issue a unit options command\n");
		if (sglcmd(xpb, cfcontroller_info[device->md_ctlr])) {
			printf("cfioctl: Cannot set Unit Options!\n");
			blkpr(&xpb->sb, sizeof(STATBLK));
			safefree((caddr_t)xpb);
			return(EIO);
		}
		printf("RFIOCUNITOPT: return to cs35ut\n");
		return(0);
		break;

	case RFIOCCMDLST:
		cmdq[device->md_ctlr] = (CMDLIST *) NULL;

		/* Release iomem, iomem not used for cmdlst */
		safefree((caddr_t)iomem);
		iomem_count--;
		if(iomem_wanted) {
			iomem_wanted = 0;
			wakeup((caddr_t)&iomem_wanted);
		}

		/* allocate mem for ext pb for command list cmd */
		xpb = (EXTPB *)safealloc(sizeof(EXTPB));
		if (xpb == (EXTPB *) NULL) {
			DEBUG(0x100, printf("cfioctl: Can't allocate memory for
			return(EIO);
		}

		cmdptr = cmdq[device->md_ctlr];
		/* Allocate a command queue for each controller. */
		if (cmdptr == (CMDLIST *) NULL) {
			cmdq[device->md_ctlr] = (CMDLIST *)safealloc(sizeof(CMDL
			DEBUG(0x2, printf("cfioctl: allocated cmdlist at %x", cm
			DEBUG(0x2, printf(" for controller %d\n", device->md_ctl
			cmdptr = cmdq[device->md_ctlr];
			if (cmdptr == (CMDLIST *) NULL) {
				printf("cfioctl: Cannot allocate memory for comm
				safefree((caddr_t)xpb);
				return(EIO);
			}

			/* Initialize the command list buffer:
			 * set IN and OUT pointers to zero
			 * set the list size fields
			 */
			cmdptr->pbin = 0;
			cmdptr->pbout = 0;
			cmdptr->sbin = 0;
			cmdptr->sbout = 0;
			cmdptr->pbsize = NPB;
			cmdptr->sbsize = NSB;
			cmdptr->resv[0] = cmdptr->resv[1] = 0;
```



```
			bzero((char *)cmdptr->pblist, sizeof(PARMBLK) * NPB);
			bzero((char *)cmdptr->sblist, sizeof(STATBLK) * NSB);

			/* Use an ext pb to issue type 0 commands
			 * dis intr & wait for cmds to complete.
			 */
			xpb->intr  = 0;
			xpb->resv0 = 0;
			xpb->resv1 = 0;

			/* Issue a 'Start command list' command
			 * Let the cntr know where the cmd list
			 * buffer is
			 */
			DEBUG(0x100, printf("cfioctl: Issuing 'Start command lis

			setup = (SETUPPB *)&xpb->pb;
			bzero((char *)setup, sizeof(EXTPB));
			setup->id = 0;
			setup->addrmod = VME_ADD_MOD;
			setup->targetid = RF3500_ID;
			setup->memaddr = realaddr(cmdptr);
			vp = device->md_mc->mc_intr;
			setup->intr = (device->md_mc->mc_intpri << 8) | vp->v_ve
			setup->command = C_STARTCL;

			if (sglcmd(xpb, device->md_mc)) {
				printf("cfattach: Cannot setup command list!\n")
				blkpr(&xpb->sb, sizeof(STATBLK));
				safefree((caddr_t)cmdptr);
				safefree((caddr_t)xpb);
				return(EIO);
			}
		}

		safefree((caddr_t)xpb);
		return(0);
		break;

	case RFIOCMDSEN:
		/* Issue a mode sense receiving just the parameter list
		 * without additional pages. */
		PB.addrmod = VME_ADD_MOD;
		PB.vmeaddr = RF_ADDR(iomem);
		PB.count = MEMSIZE;
		PB.scdb.cmd = SC_SENMODE;
		PB.scdb.byte1 = bunit << 5;
		if(rfdinfo[MINER].unit_ops & EXABYTE)
		{
			PB.scdb.byte4 = 14;

			if (rfsd_ex_MX4_23) {
				/* five vend unique */
				PB.scdb.byte4 = 17;
			}
		} else
```



```
		{
			PB.scdb.byte4 = 12;
		}

		goto exec;
		break;

	case RFIOCMDSEL:
		ms = (mode_sel *) iomem;
		*ms = *(mode_sel *)addr;
		/* Issue a mode select command sending just the parameter list
		 * without additional pages. */
		PB.addrmod = VME_ADD_MOD;
		PB.vmeaddr = RF_ADDR(iomem);
		PB.count = MEMSIZE;
		PB.scdb.cmd = SC_SELMODE;
		PB.scdb.byte1 = bunit << 5;
		if(rfdinfo[MINER].unit_ops & EXABYTE)
		{
			PB.scdb.byte4 = 14;

			if (rfsd_ex_MX4_23) {
				/* five vend unique */
				PB.scdb.byte4 = 17;
			}
		} else
		{
			PB.scdb.byte4 = 12;
		}

		/*ms->vend_uniq[0] = rfsd_ex_optional1;
		ms->vend_uniq[1] = rfsd_ex_cart;*/
		mdsel_val[unit] = *ms;
		goto exec;
		break;

	case RFIOCMDSEN3:
		pg3 = (page_3 *)addr;
		/* Issue a mode sense for the pages of geometry. */
		PB.addrmod = VME_ADD_MOD;
		PB.vmeaddr = RF_ADDR(iomem);
		PB.count = MEMSIZE;
		PB.scdb.cmd = SC_SENMODE;
		PB.scdb.byte1 = bunit << 5;
		PB.scdb.byte2 = pg3->page_code;
		PB.scdb.byte4 = 34;

		goto exec;
		break;

	case RFIOCMDSEN4:

		pg4 = (page_4 *)addr;
		/* Issue a mode sense for the pages of geometry. */
		PB.addrmod = VME_ADD_MOD;
		PB.vmeaddr = RF_ADDR(iomem);
```

```
                PB.count = MEMSIZE;
                PB.scdb.cmd = SC_SENMODE;
                PB.scdb.byte1 = bunit << 5;
                PB.scdb.byte2 = pg4->page_code;
                PB.scdb.byte4 = 30;

                goto exec;
                break;

            case RFIOCINQ:
                /* Issue a SCSI Inquiry command */
                PB.addrmod = VME_A24_MOD;
                PB.vmeaddr = RF_ADDR(iomem);
                PB.count = sizeof(inq_data);
                PB.scdb.cmd = SC_INQUIRY;
                PB.scdb.byte1 = bunit << 5;
                PB.scdb.byte4 = sizeof(inq_data);
                inqdat = (inq_data *) iomem;
                bzero((char *)inqdat, sizeof(inq_data));

                goto exec;
                break;

            case RFIOCLOAD:
                PB.scdb.cmd = SC_LOAD;
                PB.scdb.byte4 = LOAD;
                goto exec;

        /* unload */
UNLOAD:

            case RFIOCUNLOAD:
                PB.scdb.cmd = SC_LOAD;
                PB.scdb.byte1 |= 1;     /* immed */
                PB.scdb.byte4 = 0;
                goto exec;

            /* "mt status" - not much returned, but no hard error */
            case MTIOCGET:              /* get tape status */
                mtgetp = (struct mtget *) addr;
                /*
                 * Ought to do something with these sometime --ghg
                 */
                mtgetp->mt_erreg = 0;
                mtgetp->mt_resid = 0;
                mtgetp->mt_dsreg = 0;
                mtgetp->mt_fileno = 0;
                mtgetp->mt_blkno = 0;
                mtgetp->mt_type = 0x28; /* what sun uses for Exabyte */
                goto out;

            case MTIOCTOP:
                uflagoff(uwritten, unit, MINER);
                /*
                 * If we are about to do a "mt bsr" or "mt fsr" and
                 * have a "pending" EOF (read but not delivered)
```



```
                 * we have to bksp over the EOF and clear the
                 * pending flag. --ghg.
                 */
                if (uflagtst(uonEOF, unit, MINER)) {
                        switch (mtop->mt_op) {
                            case MTFSR:
                            case MTBSR:
                                PB.scdb.cmd = SC_SPACE;
                                PB.scdb.byte1 |= SFM;
                                PB.scdb.byte2 = 0xFF;
                                PB.scdb.byte3 = 0xFF;
                                PB.scdb.byte4 = 0xFF;
                                PB.targetid = rfdinfo[MINER].tag_id;
                                PB.scdb.byte1 |= rfdinfo[MINER].log_unit << 5;
                                /* Wait for the buffer header to get free */
                                ps = splx(pritospl(cfcontroller_info[ctlr]->mc_i
                                bp = &rfsdbuf[unit];
                                while (bp->b_flags & B_BUSY) {
                                        bp->b_flags |= B_WANTED;
                                        sleep((caddr_t)bp, PRIBIO);
                                        bp->b_flags &= ~B_WANTED;
                                }
                                bp->b_flags |= B_BUSY;
                                splx(ps);
                                bp->b_dev = dev;
                                bp->b_un.b_addr =(caddr_t)dev;


                                errco = 0;
                                if (rfsdcmd(&PB, device->md_mc, unit)) {
                                        printf("cfioctl: pending bksp failed\n")
                                        errco = EIO;
                                }
                                /* Wakeup any process sleeping on this buffer */
                                bp->b_flags &= ~B_BUSY;
                                if (bp->b_flags & B_WANTED) {
                                        wakeup((caddr_t)bp);
                                        bp->b_flags &= ~B_WANTED;
                                }
                                if (errco) {
                                        safefree((caddr_t)iomem);
                                        iomem_count--;
                                        if(iomem_wanted) {
                                                iomem_wanted = 0;
                                                wakeup((caddr_t)&iomem_wanted);
                                        }
                                        return(errco);
                                }
                                uflagoff(uonEOF, unit, MINER);
                                bzero((char *)&PB, sizeof(PARMBLK));
                                PB.id = (dword)&rfsdbuf[unit];
                                PB.targetid = rfdinfo[MINER].tag_id;
                        }
                }
                switch (mtop->mt_op) {
                   case MTRETEN:
```



```
                        PB.scdb.cmd = SC_LOAD;
                        PB.scdb.byte4 = RETEN|LOAD;
                        break;
                case MTWEOF:
                        PB.scdb.cmd = SC_WFM;
                        PB.scdb.byte2 = (mtop->mt_count >> 16) & 0xFF;
                        PB.scdb.byte3 = (mtop->mt_count >> 8) & 0xFF;
                        PB.scdb.byte4 = (mtop->mt_count) & 0xFF;
                        /* allow use of "short" filemarks */
                        if ((options & EXABYTE) && rfsd_ex_write_short_fm)
                                PB.scdb.byte5 = 0x80;
                        break;
                case MTFSF:
                        if (mtop->mt_count == 0)
                           goto out;
                        if (uflagtst(uonEOF, unit, MINER) &&
                        (options & EXABYTE)) {
                                uflagoff(uonEOF, unit, MINER);
                                if (--mtop->mt_count == 0) {
                                        safefree((caddr_t)iomem);
                                        iomem_count--;
                                        if(iomem_wanted) {
                                                iomem_wanted = 0;
                                                wakeup((caddr_t)&iomem_wanted);
                                        }
                                        return(0);
                                }
                        }

                        PB.scdb.cmd = SC_SPACE;
                        PB.scdb.byte1 |= SFM;
                        PB.scdb.byte2 = (mtop->mt_count>>16) & 0xFF;
                        PB.scdb.byte3 = (mtop->mt_count>>8) & 0xFF;
                        PB.scdb.byte4 = (mtop->mt_count) & 0xFF;
                        break;
                case MTFSR:
                        if (mtop->mt_count == 0)
                           goto out;
                        PB.scdb.cmd = SC_SPACE;
                        PB.scdb.byte1 |= BLOCK;
                        PB.scdb.byte2 = (mtop->mt_count>>16) & 0xFF;
                        PB.scdb.byte3 = (mtop->mt_count>>8) & 0xFF;
                        PB.scdb.byte4 = (mtop->mt_count) & 0xFF;
                        break;
                case MTBSR:
                        if (mtop->mt_count == 0)
                           goto out;
                        PB.scdb.cmd = SC_SPACE;
                        PB.scdb.byte1 |= BLOCK;
                        /* was wrong.. bksp uses forward with negative count */
                        /* this was added */
                        mtop->mt_count = -mtop->mt_count;

                        PB.scdb.byte2 = (mtop->mt_count>>16) & 0xFF;
                        PB.scdb.byte3 = (mtop->mt_count>>8) & 0xFF;
                        PB.scdb.byte4 = (mtop->mt_count) & 0xFF;
```



```
                        /*  was wrong.. bksp uses forward with negative count */
                        /* this was added */
                        mtop->mt_count = -mtop->mt_count;

                        break;
                case MTBSF:
                        if (mtop->mt_count == 0)
                           goto out;
                        if (uflagtst(uonEOF, unit, MINER) &&
                        (options & EXABYTE)) {
                                uflagoff(uonEOF, unit, MINER);
                                if (++mtop->mt_count == 0) {
out:
                                        safefree((caddr_t)iomem);
                                        iomem_count--;
                                        if(iomem_wanted) {
                                                iomem_wanted = 0;
                                                wakeup((caddr_t)&iomem_wanted);
                                        }
                                        return(0);
                                }
                        }
                        PB.scdb.cmd = SC_SPACE;
                        PB.scdb.byte1 |= SFM;
                        /* was wrong.. bksp uses forward with negative count */
                        mtop->mt_count = -mtop->mt_count;

                        PB.scdb.byte2 = (mtop->mt_count>>16) & 0xFF;
                        PB.scdb.byte3 = (mtop->mt_count>>8) & 0xFF;
                        PB.scdb.byte4 = (mtop->mt_count) & 0xFF;
                        /* was wrong.. bksp uses forward with negative count */
                        mtop->mt_count = -mtop->mt_count;

                        break;

                case MTOFFL:
                        uflagoff(uonEOF, unit, MINER);
                        if (options & EXABYTE)
                                rfsd_errlog(dev);
                        goto UNLOAD;
                        break;

                case MTREW:
                        /* these commands clear EOF read, but not reported */
                        uflagoff(uonEOF, unit, MINER);
                        PB.scdb.cmd = SC_REWIND;
                        uflagon(ugeneral, unit, MINER);
                        if (options & EXABYTE)
                                rfsd_errlog(dev);
                        break;
                case MTNOP:
                        safefree((caddr_t)iomem);
                        iomem_count--;
                        if(iomem_wanted) {
                                iomem_wanted = 0;
                                wakeup((caddr_t)&iomem_wanted);
```

```
                        }
                        return(0);
                        break;
                    case MTERASE:
                        /* these commands clear EOF read, but not reported */
                        uflagoff(uonEOF, unit, MINER);
                        PB.scdb.cmd = SC_ERASE; /* erase */
                        PB.scdb.byte1 |= 1;     /* erase to EOT */
                        break;

                }
                /* these commands clear EOF read, but not reported */
                uflagoff(uonEOF, unit, MINER);
                break;
        }

        PB.targetid = rfdinfo[MINER].tag_id;
        PB.scdb.byte1 |= rfdinfo[MINER].log_unit << 5;

        /* Execute rest of the commands */
        switch (utype[unit]) {
            case DIR_ACC:
            case WORM:
            case RDONLYDIR_ACC:
                switch (cmd) {
                    case DKIOCGPART:    /* Get partition info */
                        if (PB.targetid == 0xFE) {      /* Floppy */
                                PB.scdb.cmd = SC_RDCAP;
                                PB.count    = sizeof(read_cap);
                                PB.addrmod = VME_ADD_MOD;
                                PB.vmeaddr = RF_ADDR(iomem);
                        } else {
                                safefree((caddr_t)iomem);
                                iomem_count--;
                                if(iomem_wanted) {
                                        iomem_wanted = 0;
                                        wakeup((caddr_t)&iomem_wanted);
                                }
                                /* just copy what we have */
                                *(struct dk_map *)addr=partitions[unit][partno];
                                return(0);
                        }
                        break;

                    case DKIOCSPART:    /* Set partition info */
                        safefree((caddr_t)iomem);
                        iomem_count--;
                        if(iomem_wanted) {
                                iomem_wanted = 0;
                                wakeup((caddr_t)&iomem_wanted);
                        }
                        /* change our copy */
                        if (PB.targetid == 0xFE)
                                partitions[unit][0] = *(struct dk_map *)addr;
                        else
                                partitions[unit][partno] = *(struct dk_map *)add
```



```
                        return(0);

                    case RFIOCVFY:              /* verify */
                        vreq = (struct dk_vfy *) addr;
                        PB.count = 14;
                        PB.scdb.cmd = SC_VERIFY;
                        PB.scdb.byte2 = BYTE3(vreq->dkv_blkno);
                        PB.scdb.byte3 = BYTE2(vreq->dkv_blkno);
                        PB.scdb.byte4 = BYTE1(vreq->dkv_blkno);

                        PB.scdb.byte5 = BYTE0(vreq->dkv_blkno);
                        PB.scdb.byte7 = BYTE1(vreq->dkv_nblk);
                        PB.scdb.byte8 = BYTE0(vreq->dkv_nblk);
                        break;
                    case RFIOCMAP:              /* Map request */
                        mreq = (struct dk_mapr *) addr;
                        deflp = (struct defect_list *) iomem;
                        /* Wait for the buffer header to get free */
                        ps = splx(pritospl(cfcontroller_info[ctlr]->mc_intpri));
                        bp = &rfsdbuf[unit];
                        buf_free(dev, bp);
                        splx(ps);
                        bp->b_dev = dev;
                        bp->b_un.b_addr = (caddr_t)dev;

                        errco = 0;

                        PB.count = 14;
                        PB.addrmod = VME_ADD_MOD;
                        PB.vmeaddr = RF_ADDR(deflp);
                        PB.scdb.cmd = SC_REASSIGN;
                        deflp->dll = 4;
                        deflp->resv0 = deflp->resv1 = 0;
                        while (!errco && mreq->dkm_nblk) {
                                deflp->lba = mreq->dkm_fblk;
                                if (rfsdcmd(&PB, device->md_mc, unit)) {
                                        DEBUG(0x204, printf("cfioctl: operation
                                        errco = EIO;
                                }
                                mreq->dkm_fblk++;
                                mreq->dkm_nblk--;
                        }
                        /* Wakeup any process sleeping on this buffer */
                        wakeprocess(bp);
                        safefree((caddr_t)iomem);
                        iomem_count--;
                        if(iomem_wanted) {
                                iomem_wanted = 0;
                                wakeup((caddr_t)&iomem_wanted);
                        }
                        return(errco);
                    case RFIOCRDDEF:            /* read defect list */
#ifdef notyet
                        /*deflst = (def_list *)addr;*/
                        /*if (utype[unit] != DIR_ACC) {
                                safefree((caddr_t)iomem);
```



```
                                iomem_count--;
                                if(iomem_wanted) {
                                        iomem_wanted = 0;
                                        wakeup((caddr_t)&iomem_wanted);
                                }
                                return(EINVAL);
                        }*/
#endif
                        PB.scdb.cmd = SC_DEFLIST;
                        PB.scdb.byte2 = 0x18;
                        /*PB.scdb.byte2 = deflst->list[0];*/
                        /*PB.scdb.byte7 = BYTE1(MEMSIZE);
                        PB.scdb.byte8 = BYTE0(MEMSIZE);*/
                        PB.scdb.byte7 = 0x04;
                        PB.scdb.byte8 = 0x00;
                        PB.count = MEMSIZE;
                        PB.addrmod = VME_ADD_MOD;
                        PB.vmeaddr = RF_ADDR(iomem);
                        break;

                    case RFIOCFMT:              /* format */
                        fmt = (format *)addr;
                        if (utype[unit] != DIR_ACC) {
                                safefree((caddr_t)iomem);
                                iomem_count--;
                                if(iomem_wanted) {
                                        iomem_wanted = 0;
                                        wakeup((caddr_t)&iomem_wanted);
                                }
                                return(EINVAL);
                        }
                        /* Wait for the buffer header to get free */
                        ps = splx(pritospl(cfcontroller_info[ctlr]->mc_intpri));
                        bp = &rfsdbuf[unit];
                        buf_free(dev, bp);
                        splx(ps);
                        bp->b_dev = dev;
                        bp->b_un.b_addr = (caddr_t)dev;
                        /* If def_lst_type is true, user wants to include
                         * defect list information during format
                         */
                        if(fmt->def_lst_type)
                        {
                                PB.scdb.cmd = SC_DEFLIST;
                                PB.scdb.byte7 = BYTE1(MEMSIZE);
                                PB.scdb.byte8 = BYTE0(MEMSIZE);
                                PB.count = MEMSIZE;
                                PB.addrmod = VME_ADD_MOD;
                                PB.vmeaddr = RF_ADDR(iomem);
                                errco = 0;
                                for(i = 0; i < 3; i++)
                                {
                                        PB.scdb.byte2 = (fmt->def_lst_type | def
                                        if(!(rfsdcmd(&PB,device->md_mc,unit)))
                                        {
                                                /* this means it worked! */
```



```
                                                def_lst = (def_list *)iomem;
                                                def_list_byte = def_lst->list_by
                                                def_lst->list_byte = 0;
                                                good_deflist = 1;
                                                break;
                                        }
                                }
                                if(!good_deflist)
                                {
                                        DEBUG(0x204, printf("cfioctl: operation
                                        errco = EIO;
                                        /* Wakeup any process sleeping on
                                         * this buffer */
                                        wakeprocess(bp);
                                        safefree((caddr_t)iomem);
                                        iomem_count--;
                                        if(iomem_wanted) {
                                                iomem_wanted = 0;
                                                wakeup((caddr_t)&iomem_wanted);
                                        }
                                        return(errco);

                                }
                        }

                        bzero((char *)&PB, sizeof(PARMBLK));
                        PB.targetid = rfdinfo[MINER].tag_id;
                        PB.scdb.cmd = SC_FORMAT;
                        PB.scdb.byte1 = (bunit << 5 | def_list_byte);
                        PB.scdb.byte3 = BYTE1(fmt->interleave);
                        PB.scdb.byte4 = BYTE0(fmt->interleave);
                        if(fmt->def_lst_type)
                                PB.flags = VALID | DAT | DIR;
                        PB.count = MEMSIZE;
                        PB.addrmod = VME_ADD_MOD;
                        PB.vmeaddr = RF_ADDR(iomem);

                        errco = 0;
                        if(rfsdcmd(&PB,device->md_mc,unit))
                        {
                                DEBUG(0x204, printf("cfioctl: operation failed\n
                                errco = EIO;
                        }

                        /* Wakeup any process sleeping on
                         * this buffer */
                        wakeprocess(bp);
                        safefree((caddr_t)iomem);
                        iomem_count--;
                        if(iomem_wanted) {
                                iomem_wanted = 0;
                                wakeup((caddr_t)&iomem_wanted);
                        }
                        return(errco);
                        break;
```

```
                case RFIOCRDCAP:            /* Read capacity */
                    PB.scdb.cmd = SC_RDCAP;
                    PB.scdb.byte2 = 0;
                    PB.scdb.byte3 = 0;
                    PB.scdb.byte4 = 0;
                    PB.scdb.byte5 = 0;
                    PB.scdb.byte8 = 0;
                    PB.count    = sizeof(read_cap);
                    PB.addrmod = VME_ADD_MOD;
                    PB.vmeaddr = RF_ADDR(iomem);
                    break;

                default:
                    safefree((caddr_t)iomem);
                    iomem_count--;
                    if(iomem_wanted) {
                            iomem_wanted = 0;
                            wakeup((caddr_t)&iomem_wanted);
                    }
                    return (EINVAL);
            }
            break;

        case SEQ_ACC:
            switch (cmd) {
                case DKIOCGPART:    /* Get partition info */
                    safefree((caddr_t)iomem);
                    iomem_count--;
                    if(iomem_wanted) {
                            iomem_wanted = 0;
                            wakeup((caddr_t)&iomem_wanted);
                    }
                    /* just copy what we have */
                    *(struct dk_map *)addr = partitions[unit][partno];
                    return(0);
                case RFIOCFMT:              /* format */
                    PB.scdb.cmd = SC_FORMAT;
                    break;
                case MTIOCTOP:
                    /* our work is already done above -- pass through here
                    */
                    break;
NOP:                default:
                    safefree((caddr_t)iomem);
                    iomem_count--;
                    if(iomem_wanted) {
                            iomem_wanted = 0;
                            wakeup((caddr_t)&iomem_wanted);
                    }
                    return(EINVAL);
            }
            break;
    }

exec:
    /* Wait for the buffer header to get free */
```



```
    ps = splx(pritospl(cfcontroller_info[ctlr]->mc_intpri));
    bp = &rfsdbuf[unit];
    buf_free(dev, bp);
    splx(ps);
    bp->b_dev = dev;
    bp->b_un.b_addr = (caddr_t)dev;

    errco = 0;
    if (rfsdcmd(&PB, device->md_mc, unit)) {
            DEBUG(0x204, printf("cfioctl: operation failed\n");)
            errco = EIO;
            if (PB.scdb.cmd == SC_VERIFY) {
                    vreq->dkv_badblk = bp->b_resid;
                    vreq->dkv_error = MISCOMPARE;
                    errco = 0;
            }
    }

    /**** commands  that return data ****/
    if (!errco) {
            switch (cmd) {
                case RFIOCRDCAP:
                    *(read_cap *)addr = *(read_cap *)iomem;
                    break;
                case DKIOCGPART:    /* Only should do this for floppies. */
                    rc = (read_cap *) iomem;
                    record_info[device->md_unit].nblk =
                            (rc->nblk[0] << 24) |
                            (rc->nblk[1] << 16) |
                            (rc->nblk[2] << 8) |
                            (rc->nblk[3]);
                    record_info[device->md_unit].nblk++;
                    xpart.dkl_cylno = 0;
                    xpart.dkl_nblk = record_info[device->md_unit].nblk;
                    *(struct dk_map *)addr = xpart;
                    break;
                case RFIOCMDSEN:
                    ms = (mode_sel *) iomem;
                    *(mode_sel *)addr = *ms;
                    break;
                case RFIOCMDSEL:
                    *(mode_sel *)addr = *ms;
                    break;
                case RFIOCMDSEN3:
                    ms = (mode_sel *) iomem;
                    offset = (ms->blk_des_len - 8);
                    *pg3 = *(page_3 *) &(ms->vend_uniq[offset]);
                    *(page_3 *)addr = *pg3;
                    break;
                case RFIOCMDSEN4:
                    ms = (mode_sel *) iomem;
                    offset = (ms->blk_des_len - 8);
                    *pg4 = *(page_4 *) &(ms->vend_uniq[offset]);
                    *(page_4 *)addr = *pg4;
                    break;
                case RFIOCINQ:
```



```
                    *(inq_data *)addr = *inqdat;
                    break;
                case RFIOCRDDEF:
                    *(def_list *)addr = *(def_list *)iomem;
                    break;
            }
     }

    /* Wakeup any process sleeping on this buffer */
    wakeprocess(bp);
    safefree((caddr_t)iomem);
    iomem_count--;
    if(iomem_wanted) {
            iomem_wanted = 0;
            wakeup((caddr_t)&iomem_wanted);
    }
    return(errco);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             cfsize                                       *
*                                                                            *
*       Calling Sequence:       cfsize(dev)                                  *
*                                                                            *
*       Called by:              UNIX I/O system                              *
*                                                                            *
*       Calling Parameters:     dev - major and minor device number          *
*                                                                            *
*       Local Variables:        device - pointer to device information       *
*                                                                            *
*       Calls Subroutines:      None                                         *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*               This routine returns the size of the requested partition in  *
*               blocks.                                                      *
*                                                                            *
*****************************************************************************/

daddr_t
cfsize(dev)
dev_t dev;
{
        int MINER;
        int unit;
        int partno;
        REGISTER struct mb_device *device;

        DEBUG(0x1, printf("cfsize:\n");)

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        partno = rfdinfo[MINER].partition;

        if(rfdinfo[MINER].dev_index >= NCF){
                DEBUG(0x1, printf("device not configured!\n");)
                return(-1);
        }

        device = cfdrive_info[unit];

        if (device == NULL || device->md_alive == 0) {
                DEBUG(0x1, printf("device not alive\n");)
                return(-1);
        }
        if (rfdinfo[MINER].tag_id == 0xFE) {
                return(partitions[device->md_unit][0].dkl_nblk);

        } else {
                return(partitions[device->md_unit][partno].dkl_nblk);
        }
}
```

```
/*****************************************************************************
*                                                                            *
*       Subroutine:     cfprint                                              *
*                                                                            *
*       Calling Sequence:       cfprint(str,dev)                             *
*                                                                            *
*       Called By:                                                           *
*                                                                            *
*       Calling Parameters:                                                  *
*               str:    string to be printed                                 *
*               dev:    device number                                        *
*       Local Variables:                                                     *
*                                                                            *
*       Calls Subroutines:      None.                                        *
*                                                                            *
*       Public/Global Variables:        None.                                *
*                                                                            *
*       Description:                                                         *
*               Print out an error message. Called from Kernel.              *
*                                                                            *
*****************************************************************************/

cfprint(dev, str)
char *str;
{
      printf( "RF3500: %s, drive %d\n", str, dev&0177);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             sglcmd                                       *
*                                                                            *
*       Calling Sequence:       sglcmd()                                     *
*                                                                            *
*       Called by:              cfattach                                     *
*                                                                            *
*       Calling Parameters:     xpb - extended RF 3500 parameter block       *
*                                                                            *
*       Local Variables:        pb - address of the extended parameter block *
*                               rfreg - pointer to RF 3500 hardware ports    *
*                               timer - loop counter                         *
*                               msw, lsw - MSW and LSW of a long word        *
*                                                                            *
*       Calls Subroutines:      None                                         *
*                                                                            *
*       Public/Global Variables: xpb - extended RF 3500 parameter block      *
*                                                                            *
*       Description:                                                         *
*               A type 0 command is sent to the RF 3500.  The address of the *
*               extended parameter block is written to the address buffer port *
*               and a 0 is written to the channel attention port.  The status *
*               block is then polled until the command completes.            *
*                                                                            *
*****************************************************************************/

static
sglcmd(xpb, ctlr)
REGISTER EXTPB *xpb;
REGISTER struct mb_ctlr *ctlr;
{
        REGISTER RF35REG *rfreg = (RF35REG *) ctlr->mc_addr;
        register int waitfor;
        REGISTER dword  pb;
        REGISTER word   lsw, msw;
        REGISTER long   timer;

        /* Clear the status block */
        bzero((char *)&xpb->sb, sizeof(STATBLK));

        /* Wait until we can write to address buffer port */
        timer = 0L;
        waitfor = SWAB(bsybit[ctlr->mc_ctlr]);
        while ((rfreg->status & waitfor) != waitfor)
#ifdef TOOLONG
                if (++timer > TOOLONG) {
                        printf("sglcmd: timeout ... BSY\n");
                        printf("sglcmd: status port = %x\n", rfreg->status);
                        blkpr(xpb, sizeof (EXTPB));
                        return(1);
                }
#else TOOLONG
                ;
#endif TOOLONG
```



```
        /* Tell the controller our system characteristics */
        /* Write the control and address modifier to the address buffer port */
        rfreg->addrbuf = SWAB(CNTRL << 8 | VME_ADD_MOD);

        /* convert address of xpb to a dword for shifting */
        pb = RF_ADDR(xpb);
        msw = SWAB(pb >> 16 & 0xffff);
        lsw = SWAB(pb & 0xffff);

        DEBUG(0x800, printf("sglcmd: pb=%x msw=%x lsw=%x\n", pb, msw, lsw);)

        /* write the msw and lsw of the pb to the address buffer port */
        rfreg->addrbuf = msw;
        rfreg->addrbuf = lsw;

        /* Get the controllers attention */
        rfreg->attention = 0;
        bsybit[ctlr->mc_ctlr] ^= 1;     /* Toggle the bit */

        /* Wait for the command to complete */
        DEBUG(0x801,
                printf("sglcmd: Waiting for ST_CC\n");
                printf("sglcmd: status port = %x\n", SWAB(rfreg->status));
                blkpr(xpb, sizeof(EXTPB));
        )

        timer = 0;
        while ((xpb->sb.flags & ST_CC) != ST_CC) {
                if (++timer > MAXWAIT) {
                        printf("RF3500: sglcmd: timeout ... no ST_CC\n");
                        printf("RF3500: sglcmd: status port = %x\n", SWAB(rfreg-
                        blkpr(&xpb->sb, sizeof(STATBLK));
                        return(1);
                }
        }

        if ((xpb->sb.flags & ST_ERR) == ST_ERR) {
                if(rfsddbg) {
                        printf("RF3500: sglcmd error 0x%x, on cmd 0x%x\n",
                                xpb->sb.error, xpb->pb.scdb.cmd);
                        if (xpb->sb.error < EE_FRMERR)
                            printf("    (%s)\n", cfcntrlerr[xpb->sb.error]);
                        blkpr(&xpb->sb, sizeof(STATBLK));
                }

                if (rfsddbg)
                        blkpr(&xpb->sb, sizeof(STATBLK));
                if (xpb->sb.error == EE_SCSISELTO)
                        return(ENXIO);
                if ((xpb->sb.scsistat & STATMASK) == CHECK_COND) {
                        if ((xpb->sb.scsiflags & SENSEMASK) == UNIT_ATTEN)
                                return(0);
                        if (xpb->sb.infob3 == 0x42)
                                return(ENXIO);
                }
                return(1);
```

```
        }
        return(0);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             rfdump                                       *
*                                                                            *
*       Calling Sequence:       rfdump(dev, addr, blkno, nblk)               *
*                                                                            *
*       Called by:              The UNIX panic/dump routines                 *
*                                                                            *
*       Calling Parameters:     dev - major and minor device of swap         *
*                               addr - starting address of click to dump     *
*                               blkno - starting block at which to dump it   *
*                               nblk - number of disk blocks to dump         *
*                                                                            *
*       Local Variables:        xpb - extended paramter block for write      *
*                               pb - point to paramters in above             *
*                               unit - unit number (drive number)            *
*                               device - system device description           *
*                                                                            *
*       Calls Subroutines:      None                                         *
*                                                                            *
*       Public/Global Variables:DVMA                                         *
*                                                                            *
*       Description:                                                         *
*               Called for each core click of the system which paniced.      *
*               Writes out the mapped page to the device at the block        *
*               specified.                                                   *
*                                                                            *
*****************************************************************************/

cfdump(dev, addr, blkno, nblk)
dev_t dev;
caddr_t addr;
daddr_t blkno, nblk;
{
        REGISTER EXTPB  xpb;    /* extended parameter block */
        REGISTER PARMBLK *pb;
        register int MINER;
        register int unit;
        register int part;
        REGISTER struct mb_device *device;
        register int nsectors, blklen;
        REGISTER byte btarget, bunit;
        REGISTER struct dk_geom *gp;

        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        part = rfdinfo[MINER].partition;
        device = cfdrive_info[unit];

        /* does the unit make sense and is the drive alive */
        if (unit > NCF || device == NULL || !device->md_alive) {
                return(ENXIO);
        }
```



```
        /* check to see if we are asking for a blk past the end */
        if(blkno > partitions[unit][part].dkl_nblk ||
            (blkno + nblk) > partitions[unit][part].dkl_nblk) {
                return(EINVAL);
        }

        /* initialize extended parameter block - disable interrupts */
        bzero(&xpb, sizeof (EXTPB));

        /* Calculate some of the inputs... */
        gp = &geometry[unit];
        btarget = rfdinfo[MINER].tag_id;
        bunit = rfdinfo[MINER].log_unit;
        blklen = record_info[unit].bsize;
        blkno += (partitions[unit][part].dkl_cylno * gp->dkg_nsect * gp->dkg_nhe
        nsectors = nblk * DEV_BSIZE;    /* Really bytes */
        if (blklen)
                nsectors /= blklen;

        /* Setup the write command */
        pb = (PARMBLK *)&xpb.pb;
        bzero((char *)pb, sizeof(PARMBLK));
        pb->id = 0;
        pb->targetid = btarget;
        pb->addrmod  = VME_ADD_MOD;
        pb->vmeaddr  = RF_ADDR(addr-DVMA);
        pb->count    = nblk * DEV_BSIZE;
        pb->scdb.cmd = SC_WRITE;
        if (utype[unit] == SEQ_ACC) {
                pb->scdb.byte1 = (bunit << 5) | (BYTE2(blkno) & 0x1F);
                if (blklen)
                        pb->scdb.byte1 |= 1;    /* Fixed */
                pb->scdb.byte2 = BYTE2(nsectors);
                pb->scdb.byte3 = BYTE1(nsectors);
                pb->scdb.byte4 = BYTE0(nsectors);
        } else {
                pb->scdb.byte1 = (bunit << 5) | (BYTE2(blkno) & 0x1F);
                pb->scdb.byte2 = BYTE1(blkno);
                pb->scdb.byte3 = BYTE0(blkno);
                pb->scdb.byte4 = nsectors;
        }

        /* issue the write command and check for errors */
        if (sglcmd(&xpb, device->md_mc)) {
                printf("cfdump: write failed from block %d for %d blocks\n",
                        blkno, nblk);
                return(EIO);
        }
        return(0);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             rfsdcmd                                      *
*                                                                            *
*       Calling Sequence:       rfsdcmd(pb, ctlr, unit)                      *
*                                                                            *
*       Called by:              cfopen, cfclose, cfioctl                     *
*                                                                            *
*       Calling Parameters:     pb - RF 3500 parameter block                 *
*                                                                            *
*       Local Variables:        bp - pointer to our local buffer header      *
*                               rfreg - pointer to RF 3500 hardware ports    *
*                                                                            *
*       Calls Subroutines:      None                                         *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*               A type 1 command is sent to the RF 3500.                     *
*                                                                            *
*****************************************************************************/

rfsdcmd(pb, ctlr, unit)
PARMBLK *pb;
REGISTER struct mb_ctlr *ctlr;
register int unit;
{
        REGISTER RF35REG *rfreg = (RF35REG *) ctlr->mc_addr;
        register int ps;
        REGISTER byte error;

#ifdef STANDALONE
        static EXTPB *xpb = NULL;       /* extended paramter block for
                                           executing commands */

        if (xpb == NULL)
                xpb = (EXTPB *)safealloc(sizeof(EXTPB));
        bzero(xpb, sizeof(EXTPB));
        bcopy(pb, xpb, sizeof(PARMBLK));
        return(sglcmd(xpb, ctlr));
#else
        DEBUG(0x900, printf("rfsdcmd:\n");)

        ps = splx(pritospl(ctlr->mc_intpri));

        pb->id = (dword)&rfsdbuf[unit];

        /* Stuff the parameter block in the command list
         * and issue an attention to the controller
         */
#if NPBMASK
        while (((cmdq[ctlr->mc_ctlr]->pbin + 1) & NPBMASK) == cmdq[ctlr->mc_ctlr
#else
        while ((cmdq[ctlr->mc_ctlr]->pbin + 1) % NPB == cmdq[ctlr->mc_ctlr]->pbo
#endif NPBMASK
```

```
        {
                pb_wanted[ctlr->mc_ctlr] = 1;
                DEBUG(0x5, printf("rfsdcmd: command list fill\n");)
                sleep((caddr_t)&pb_wanted[ctlr->mc_ctlr], PRIBIO);
        }
        cmdq[ctlr->mc_ctlr]->pblist[cmdq[ctlr->mc_ctlr]->pbin] = *pb;
#if NPBMASK
        cmdq[ctlr->mc_ctlr]->pbin = (cmdq[ctlr->mc_ctlr]->pbin + 1) & NPBMASK;
#else
        cmdq[ctlr->mc_ctlr]->pbin = (cmdq[ctlr->mc_ctlr]->pbin + 1) % NPB;
#endif NPBMASK

        DEBUG(0x900,
                blkpr(pb, sizeof(PARMBLK));
                printf("rfsdcmd: issuing attention\n");
        )

        /* stuff the SCSI cdb command into b_resid for intr proc */
        rfsdbuf[unit].b_resid = pb->scdb.cmd;
        rfsdbuf[unit].b_error = 0;      /* nobody seems to clear the error! */
        rfsdbuf[unit].b_flags &= ~B_ERROR;

        rfreg->attention = SWAB(1);

        /* Wait for the command to get finished */
        sleep(&rfsdbuf[unit].b_un.b_addr, PRIBIO);

        error = rfsdbuf[unit].b_error;

        /* check B_ERROR as well --ghg */
        if (error == 0 && (rfsdbuf[unit].b_flags & B_ERROR))
                error = EIO;    /* something better ? */
        splx(ps);
        return(error);
#endif STANDALONE
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             blkpr                                        *
*                                                                            *
*       Calling Sequence:       blkpr(ca,n)                                  *
*                                                                            *
*       Called by:              this driver                                  *
*                                                                            *
*       Calling Parameters:     ca - pointer to structure                    *
*                               n  - number of bytes in the structure        *
*                                                                            *
*       Local Variables:        None                                         *
*                                                                            *
*       Calls Subroutines:      None                                         *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*               Displays command or status block.                            *
*                                                                            *
*****************************************************************************/
blkpr(ca,n)
unsigned char *ca;
byte n;
{
        REGISTER byte i;

        for (i=0; i<n; i++) {
                printf(" %x",ca[i]);
                if (!((i+1)%16))
                        printf("\n");
        }
        if (i % 16) printf("\n");       /* append trailing CRLF. */
}
/*****************************************************************************
*                                                                            *
*       Subroutine:             safealloc,safefree                           *
*                                                                            *
*       Calling Sequence:       safealloc(size)                              *
*                               safefree(ptr)                                *
*                                                                            *
*       Called by:              rfprobe, rfslave, rfattach, rfstrategy and   *
*                               rfintr just for scatter/gather header, and   *
*                               various rfioctl support routines             *
*                                                                            *
*       Calling Parameters:     size - the size of DVMA needed               *
*                               ptr - pointer to a DVMA chunk to release     *
*                                                                            *
*       Calls Subroutines:      safealloc: rmalloc                           *
*                               safefree: rmfree                             *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*               Called to allocate and return iopbmap resources to insulate us *
```



```
*               from unaligned iopbmap allocations.  To just protect ourselves,*
*               we could ask for (size + 3) bytes, but that might cause      *
*               problems for other drivers, so we will ask for (size + 4)    *
*               bytes, thereby leaving iopbmap aligned or unaligned as we    *
*               found it.                                                    *
*                                                                            *
******************************************************************************/

struct sapool {
        caddr_t start;
        long    len;
        };

static caddr_t
safealloc(size)
int size;
{
    REGISTER char *chunk;
    REGISTER struct sapool *alchunk;
    register int chsize;

    chsize = size + sizeof (struct sapool) + sizeof (int);
    if ((chunk = (caddr_t)rmalloc(iopbmap,chsize)) == NULL)
    {
                return(chunk);
    }
    alchunk = (struct sapool *)((int)(chunk+sizeof (int)) & ~(sizeof (int)-1));
    alchunk->start = chunk;
    alchunk->len = chsize;
    return ((char *)(alchunk + 1));
} /* safealloc */

static void
safefree(chunk)
caddr_t chunk;
{
    REGISTER struct sapool *alchunk;

    alchunk = ((struct sapool *)chunk) - 1;
    rmfree (iopbmap, alchunk->len, alchunk->start);
}

/*****************************************************************************
*                                                                            *
*       Subroutine:             rfsd_errlog                                  *
*                                                                            *
*       Calling Sequence:       rfsd_errlog(dev)                             *
*                                                                            *
*                                                                            *
*       Called by:              cfread, cfwrite, cfioctl, cfclose            *
*                                                                            *
*       Calling Parameters:     dev                                          *
*                                                                            *
*       Calls Subroutines:      rfsdcmd                                      *
*                                                                            *
*       Public/Global Variables:None                                         *
```



```
*                                                                            *
*       Description:                                                         *
*               Optional Exabyte error log statistics.  Exabyte (since       *
*       ~October '87) maintains a 3 byte error count for error logging       *
*       purposes.  It can be read by doing a request sense command, and      *
*       looking at sense bytes 16, 17, 18.  For writes, this is the number   *
*       of blocks (1 Kbyte on tape) which were rewritten (rewrites). For     *
*       reads, this is the number of ECC corrections performed.              *
*                                                                            *
*       The drive clears the errlog counters whenever:                       *
*       1) A tape is loaded                                                  *
*       2) A reading to writing transistion takes place                      *
*       3) A writing to reading transistion takes place                      *
*       4) A sense request is made with vendor byte 5, bit 7 set in the cdb  *
*          (cdb->vu_v57)                                                     *
*                                                                            *
*       For now, we just issue a simple 1 line printf, whenever something    *
*       happens which would cause the drive to clear its error counters (1-3 *
*       above). A REW or OFFLINE command also forces the stats.  Hitting the *
*       drive eject button causes the stats to be lost. Also have to keep    *
*       track of bytes written/read, so we can come up with an error         *
*       percentage (Exabyte folks like to just deal with the percentages).   *
*       Transferrs, not a multiple of 1Kbyte, are rounded up to the next whole *
*       Kbyte for the errlog stats.. since this is what gets occupied on the *
*       tape.                                                                *
*       To enable/disable errlog stats, use adb to patch kernel variable     *
*       "rfsd_pr_errlog" (an int) to one of the below values (hex):          *
*                                                                            *
*       0x0  - do not print anything                                         *
*       0x1  - uprintf - goes to "/dev/tty" (current window or login session) *
*       0x2  - printf - goes to "/dev/console".                              *
*       0x8  - misc errlog debug info                                        *
*                                                                            *
*       bits can be OR'd together (default is 0x3).                          *
*       We are seeing good tapes and drives have a write error percentage    *
*       around 0.5% to 2.0% or less, and less than 1% retries for reading.   *
*                                                                            *
******************************************************************************/

#define EXABYTE_SENSE_LEN       26

rfsd_errlog(dev)
register dev_t dev;
{
        register int MINER;
        register int unit;
        REGISTER struct mb_device *device;
        REGISTER struct buf *bp;
        REGISTER PARMBLK PB;
        unsigned char *ucp;
        REGISTER char *iomem;
        int l, left;
        int errco = 0;
        byte bunit;
        byte btarget;
        char *s;
```

```
        int tot_retries;
        register int percent;

        /*
         * We first run a sense command....
         * need to dig up buffers, iomem, etc, first..
         */
        MINER = minor(dev);
        unit = rfdinfo[MINER].dev_index;
        device = cfdrive_info[unit];
        btarget = rfdinfo[MINER].tag_id;
        bunit = rfdinfo[MINER].log_unit;

        /* allocate space for iomem */
        /* probably no need to word align this, since we will do */
        /* only byte ref to it */
        while ((iomem = (char *)rmalloc(iopbmap, MEMSIZE)) == (char *)NULL) {
                if(iomem_count) {
                        iomem_wanted = 1;
                        sleep((caddr_t)iomem_wanted, PRIBIO);
                } else {
                        DEBUG(100, printf("rfsd_errlog: Can't allocate memory fo
                        return(ENOMEM);
                }
        }
        iomem_count++;
        DEBUG(2, printf("rfsd_errlog: allocated iomem at %x\n", iomem);)

        /* Get the buffer for commands... */
        l = splx(pritospl(device->md_mc->mc_intpri));
        bp = &rfsdbuf[unit];
        while (bp->b_flags & B_BUSY) {
                bp->b_flags |= B_WANTED;
                sleep(bp, PRIBIO);
                bp->b_flags &= ~B_WANTED;
        }
        bp->b_flags |= B_BUSY;
        splx(l);
        bp->b_dev = dev;
        bp->b_un.b_addr = iomem;

        bzero((char *)&PB, sizeof(PARMBLK));
        PB.targetid = btarget;
        PB.addrmod = VME_ADD_MOD;
        PB.vmeaddr = RF_ADDR(iomem);
        PB.count = EXABYTE_SENSE_LEN;
        PB.scdb.cmd = SC_SENSE;
        PB.scdb.byte1 = bunit << 5;
        PB.scdb.byte4 = EXABYTE_SENSE_LEN;
        PB.scdb.byte5 = 0x80;   /* clear counters */
        bzero((char *) iomem, EXABYTE_SENSE_LEN);
        DEBUG(0x201,printf("rfsd_errlog: Issuing 'sense' command to unit %d\n",
        if (rfsdcmd(&PB, device->md_mc, unit)) {
                DEBUG(0x201, printf("rfsd_errlog: sense command failed\n", bp->b
                errco = ENXIO;
        }
```



```
        if (errco) {
                bp->b_flags &= ~B_BUSY;
                if (bp->b_flags & B_WANTED) {
                        wakeup((caddr_t)bp);
                }
                rmfree(iopbmap, MEMSIZE, (caddr_t)iomem);
                iomem_count--;
                if(iomem_wanted) {
                        iomem_wanted = 0;
                        wakeup((caddr_t)&iomem_wanted);
                }
                return(errco);
        }

        s = "";
        if (lastop[unit] != UNDEF && kb_xfer[unit]) {
                ucp = (unsigned char *) iomem;
                if (lastop[unit] == READ)
                        s = "read";
                if (lastop[unit] == WRITE)
                        s = "write";
                tot_retries = (ucp[16]<<16) |
                   (ucp[17]<<8) | ucp[18];

                percent = (tot_retries * 1000) / kb_xfer[unit];
                left = (ucp[23]<<16) |
                   (ucp[24]<<8) | ucp[25];
                if (left & 0x00800000)
                        left |= 0xFF000000;     /* sxt */
                left -= 0x500; /* tape used for the BOT */
                left >>= 10;    /*   Kbytes -> Mbytes */
                if (rfsd_pr_errlog & 01)
printf("st%d: %d %s retries out of %d Kbytes (%d.%d%%), %d Mbytes until LEOT\n",
   unit, tot_retries, s, kb_xfer[unit],percent/10,percent%10,left);
                if (rfsd_pr_errlog & 02)
                {
#ifndef STANDALONE
uprintf("st%d: %d %s retries out of %d Kbytes (%d.%d%%), %d Mbytes until LEOT\n"
   unit, tot_retries, s, kb_xfer[unit],percent/10,percent%10,left);
#endif
                }
        }

        lastop[unit] = UNDEF;
        bp->b_flags &= ~B_BUSY;
        if (bp->b_flags & B_WANTED) {
                wakeup((caddr_t)bp);
        }
        rmfree(iopbmap, MEMSIZE, (caddr_t)iomem);
        iomem_count--;
        if(iomem_wanted) {
                iomem_wanted = 0;
                wakeup((caddr_t)&iomem_wanted);
        }
        kb_xfer[unit] = 0;
```



```
        lastop[unit] = UNDEF;
        return(0);
}
#endif NCF
```



```
/*****************************************************************************
******************************************************************************
****                                                                      ****
****                    Copyright (C) Ciprico Incorporated 1987.          ****
****                                                                      ****
****                            Ciprico Incorporated                      ****
****                              2955 Xenium Lane                        ****
****                             Plymouth, MN 55441                       ****
****                               (612) 559-2034                         ****
****                                                                      ****
****    Module Name:    cs35err.h                                         ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus     ****
****    Module Rev:     $Revision$                          ****
****    Date:           $Date$              ****
****                                                                      ****
****    Subroutines:                                                      ****
****                                                                      ****
****    Description:                                                      ****
****            This is an include file for Rimfire 3500 driver. It defines ****
****            the errors returned by RF-3500 controller. And there is an  ****
****            array of one line messages for the errors.                ****
****                                                                      ****
******************************************************************************
******************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                              Revision History                              *
*                                                                            *
*****************************************************************************

    Revision    Date            Author
        Description of Change
----------------------------------------------------------------------------

    1.1         10/17/86        Umesh Gupta
        Pre-Release.
    1.1         08/20/87        D. A. Dickey
        Initial A Release.

$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35err.h_v  $

      Rev 2.0   07 Apr 1989 13:38:58   JMartin
   Initial revision.

    2.0         04/05/89        J. Martin
        Made changes for multiple controllers.  See cs35.c for details.
$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523.  See cs35.c
           revision history for details.

*****************************************************************************/
```



```
#ifndef lint
static  char Sccsh2id[] = "@(#)cs35err.h (75222006)  Copyright Ciprico Inc. 1987
#endif


/* One line messages for the controller errors */

char    *cfcntrlerr[] = {
/*     0 */     "No error",
/*     1 */     "Invalid command",
/*     2 */     "Bad unit number specified",
/*   3-A */     "","","","","","","","",
/*     B */     "Reserved field not zero",
/*   C-D */     "","",
/*     E */     "Command list stopped",
/*     F */     "Bad command list size field",
/*    10 */     "Bad command list number to start/stop",
/*    11 */     "List state wrong for start/stop command",
/* 12-13 */     "","",
/*    14 */     "Software bus timeout error",
/*    15 */     "Bus error reported by control chip",
/* 16-1D */     "","","","","","","","",
/*    1E */     "SCSI select timeout",
/*    1F */     "SCSI disconnect timeout",
/*    20 */     "SCSI parity error",
/*    21 */     "Unexpected disconnect",
/*    22 */     "Undefined or uninterpretable SCSI error",
/*    23 */     "SCSI device-specific error",
/* 24-2F */     "","","","","","","","","","","","","","","","",
/* 30-3F */     "","","","","","","","","","","","","","","","",
/*    40 */     "Bad gap size found during format",
/*    41 */     "Command complete timeout",
/*    42 */     "Floppy disk unit not ready",
/*    43 */     "Seek fault on floppy drive",
/*    44 */     "CRC error in iD",
/*    45 */     "Write fault on drive",
/*    46 */     "Data CRC error",
/*    47 */     "Sector not found",
/*    48 */     "Floppy dusk data lost from 2793",
/*    49 */     "Data underrun during operation",
/* 4A-4F */     "","","","","","",
/* 50-5F */     "","","","","","","","","","","","","","","","",
/* 60-6F */     "","","","","","","","","","","","","","","","",
/* 70-7F */     "","","","","","","","","","","","","","","",""
};
#define EE_SCSISELTO    0x1E            /* SCSI select timeout */
#define EE_SCSIERR      0x23            /* SCSI returned bad status */
#define EE_FRMERR       0x80            /* 80H and above are firmware errors */


/* SCSI status (only relevent bits are defined) */
#define CHECK_COND      0x02            /* Check condition */
#define DEVICE_BUSY     0x08            /* Busy */
#define RESV_CONF       0x18            /* Reservation conflict */
```



```
#define STATMASK        0x1E            /* Status mask */

#define FM              0x80            /* file mark */
#define ILI             0x20            /* illegal length indicator */
#define EOM             0x40            /* end of media */

/* One line message for SCSI errors */
char    *sensekey[] = {
/*     0 */     "No sense",
/*     1 */     "Recovered error",
/*     2 */     "Not ready",
/*     3 */     "Medium error",
/*     4 */     "Hardware error",
/*     5 */     "Illegal request",
/*     6 */     "Unit attention",
/*     7 */     "Data protect",
/*     8 */     "Blank check",
/*     9 */     "Vender unique",
/*     A */     "Copy aborted",
/*     B */     "Aborted command",
/*     C */     "Equal",
/*     D */     "Volume overflow",
/*     E */     "Miscompare",
/*     F */     "Reserved"
};
#define NOSENSE         0x00            /* No sense? */
#define RECOVERED       0x01            /* Recovered error */
#define UNIT_ATTEN      0x06            /* Unit attention */
#define PROTECTED       0x07            /* Data Protected */
#define BLANK           0x08            /* Blank check */
#define MISCOMPARE      0x0E            /* Data Verify failed */
#define SENSEMASK       0x0F            /* Sense mask */

#define AV              0x80            /* Valid information */
```



```
/*****************************************************************************
******************************************************************************
****                                                                      ****
****                    Copyright 1987 Ciprico Incorporated.              ****
****                                                                      ****
****                            Ciprico Incorporated                      ****
****                              2955 Xenium Lane                        ****
****                             Plymouth, MN 55441                       ****
****                               (612) 559-2034                         ****
****                                                                      ****
****    Module Name:    cs35flp.h                                         ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus     ****
****    Module Rev:     $Revision$                                  ****
****    Date:           $Date$              ****
****                                                                      ****
****    Subroutines:                                                      ****
****                                                                      ****
****    Description:                                                      ****
****            This is an include file for Rimfire 3500 driver. It defines ****
****            the floppy configuration for release 1.5 of the driver or ****
****            later.                                                    ****
****                                                                      ****
******************************************************************************
******************************************************************************/
```

```
/*****************************************************************************
*                                                                           *
*                              Revision History                             *
*                                                                           *
*****************************************************************************

    Revision    Date            Author
        Description of Change
---------------------------------------------------------------------------
$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35flp.h_v  $

      Rev 2.0   07 Apr 1989 13:39:52   JMartin
   Initial revision.

    2.0         04/05/89        Jody Martin
        New file at release of 2.0 for floppy configuration.  See cs35.c for
        details of other changes.

$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523.  See cs35.c
           revision history for details.

*****************************************************************************/
```



```
/*****************************************************************************
*****************************************************************************
****                                                                     ****
****                    Copyright (C) Ciprico Incorporated 1987.         ****
****                                                                     ****
****                            Ciprico Incorporated                     ****
****                              2955 Xenium Lane                       ****
****                             Plymouth, MN 55441                      ****
****                               (612) 559-2034                        ****
****                                                                     ****
****    Module Name:    cs35if.h                                         ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus    ****
****    Module Rev:     $Revision$                                  ****
****    Date:           $Date$              ****
****                                                                     ****
****    Subroutines:                                                     ****
****                                                                     ****
****    Description:                                                     ****
****            This is an include file for the Rimfire 3500 Driver. It  ****
****            defines the software interface to the board, defining the ****
****            various bit level patterns used by the  RF-3500.   Also  ****
****            defined are the various structures used by the driver    ****
****            and the controller board.                                ****
****                                                                     ****
*****************************************************************************
*****************************************************************************/
```



```
#ifndef lint
static  char Sccsh5id[] = "@(#)cs35flp.h  (75222006)  Copyright 1989, Ciprico In
#endif

/* floppy media types */

#define FLM_200SSSD     (0x05)     /* 8" (200mm) SS/SD, 48tpi */
#define FLM_200DSSD     (0x6)      /* 8" (200mm) DS/SD, 48tpi */
#define FLM_200SSDD     (0x9)      /* 8" (200mm) SS/DD, 48tpi */
#define FLM_200DSDD     (0xa)      /* 8" (200mm) DS/DD, 48tpi */
#define FLM_130SSSD48   (0xd)      /* 5.25" (130mm) SS/SD 48tpi */
#define FLM_130DSDD48   (0x12)     /* 5.25" (130mm) DS/DD 48tpi */
#define FLM_130DSDD96   (0x16)     /* 5.25" (130mm) DS/DD 96tpi */
#define FLM_130DSQD96   (0x1a)     /* 5.25" (130mm) DS/QD 96tpi */
#define FLM_90DSDD135   (0x1e)     /* 3.5" (90mm) DS/DD 135tpi */

#define FL_MEDIA(x)     ((x)&0x3f)  /* get media type from field */

/* floppy sector sizes */

#define FLSS_128        (0)        /* 128 byte sectors */
#define FLSS_256        (0x40)     /* 256 byte sectors */
#define FLSS_512        (0x80)     /* 512 byte sectors */
#define FLSS_1024       (0xc0)     /* 1024 byte sectors */
#define FLSS_2048       (0x100)    /* 2048 byte sectors */

#define FL_SECTS(x)     (128 << ( ((x) >> 6) &~0x3f8))  /* floppy sector size */
#define FL_SECTC(x)     ((x) & 0x1c)           /* floppy sector size code */

/* floppy sectors per track */

#define FLSPT_5         (5<<10)    /* 5 sectors/track */
#define FLSPT_8         (8<<10)    /* 8 sectors/track */
#define FLSPT_9         (9<<10)    /* 9 sectors/track */
#define FLSPT_10        (10<<10)   /* 10 sectors/track */
#define FLSPT_15        (15<<10)   /* 15 sectors/track */
#define FLSPT_16        (16<<10)   /* 16 sectors/track */
#define FLSPT_18        (18<<10)   /* 18 sectors/track */
#define FLSPT_31        (31<<10)   /* 31 sectors/track */

#define FL_SPT(x)       (((x) & ~0xc000) >> 10)  /* floppy sectors per track */

/* floppy single/double step option (reading 48tpi floppies on 96tip drive)*/
/* should not format with this set, and really shouldn't write either,
   though it is often nice to be able to do so */
#define FL_DSTP         (1<<15)

#define FL_RDSTP(x)         ((x) >> 15)
```



```
/*****************************************************************************
*                                                                           *
*                              Revision History                             *
*                                                                           *
*****************************************************************************

    Revision    Date            Author
        Description of Change
---------------------------------------------------------------------------

    1.0         07/14/86        Umesh Gupta
        Initial Release.

    1.1         02/02/87        D. A. Dickey
        Initial B Release.
    1.1         08/20/87        D. A. Dickey
        Initial A Release.
    1.4         07/13/88        J. K. Martin
        1. Added SunOS3 and SunOS4 defines for the two sun operating
           systems.
$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35if.h_v  $

      Rev 2.0   07 Apr 1989 14:05:16   JMartin
   Initial revision.

    2.0         04/05/89        J. Martin
        1. Added changes for multiple controllers. See cs35.c for details.

$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523.  See cs35.c
           revision history for details.

*****************************************************************************/
```

```
/*
        The definitions within this file are as follows:

        - Type definitions for 8, 16, and 32 bit quantities.
        - Macros for busting the minor device number.
        - Definition for special control registers on RF-3500.
        - Definition for standard parameter block and its variations.
        - Definition for memory based structures used by the controller.
        - Driver internal structures.
*/
```



```
#ifndef lint
static  char Sccshlid[] = "@(#)cs35if.h (75222006)  Copyright 1987, 1989 Ciprico
#endif

/*                          N O T E
 * You must define exactly one of the following system symbols.
 */

/*#define SunOS3        /* define for SunOS 3.2, 3.4, and 3.5 systems   */
/*#define SunOS4        /* define for SunOS 4.0 systems                 */

#if !defined (SunOS3) && !defined (SunOS4)
        char c = OSLEVEL;
#endif

#if defined (SunOS3) && defined (SunOS4)
        char c = OSERROR;
#endif

/*   First a few defines to make life a little easier */
#define byte    unsigned char           /*  8 bit quantity */
#define word    unsigned short          /* 16 bit quantity */
#define dword   unsigned int            /* 32 bit quantity */

/* These two defines should be the same */
#define LOGDSK  8               /* Partitions on physical disk */
#define FLPFMT  8               /* Number of different floppy formats */

#define MEMSIZE 1024            /* DMA memory size */

#define BYTE0(n)        ((byte)(n) & 0xFF)
#define BYTE1(n)        BYTE0((dword)(n)>>8)
#define BYTE2(n)        BYTE0((dword)(n)>>16)
#define BYTE3(n)        BYTE0((dword)(n)>>24)

/* UNIX minor device number
 *
 *      The minor device number is broken down as follows:
 *
 *         7     6     5     4     3     2     1     0
 *      +-----+-----+-----+-----+-----+-----+-----+-----+
 *      |            Unit no.           |      Lunit      |
 *      +-----+-----+-----+-----+-----+-----+-----+-----+
 *
 *      Unit number     Specifies UNIT unit (0-31), this in turn is mapped
 *                      into a specific target id which is interpreted from
 *                      the md_slave value in each mb_device struct.
 *      Lunit           Specifies UNIX logical disk partition (0-7), or
 *                      Specifies format for floppy, or
 *                      Specifies no rewind on tape
 *
 */


/* Macros to manipulate the minor device number */
```



```
#define UNIT(dev)       ((dev) >> 3 & 0x1F)     /* Unit number */

#define LUNIT(dev)      ((dev) & 0x7)           /* Logical unit */
#define PARTITION(dev)  ((dev) & 0x7)           /* Partition on disk */

#define SWAB(x) ((x) & 0xFFFF)          /* no swap for motorola processor */
```



```
/* Hardware Ports
 *
 *      The Rimfire 3500 is seen as a 512 byte area in the 16 bit
 *      address space.  There are four registers in this space,
 *      each on an 8 byte boundary.  The registers are defined below.
 */

/* Hardware ports */
typedef struct {
        word    resv;           /* msw address */
        word    addrbuf;        /* Address Buffer Port (see below) */
        word    space1[3];
        word    attention;      /* Channel Attention Port */
        word    space2[3];
        word    status;         /* Board Status Port */
        word    space3[3];
        word    reset;          /* Controller Reset Port */
} RF35REG;

/* Address Buffer Port
 *      Requires three writes in the order provided below:
 *
 *      +---------------+---------------+
 *      |    Control    | AM bits for PB|
 *      +---------------+---------------+
 *      |     PB address:  16 MSW       |
 *      +-------------------------------+
 *      |     PB address:  16 LSW       |
 *      +-------------------------------+
 */

/* Control Field bit masks */
#define CTRLBSW 0x01            /* byte swap control, 0=no swap, 1=swap */
#define CTRLWSW 0x02            /* word swap control, 0=no swap, 1=swap */
#define CTRLWID 0x04            /* width of data transfer, 0=16, 1=32 bit */
#define CTRLSET 0x80            /* apply these controls this command */

/* Status Port bit masks */
#define STATRST 0xFE            /* status of a board reset */
#define STATBSY 0x01            /* status of the Address Buffer Port */
#define STATRDY 0x02            /* board is ready */
#define RESETDONE 0x02          /* reset has completed */
#define STATCTYPE 0xFF00        /* Mask to grab controller type. */
#define STATRF3500 0x0200       /* A Rimfire 3500 board ID */

/* Standard Parameter Block
 *
 *      +-----------------------------------------------------+
 *      |                  Command Identifier                 |
 *      +-----------------------+-------------+---------------+
 *      | Reserved    |   Flags    |  Addr Mod   |  Target ID  |
 *      +-----------------------+-------------+---------------+
 *      |                  VME Memory Address                 |
 *      +-----------------------------------------------------+
 *      |                    Transfer Count                   |
```

```
 *      +-------------+-------------+-------------+-------------+
 *      |      0      |      1      |      2      |      3      |
 *      +-------------+----         -----+-------------+-------------+
 *      |      4      | 5  SCSI Command Block  6  |      7      |
 *      +-------------+----         -----+-------------+-------------+
 *      |      8      |      9      |      10     |      11     |
 *      +-------------+-------------+-------------+-------------+
 */
typedef struct {
        dword   id;             /* unique command identifier */
        byte    resv;
        byte    flags;
        byte    addrmod;        /* address modifier used to access VME memory */
        byte    targetid;       /* SCSI target ID */
        dword   vmeaddr;        /* VME address to read from or write to */
        dword   count;          /* Transfer count */
        struct  scdb {          /* SCSI command descriptor block */
                byte    cmd;            /* SCSI command */
                byte    byte1;          /* 3 high bits are logical unit # */
                byte    byte2;
                byte    byte3;          /* LSB sectors */
                byte    byte4;          /* Number of sectors */
                byte    byte5;          /* Control field */
                byte    byte6;
                byte    byte7;
                byte    byte8;
                byte    byte9;
                byte    byte10;
                byte    byte11;
        }scdb;
} PARMBLK;
/* Bits in flags field */
#define SGO     0x01            /* Scatter/gather operation */
#define DAT     0x02            /* Data transmitted in this operation */
#define DIR     0x04            /* Direction of data transfer 1=to the target */
#define IRS     0x08            /* Inhibit request sense */
#define VALID   0x80            /* Valid */

/* Status Block
 *
 *      +-------------+-------------+-------------+-------------+
 *      |                   Command Identifier                  |
 *      +-------------+-------------+-------------+-------------+
 *      |      0      | SCSI status |    Error    |    Flags    |
 *      +-------------+-------------+-------------+-------------+
 *      |                   Extra Information                   |
 *      +-------------------------------------------------------+
 *      |                   Extra Information                   |
 *      +-------------------------------------------------------+
 */
typedef struct {
        dword   id;             /* command identifier generating status */
        byte    zero;
        byte    scsistat;       /* SCSI status, device specific */
        byte    error;          /* RF3500 specific error */
```



```
        byte    flags;          /* indicates type of status */
        byte    class;          /* class/code */
        byte    segment ;       /* Segment */
        byte    scsiflags;      /* SCSI flags */
        byte    infob3;         /* Information byte 3 */
        byte    infob4;         /* Information byte 4 */
        byte    infob5;         /* Information byte 5 */
        byte    infob6;         /* Information byte 6 */
        byte    exlength;       /* Extra length */
} STATBLK;

/* Status Block Flags Field bit masks */
#define ST_RTY  0x20            /* Retry required */
#define ST_ERR  0x40            /* Error, check error code */
#define ST_CC   0x80            /* Command complete, last status block */
#define ST_CON  0x04            /* This status block is continued from */
                                /* the previous one */
#define ST_SOFT 0x02            /* A soft error occured (scsi sense key 01) */

/* Status Block Class bit masks */
#define ADVALID 0x80            /* Address field information valid */

/* Extended Parameter Block
 *
 *      +-------------------------------------------------------+
 *      |                                                       |
 *      +--                                                   --+
 *      |                                                       |
 *      +--                                                   --+
 *      |                                                       |
 *      +--                                                   --+
 *      |               Standard Parameter Block                |
 *      +--                                                   --+
 *      |                                                       |
 *      +--                                                   --+
 *      |                                                       |
 *      +--                                                   --+
 *      |                                                       |
 *      +-------------------------+-----------------------------+
 *      |         Reserved        |          Interrupt          |
 *      +-------------------------+-----------------------------+
 *      |                   Reserved, MUST be 0                 |
 *      +-------------------------------------------------------+
 *      |                                                       |
 *      +--                                                   --+
 *      |                Standard Status Block                  |
 *      +--                                                   --+
 *      |                                                       |
 *      +-------------------------------------------------------+
 */
typedef struct {
        PARMBLK pb;             /* standard parameter block */
        word    resv0;          /* reserved */
        word    intr;           /* interrupt vector and level */
        dword   resv1;          /* reserved, must be 0 (zero) */
        STATBLK sb;             /* status block */
```



```
} EXTPB;

/* Command list format
 *
 *      +-------------------------------------------------------+
 *      |              Parameter block IN pointer               |
 *      +-------------------------------------------------------+
 *      |              Parameter block OUT pointer              |
 *      +-------------------------------------------------------+
 *      |                Status block IN pointer                |
 *      +-------------------------------------------------------+
 *      |                Status block OUT pointer               |
 *      +-------------------------------------------------------+
 *      |              Parameter block area size                |
 *      +-------------------------------------------------------+
 *      |                Status block area size                 |
 *      +-------------------------------------------------------+
 *      |                                                       |
 *      |                                                       |
 *      |                  Parameter block area                 |
 *      |                                                       |
 *      |                                                       |
 *      +-------------------------------------------------------+
 *      |                                                       |
 *      |                                                       |
 *      |                   Status block area                   |
 *      |                                                       |
 *      |                                                       |
 *      +-------------------------------------------------------+
 */
#define NPB     0x20            /* Max. number of parameter blocks */
#define NSB     0x20            /* Max. number of status blocks */

/* Masks for wrapping around IN/OUT pointers - do NOT define these if
 * the corresponding NxB value is not a power of 2.
 */
#define NPBMASK 0x1F            /* mask for valid parameter block index */
#define NSBMASK 0x1F            /* mask for valid status block index */

typedef struct {
        dword   pbin;           /* parameter block in pointer */
        dword   pbout;          /* parameter block out pointer */
        dword   sbin;           /* status block in pointer */
        dword   sbout;          /* status block out pointer */
        dword   pbsize;         /* parameter block area size */
        dword   sbsize;         /* status block area size */
        dword   resv[2];        /* reserved */
        PARMBLK pblist[NPB];    /* parameter blocks */
        STATBLK sblist[NSB];    /* status blocks */
} CMDLIST;

/* RF 3500 commands for Targets ID 0xFF */
#define C_STARTCL       1       /* start command list */
#define C_STOPCL        2       /* stop command list */
#define C_IDENTIFY      5       /* identify */
#define C_STATS         6       /* board statistics */
```



```
#define C_OPTION        7       /* general options */
#define C_UNITOPT       8       /* init I/O control group */
#define C_DIAGNOSTIC    9       /* board self-test command */


/*************************************************************************
 *      Command #1 - Start Command List
 *************************************************************************/

/* Alternate Parameter Block - Setup command list */
typedef struct {
        dword   id;             /* unique command identifier */
        word    resv1;          /* reserved */
        byte    addrmod;        /* address modifier used to access VME memory */
        byte    targetid;       /* SCSI target ID, set to 0xFF */
        dword   memaddr;        /* command list memory address */
        word    resv0;          /* reserved */
        word    intr;           /* interrupt vector and level */
        byte    command;        /* command to execute */
        byte    resv2[3];
        dword   resv3[2];       /* reserved */
} SETUPPB;

/*************************************************************************
 *      Command #2 - Stop Command List
 *************************************************************************/

/* Alternate Parameter Block - Stop command list */
typedef struct {
        dword   id;             /* unique command identifier */
        byte    resv1[3];       /* reserved */
        byte    targetid;       /* SCSI target ID, set to 0xFF */
        dword   resv2[2];
        byte    command;        /* command to execute */
        byte    resv3[3];
        dword   resv4[2];
} STOPPB;

/*************************************************************************
 *      Command #5 - Identify
 *************************************************************************/

/* Alternate Status Block for Identify command */
typedef struct {
        dword   id;             /* identifier */
        byte    fwrev;          /* firmware revision level */
        byte    engrev;         /* engineering revision level */
        byte    error;          /* error code */
        byte    flags;          /* flags */
        byte    optflags;       /* optional flags (see below) */
        byte    day;            /* day the firmware was generated */
        byte    month;          /* month the firmware was generated */
        byte    year;           /* year the firmware was generated */
        dword   resv;           /* reserved */
} RETID;
```

```
/* Opt flags field in Identify Status Block */
#define FDO     0x01            /* set   = Floppy Disk Option present */

/************************************************************************
 *      Command #6 - Board Statistics
 ************************************************************************/

/* Alternate Parameter Block - Board Statistics */
typedef struct {
        dword   id;             /* unique command identifier */
} STATPB;

/************************************************************************
 *      Command #7 - General Options
 ************************************************************************/

/* Alternate Parameter Block - General Options command */
typedef struct {
        dword   id;             /* identifier */
        byte    optflags;       /* optional flags (see below) */
        byte    throttle;       /* bus throttle */
        byte    ownid;          /* SCSI target ID */
        byte    targetid;       /* SCSI target ID, set to 0xFF */
        dword   resv0[2];       /* reserved */
        byte    command;        /* command to execute */
        byte    resv3[3];       /* status block interrupts (see below) */
        dword   resv4[2];
} GOPTPB;

/* Opt flags field in parameter block */
#define DIS     0x01            /* set   = Allow disconnect/reselect in SCSI ope
#define PAR     0x02            /* set   = Report parity errors on SCSI bus */
#define BMT     0x04            /* set   = Use Block Mode Transfers to/from Memo
/* NOTE: Suns DO NOT support the BMT option. */

/************************************************************************
 *      Command #8 - Unit options
 ************************************************************************/

/* Alternate Parameter Block - Unit Options */
typedef struct {
        dword   id;             /* command identifier */
        word    distimeout;     /* SCSI disconnect timeout, .1 sec unit */
        byte    unitid;         /* SCSI target id */
        byte    targetid;       /* SCSI target id, set to 0xFF */
        word    seltimeout;     /* SCSI select timeout, millisec units */
        byte    retrycntrl;     /* Retry control (see below) */
        byte    retrylimit;     /* Number of retries if retry enabled */
        word    resv2;          /* Reserved */
        byte    reqlength;      /* Extended sense count for req sense cmd */
        byte    uflags;         /* Unit flags (see below) */
        byte    command;        /* Command to execute (0x8) */
        byte    resv4[3];       /* Reserved */
        dword   resv5[2];       /* Reserved */
} UOPPB;
```



```
/* Retry control bit fields */
#define SCINT   0x01            /* Issue "error" interrupt for each retry */
#define RCISB   0x02            /* Issue status block for each retry */
#define RCRPE   0x04            /* Retry parity error */
#define RCRCE   0x08            /* Retry command errors (SCSI errors) */
#define RCRBE   0x10            /* Retry bus errors (selection timeouts, etc) */

/* Unit flags bit fields */
#define UF_IDI  0x01            /* Inhibit Disconnect */
#define UF_SYN  0x02            /* Synchronous Transfers */
#define UF_IAT  0x04            /* Inhibit ATN Signal */
#define UF_ISE  0x20            /* Don't retry soft errors, but report using  */
                                /* the flag field in the stat blk (ST_SOFT)   */
/************************************************************************
 *      Command #9 - Diagnostic / Self Test
 ************************************************************************/

/* Alternate Parameter Block - Configure disk */
typedef struct {
        dword   id;             /* identifier */
        byte    resv1[3];       /* reserved */
        byte    targetid;       /* SCSI target id, set to 0xFF */
        dword   resv2[2];       /* reserved */
        byte    command;        /* Command to be executed */
        byte    flags;          /* Test flags */
        word    resv3;          /* Reserved */
        dword   resv4[2];       /* Reserved */
} TESTPB;

/* Test flags field in PB */
#define TESTSRT 0x01            /* Static RAM test */
#define TESTPCS 0x02            /* PROM checksum test */

/* Alternate Status Block for Diagnostic / Self Test command */
typedef struct {
        dword   id;             /* identifier */
        byte    zero[2];        /* Set to zero */
        byte    error;          /* error code */
        byte    flags;          /* flags */
        word    erroraddr;      /* Address where error found */
        byte    expected;       /* Pattern expected at error location */
        byte    found;          /* Pattern found at error location */
} RETTEST;



/* SCSI commands */
#define SC_READY        0x00    /* Test unit ready */
#define SC_REZERO       0x01    /* Rezero unit */
#define SC_REWIND       0x01    /* Rewind */
#define SC_SENSE        0x03    /* Request sense */
#define SC_FORMAT       0x04    /* Format unit */
#define SC_RDBLKLIM     0x05    /* Read Block Limits - Sequential devices only *
#define SC_REASSIGN     0x07    /* Reassign blocks - Map Sector(s) */
#define SC_READ         0x08    /* Read */
#define SC_WRITE        0x0A    /* Write */
```



```
#define SC_SEEK         0x0B    /* Seek */
#define SC_WFM          0x10    /* Write filemark */
#define SC_SPACE        0x11    /* Space blocks, filemarks, EOT */
#define SC_INQUIRY      0x12    /* Inquiry */
#define SC_SELMODE      0x15    /* Mode select */
#define SC_RESERVE      0x16    /* Reserve */
#define SC_RELEASE      0x17    /* Release */
#define SC_ERASE        0x19    /* Erase */
#define SC_SENMODE      0x1A    /* Mode sense */
#define SC_LOAD         0x1B    /* Load/Unload & Start/Stop Device */
#define SC_RDCAP        0x25    /* Read capacity */
#define SC_VERIFY       0x2F    /* Verify the media */
#define SC_DEFLIST      0x37    /* Read defect list */

/* Put these into byte1 of scdb for the SC_SPACE command to tell it what */
/* to search for. */
#define BLOCK   0x0
#define SFM     0x1
#define SQFM    0x2
#define PEOM    0x3

/* Put these into byte4 of scdb for the SC_LOAD command to tell it what */
/* to do.  These are bit masks...to not put them in is to do the opposite. */
#define LOAD    0x1
#define RETEN   0x2
```



```
/************************************************************
 *       SCSI structures
 ************************************************************/

/* MODE SELECT parameter list */
typedef struct  {
        byte    byte0;          /* Reserved */
        byte    medium_type;    /* Medium type */
        byte    byte2;          /* Reserved */
        byte    blk_des_len;    /* Block descriptor length */
        byte    density_code;   /* Density code */
        byte    nblk[3];        /* Number of blocks (MSB) - (LSB) */
        byte    byte8;          /* Reserved */
        byte    blklen[3];      /* Block length (MSB) - (LSB) */
        byte    vend_uniq[50];  /* Vendor Unique parameter bytes */
} mode_sel;

/* Page 1 for mode select commands on hard drives. */
typedef struct {
        byte    page_code;
        byte    page_length;
        byte    page_1_flags;   /* (See Below) */
        byte    retry_cnt;      /* retry count */
        byte    corr_span;      /* correction span */
        byte    hd_off_cnt;     /* head offset count */
        byte    data_str_cnt;   /* data strobe count */
        byte    recov_time;     /* recovery time limit */
} page_1;

/* Page_1_flags bit field */
#define DCR     0x01            /* Disable Correction */
#define DTE     0x02            /* Disable Transfer on Error */
#define PER     0x04            /* Post Error */
#define EEC     0x08            /* Enable Early Correction */
#define READCON 0x10            /* Read Continuous */
#define XFERBLK 0x20            /* Transfer Block */
#define ARRE    0x40            /* Automatic Read Allocation of Defective Data B
#define AWRE    0x80            /* Automatic Write Reallocation of Defective Dat

/* Page 2 for mode select commands on hard drives. */
typedef struct {
        byte    page_code;
        byte    page_length;
        byte    b_full;         /* Buffer Full Ratio */
        byte    b_empty;        /* Buffer Empty Ratio */
        word    b_inactive;     /* Bus Inactivity Limit */
        word    b_dis_time;     /* Disconnect Time Limit */
        word    b_con_time;     /* Connect Time Limit */
        word    b_reserved;     /* Reserved */
} page_2;

/* Page 3 for mode select commands on hard drives. */
typedef struct {
        byte    page_code;
```

```
        byte    page_length;
        word    trkpzone;       /* Tracks Per Zone */
        word    altspzone;      /* Alternate sectors per zone */
        word    alttpzone;      /* Alternate tracks per zone */
        word    alttpvol;       /* Alternate tracks per volume */
        word    spt;            /* Sectors Per Track */
        word    nbps;           /* Bytes Per Sector */
        word    interleave;     /* Interleave */
        word    trk_skew;       /* Track skew factor */
        word    cyl_skew;       /* Cylinder skew factor */
        byte    dtype[4];       /* Drive type fields (1-3 reserved) */
} page_3;
/* Page 4 for mode select commands on hard drives. */
typedef struct {
        byte    page_code;
        byte    page_length;
        byte    ncyl_b0;        /* Number of cylinders - MSB */
        byte    ncyl_b1;        /* Number of cylinders */
        byte    ncyl_b2;        /* Number of cylinders - LSB */
        byte    nhead;          /* Number of heads */
        byte    scylwp_b0;      /* Starting cylinder of write precomp - MSB */
        byte    scylwp_b1;      /* Starting cylinder of write precomp */
        byte    scylwp_b2;      /* Starting cylinder of write precomp - LSB */
        byte    scylrwc_b0;     /* Starting cylinder of reduced write current -
        byte    scylrwc_b1;     /* Starting cylinder of reduced write current */
        byte    scylrwc_b2;     /* Starting cylinder of reduced write current -
        word    dsr;            /* Drive Step Rate */
        byte    lzc_b0;         /* Landing Zone Cylinder - MSB */
        byte    lzc_b1;         /* Landing Zone Cylinder */
        byte    lzc_b2;         /* Landing Zone Cylinder - LSB */
        byte    resv[3];        /* Reserved */
} page_4;
/* Page 5 for mode select commands on floppies. */
typedef struct {
        byte    page_code;
        byte    page_length;
        word    xfer_rate;      /* Transfer rate */
        byte    nheads;         /* Number of heads */
        byte    spt;            /* Sectors Per Track */
        word    nbps;           /* Bytes Per Sector */
        word    ncyls;          /* Number of Cylinders */
        word    s_wpre;         /* Starting cylinder - Write Precomp */
        word    s_rwc;          /* Starting cylinder - Reduced Write Current */
        word    dsr;            /* Drive Step Rate */
        byte    dspw;           /* Drive Step Pulse Width */
        byte    hd_st_dly;      /* Head Settle Delay */
        byte    on_dly;         /* Motor On Delay */
        byte    off_dly;        /* Motor Off Delay */
        byte    trdy;           /* Drive Provides a True Ready Signal */
        byte    hd_ld_dly;      /* Head Load Delay */
        byte    ssn_s0;         /* Starting Sector #, Side zero */
        byte    ssn_s1;         /* Starting Sector #, Side one */
} page_5;
/* Page 20 for mode select commands on floppies. */
typedef struct {
        byte    page_code;
```



```
        byte    page_length;
        byte    post_index;     /* Post Index Gap */
        byte    inter_sector;   /* Inter Sector Gap */
        byte    tverify;        /* Seek Verification */
        byte    tsteps;         /* Steps Per Track */
        byte    resv0;
        byte    resv1;
} page_20;

/* READ CAPACITY data list */
typedef struct {
        byte    nblk[4];        /* Logical block address */
        byte    blklen[4];      /* Block length */
} read_cap;

/* INQUIRY Data */
typedef struct {
        byte    dtype;
        byte    rmb_dtq;
        byte    version;
        byte    byte3;
        byte    add_len;
        byte    vend_uniq[41];  /* This could be a MAX of 507 */
} inq_data;

/* Read Block Limits Data */
typedef struct {
        byte    byte0;
        byte    mxblklen[3];
        byte    mnblklen[2];
} blk_lim;

/* Defect list for mapping bad blocks */
struct defect_list {
        byte    resv0;
        byte    resv1;
        word    dll;    /* Defect list length */
        dword   lba;    /* Defect logical block address */
};

/* Defect list for reading defect list and formatting */
typedef struct {
        byte    resv0;
        byte    list_byte;
        byte    dll_msb;    /* Defect list length */
        byte    dll_lsb;    /* Defect logical block address */
        byte    list[1020];    /* defect list */
} def_list;

typedef struct {
        dword   interleave;     /* Interleave used while formatting */
        byte    def_lst_type;   /* Defect list type - Grown, Manufacturers */
} format;
```



```
/*************************************************************
 *       Structures used internally by the driver
 *************************************************************/

typedef struct {
        byte    id;             /* SCSI target ID */
        byte    unit;           /* Target unit number for this device */
} target;


/* Device parameters */
typedef struct {
        dword   bsize;          /* Block length */
        dword   nblk;           /* Number of blocks */
} rec_info;

/* Defines the way in which the driver knows which controller has wich
   devices.  Also defines the device and any unit options the device may
   have.  This is a 32 bit word, with 29 bits actually used.  (See xfsdint.h
   for initialization).  This is the structure that defines the minor device
   number set up, as follows:  dev_id = device type (SEQ_ACC, DIR_ACC, etc),
   tag_id = target id of the device, log_unit

        */

struct device_word {
        unsigned dev_index      :  5;   /* unit number on system */
        unsigned dev_id         :  8;   /* device Id number (device type) */
        unsigned tag_id         :  8;   /* devices SCSI target Id number */
        unsigned log_unit       :  3;   /* logical unit number */
        unsigned unit_ops       :  17;  /* devices unit options */
        unsigned partition      :  15;  /* device partion number */
};
```



```
/*****************************************************************************
******************************************************************************
****                                                                      ****
****                    Copyright 1987 Ciprico Incorporated.              ****
****                                                                      ****
****                            Ciprico Incorporated                      ****
****                              2955 Xenium Lane                        ****
****                             Plymouth, MN 55441                       ****
****                               (612) 559-2034                         ****
****                                                                      ****
****    Module Name:    cs35int.h                                         ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus     ****
****    Module Rev:     $Revision$                          ****
****    Date:           $Date$              ****
****                                                                      ****
****    Subroutines:                                                      ****
****                                                                      ****
****    Description:                                                      ****
****            This is an include file for Rimfire 3500 driver. It defines ****
****            the initialization for release 1.5 of the driver or       ****
****            later.                                                    ****
****                                                                      ****
******************************************************************************
******************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                              Revision History                              *
*                                                                            *
*****************************************************************************

    Revision    Date            Author
        Description of Change
-----------------------------------------------------------------------------
$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35int.h_v  $

      Rev 2.0   21 Apr 1989 11:14:30   JMartin
   Initial revision.

    2.0         10/01/88        Jody Martin
        New file at release of 2.0 for new minor device array.  See cs35.c for
        more details of changes.

$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523.  See cs35.c
           revision history for details.

*****************************************************************************/
```



```
/*****************************************************************************
*****************************************************************************
****                                                                     ****
****                    Copyright (C) Ciprico Incorporated 1987.         ****
****                                                                     ****
****                            Ciprico Incorporated                     ****
****                              2955 Xenium Lane                       ****
****                             Plymouth, MN 55441                      ****
****                               (612) 559-2034                        ****
****                                                                     ****
****    Module Name:    cs35io.h                                         ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus    ****
****    Module Rev:     $Revision$                                  ****
****    Date:           $Date$              ****
****                                                                     ****
****    Subroutines:                                                     ****
****                                                                     ****
****    Description:                                                     ****
****            This is an include file for Rimfire 3500 driver. It contains****
****            defines for I/O control commands and structures related  ****
****            to ioctl call.                                           ****
****                                                                     ****
*****************************************************************************
*****************************************************************************/
```



```
#ifndef lint
static  char Sccsh6id[] = "@(#)cs35int.h  (75222006)  Copyright 1987, 1989 Cipri
#endif

/* MAXMINOR is the total amount of minor device numbers or indexes into the */
/* device_word array.                                                       */
#define MAXMINOR        (sizeof(rfdinfo)/sizeof(struct device_word))

/* MAXBOARDS is the total number of 3500 controllers that could be configured
   into the Sun system. */

#define MAXBOARDS       3

/* the following is customer site specific.  The customer would set this
   up to reflect the drives that he has installed in his system.  See cs35flp.h
   for explaination of FLM_??????, FLSS_????, and FLSPT_????? */

#define FLOPPY0       (FLM_130DSDD48|FLSS_512|FLSPT_10)
#define FLOPPY1       (FLM_130DSDD48|FLSS_512|FLSPT_8)
#define FLOPPY2       (FLM_130DSDD48|FLSS_512|FLSPT_9)
#define FLOPPY3       (FLM_130DSDD48|FLSS_512|FLSPT_9)
#define FLOPPY4       (FLM_130DSDD96|FLSS_512|FLSPT_9)
#define FLOPPY5       0
#define FLOPPY6       (FLM_130DSQD96|FLSS_512|FLSPT_15)
#define FLOPPY7       0

#define NOT_USED 0

#ifdef STANDALONE
struct device_word rfdinfo[50];
#else
struct device_word rfdinfo[] = {
/* device       dev        target     logical      unit    partion */
/* index        id           id         unit       opts    number  */
/* ADD NEW ENTRIES HERE */
};
#endif

/* This array may be as deep as needed to fit your system config. of boards
   Also controller boards may have the same SCSI Id as they are talking on
   different SCSI buses.  The boards are located by address for the VME
   bus, or by slot number for MULTIBUS so there is no conflict between
   boards as long as they have differnet        addresses, or slots */

static int coninfo[MAXBOARDS] = {
    6,
    6,
    6,
};
```



```
/*****************************************************************************
*                                                                            *
*                              Revision History                              *
*                                                                            *
*****************************************************************************

    Revision    Date            Author
        Description of Change
-----------------------------------------------------------------------------

    1.1         08/20/87        D. A. Dickey
        Initial A Release.

$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35io.h_v  $

      Rev 2.0   07 Apr 1989 13:39:34   JMartin
   Initial revision.

    2.0         04/05/89        J. Martin
        Changes for multiple controllers.  See cs35.c for details.

$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523.  See cs35.c
           revision history for details.

*****************************************************************************/
```

```
#ifdef lint
static  char Sccsh4id[] = "@(#)cs35io.h (75222006)  Copyright Ciprico Inc. 1987,
#endif


/* Command codes for ioctl() calls */
#define RFIOCFMT        _IOW(r, 6, format)      /* Format drive */
#define RFIOCRDDEF      _IOR(r, 8, def_list)    /* Read defect list */
#define RFIOCGDEBUG     _IOR(r, 9, dword)       /* Get driver debug level */
#define RFIOCSDEBUG     _IOW(r, 10, dword)      /* Set driver debug level */
#define RFIOCGSTAT      _IOR(r, 11, dword)      /* Get RF3500 statistics */
#define RFIOCGDEVID     _IOR(r, 12, dword)      /* Get dev type (SEQ_ACC etc) */
#define RFIOCRESET      _IOW(r, 25, dword)      /* Reset controller */
#define RFIOCGENOPT     _IOR(r, 26, dword)      /* General Options command */
#define RFIOCUNITOPT    _IOW(r, 27, dword)      /* Unit Options command */
#define RFIOCCMDLST     _IOW(r, 28, dword)      /* Start command list */
#define RFIOCMDSEN3     _IOWR(r, 29, page_3)    /* Mode Sense command pg 3 */
#define RFIOCMDSEN4     _IOWR(r, 30, page_4)    /* Mode Sense command pg 4*/
#define RFIOCMDSEN      _IOWR(r, 31, mode_sel)  /* Mode Sense no pages */
#define RFIOCMDSEL      _IOWR(r, 32, mode_sel)  /* Mode Selectnse no pages */
#define RFIOCIDENT      _IOR(r, 67, RETID)      /* Identify */
#define RFIOCRDCAP      _IOR(r, 68, read_cap)   /* Read capacity */
#define RFIOCMAP        _IOW(r, 69, struct dk_mapr)     /* Map sector */
#define RFIOCVFY        _IOWR(r, 70, struct dk_vfy)     /* Verify */
#define RFIOCGTARGET    _IOR(r, 71, target)     /* Who are we talking to? */
#define RFIOCLOAD       _IOW(r, 74, dword)      /* Load/Start the drive */
#define RFIOCUNLOAD     _IOW(r, 75, dword)      /* Unload/Stop the drive */
#define RFIOCANY        _IOWR(r, 76, dword)     /* ANY SCSI command... */
#define RFIOCINQ        _IOWR(r, 77, inq_data)  /* Inquiry command */
#define RFIOCGBLKSIZE   _IOR(r, 126, dword)     /* Get driver block size */
#define RFIOCSBLKSIZE   _IOW(r, 127, dword)     /* Set driver block size */

#define DKC_RF3500      11      /* Another controller type for dkio.h */

/* disk re-map request */
struct dk_mapr {
        daddr_t dkm_fblk;                       /* from block */
        daddr_t dkm_tblk;                       /* to block */
        daddr_t dkm_nblk;                       /* # of blocks */
        u_char  dkm_recover;
        u_char  dkm_error;
};

/* disk verify request */
struct dk_vfy {
        daddr_t dkv_blkno;
        daddr_t dkv_nblk;
        u_char  dkv_error;
        daddr_t dkv_badblk;
};

#define SECTSIZE        (512)
```



```
/*****************************************************************************
******************************************************************************
****                                                                      ****
****                Copyright 1988 Ciprico Incorporated.                  ****
****                                                                      ****
****                      Ciprico Incorporated                            ****
****                        2955 Xenium Lane                              ****
****                       Plymouth, MN 55441                             ****
****                         (612) 559-2034                               ****
****                                                                      ****
****    Module Name:    cs35lib.c                                         ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus.    ****
****    Module Rev:     $Revision$                                  ****
****    Date:           $Date$              ****
****                                                                      ****
****    Subroutines:                                                      ****
****                                                                      ****
****    Description:                                                      ****
****                                                                      ****
******************************************************************************
******************************************************************************/
```



```
/*****************************************************************************
*                                                                           *
*                              Revision History                             *
*                                                                           *
*****************************************************************************

    Revision    Date            Author
        Description of Change
---------------------------------------------------------------------------
$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35lib.c_v  $

     Rev 2.0   07 Apr 1989 13:39:30   JMartin
  Initial revision.

    2.0         04/05/89        J. Martin
        Initial Release of this file, see cs35.c rev history for details.

$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523 as follows:

           - Made changes to support standalone utility.
           - Made many changes to the cs35ut utility, and the driver ioctl
             calls.
           - Made changes to support the new install script.
           - The format disk command now supports including the defect list
             during the disk format.
           - Changed the command codes to match as close as possible to the
             Rimfire 32XX rfutil.
           - Added support for selecting a dummy device to change debug
             value through the utility if you can't open another disk.
           - Added various features to existing commands.

*****************************************************************************


*****************************************************************************
*/
```



```
#ifndef lint
static  char Sccsid[] = "cs35lib.c:  (75222006)  Copyright 1988,1989 Ciprico Inc
#endif

#include "cf.h"

#if NCF > 0
/* Include whatever system include files are needed here. */
#include <sys/param.h>
#include <sys/dir.h>
#include <sys/user.h>
#include <sys/buf.h>
#include <sys/systm.h>
#include <sys/kernel.h>
#include <sys/map.h>
#include <sys/ioctl.h>
#include <sys/file.h>
#include <sys/dk.h>
#include <sys/mtio.h>
#include <sys/uio.h>

#include "../sun/dkio.h"
#include "../sun/dklabel.h"
#include "../machine/psl.h"
#include "../sundev/mbvar.h"

#include "../sundev/cs35if.h"
/*#include "../sundev/cs35io.h"*/
#include "../sundev/cs35prm.h"
#include "../sundev/cs35flp.h"

/*****************************************************************************/
/*  Function buf_free(dev,bp) The interrupt priority must be increased before*/
/*                            calling this routine to lock out the interrupts*/
/*                            while waiting for the buffer to become avail-  */
/*                            able.  If the buffer is busy, the driver will  */
/*                            sleep until it becomes free.                   */
/*****************************************************************************/

buf_free(dev,bp)
dev_t dev;
struct buf *bp;
{

        while(bp->b_flags & B_BUSY)
        {
          bp->b_flags |= B_WANTED;
          sleep((caddr_t)bp, PRIBIO);
          bp->b_flags &= ~B_WANTED;
        }
        bp->b_flags |= B_BUSY;
}
/*****************************************************************************/
/*                                                                           */
```

```
/*   function wakeprocess(bp)                                                */
/*                                                                           */
/*****************************************************************************/
wakeprocess(bp)
struct buf *bp;
{

  bp->b_flags &= ~B_BUSY;
  if (bp->b_flags & B_WANTED)
  {
    wakeup((caddr_t)bp);
    bp->b_flags &= ~B_WANTED;
  }
}
/*****************************************************************************/

/*****************************************************************************
*                                                                            *
* ROUTINE SCSI_RESERVE()                                                     *
*                                                                            *
* PURPOSE: Set up and issue SCSI Reserve device command                      *
*                                                                            *
*****************************************************************************/
SCSI_RESERVE(dev,PB,lrfdinfo,MINER,bp)
struct mb_device *dev;
PARMBLK PB;
struct device_word *lrfdinfo;
register int MINER;
struct buf *bp;
{
        int errco;

        PB.scdb.cmd = SC_RESERVE;
        PB.targetid = lrfdinfo->tag_id;
        PB.scdb.byte1 = lrfdinfo->log_unit << 5;
        PB.flags = VALID;       /* Don't include data phase for this cmd */
        if (rfsdcmd(&PB, dev->md_mc, lrfdinfo->dev_index))
        {
                /*DEBUG(0x201, printf("cfopen: reserve device failed\n");)*/
                errco = ENXIO;
                return(errco);
        }
}

/*****************************************************************************
*                                                                            *
*  ROUTINE SCSI_RELEASE()                                                    *
*                                                                            *
*  PURPOSE Set up and execute SCSI Release device command                    *
*                                                                            *
*****************************************************************************/

SCSI_RELEASE(dev,PB,lrfdinfo,MINER,bp)

struct mb_device *dev;
PARMBLK PB;
```



```
struct device_word *lrfdinfo;
register int MINER;
struct buf *bp;
{
        PB.scdb.cmd = SC_RELEASE;
        PB.targetid   = lrfdinfo->tag_id;
        PB.scdb.byte1 = lrfdinfo->log_unit << 5;
        PB.flags = VALID;       /* Don't include data phase for this cmd */

        /* Release the device */
        if (rfsdcmd(&PB, dev->md_mc, lrfdinfo->dev_index))
                printf("release device %x failed\n", bp->b_dev);
}
#endif NCF
```



```
/*****************************************************************************
******************************************************************************
****                                                                      ****
****                    Copyright (C) Ciprico Incorporated 1987.          ****
****                                                                      ****
****                            Ciprico Incorporated                      ****
****                              2955 Xenium Lane                        ****
****                             Plymouth, MN 55441                       ****
****                               (612) 559-2034                         ****
****                                                                      ****
****    Module Name:    cs35prm.h                                         ****
****    Package:        Rimfire 3500 Driver for UNIX Sun OS on VME bus     ****
****    Module Rev:     $Revision$                                  ****
****    Date:           $Date$              ****
****                                                                      ****
****    Subroutines:                                                      ****
****                                                                      ****
****    Description:                                                      ****
****            This is an include file for Rimfire 3500 driver. It contains****
****            configuration parameters that may change often.           ****
****                                                                      ****
******************************************************************************
*****************************************************************************/
```



```
/*****************************************************************************
*                                                                            *
*                              Revision History                              *
*                                                                            *
*****************************************************************************

    Revision    Date            Author
       Description of Change
-----------------------------------------------------------------------------

    1.1         07/14/86        Umesh Gupta
       Pre-Release.
    1.1         08/20/87        D. A. Dickey
       Initial A Release.
$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35prm.h_v  $

      Rev 2.0   07 Apr 1989 13:39:56   JMartin
   Initial revision.

    2.0         04/05/89        J. Martin
       Added changes for multiple controllers.  See cs35.c for details.

$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523.  See cs35.c
           revision history for details.

*****************************************************************************/
```

```
#ifdef lint
static  char Sccsh3id[] = "@(#)cs35prm.h (75222006)  Copyright Ciprico Inc. 1987
#endif
```



```
/*********************************************
 *      VME things, change these for your system
 *********************************************/

#ifdef sun3     /* 1.2a */
# define VME_ADD_MOD    0x0D            /* VME address modifier for Ext Sup */
#else
# define VME_ADD_MOD    0x3D            /* VME address modifier for Std Sup */
#endif

/* Data transfer control between controller and VME memory */
#define CNTRL           (CTRLSET+CTRLWID)       /* no swap, 32 bit xfer */

#define THROTTLE        8               /* Bus throttle */


/*********************************************
 *      SCSI defines, may change
 *********************************************/

#define SEL_TIMEOUT     10              /* SCSI selection timeout */
#define RFSD_RESET_DELAY        5       /* seconds to delay after SCSI-bus reset

#define PRINT_SB_HEADER 1
#define PRINT_TARGET_ID 0
#define PRINT_RECOV_ERRORS 0
```



```
/* Defines for operations that this driver can do. */
/* These may apply across device types or be device type specific. */
/* These defines appear in the specific mb_device structures as */
/* md_flags for the units. */
#define NOOPN_RDCAP     0x000001    /* Do Not issue Read Capacity at open() */
#define NOOPN_MDSEL     0x000002    /* Do Not issue Mode Select at open() call.
#define ONEFILEMARK     0x000004    /* Tape drive can only write 1 filemark at a
#define GEN_MODE        0x000008    /* Tape drive has "modes" of operations. */
                                    /* Add this if you can only do certain   */
                                    /* commands (i.e., MODE SELECT) when the */
                                    /* drive is in "general" mode            */
                                    /* (in contrast to "read" or "write" mode*/
#define NORESERVE       0x000010    /* Don't do Reserve & Release commands */
#define OS9FLOPPY       0x000040    /* This is an OS9 compatible floppy (sectors
#define SYNCHRONOUS     0x000080    /* This drive is synchronous... */
#define LONGRDYWAIT     0x000100    /* This drive may take a while to pass test
#define EXABYTE         0x000200    /* This is a vendor unique parameter to   ha
#define FIXEDBLK        0x000400    /* Force fixed blk mode if EXABYTE */

#define REQLENLO        0x000800    /* Amount of extended sense bytes expected*/
#define REQLENHI        0x001000    /* when firmware will issue the Request */
/* sense cmd after a check condition status.  REQLENLO and REQLENHI form a  */
/* binary representation of the size of the extended status desired.  Note  */
/* the table below:                                                         */
/****************************************************************************/
/*      REQLENHI        REQLENLO        REQUEST SENSE COUNT                 */
/*          0               0               8(default)                      */
/*          0               1               16                              */
/*          1               0               24                              */
/*          1               1               32                              */
/****************************************************************************/

#define NORETRYSOFT     0x002000    /* Don't retry soft errors, but print  */
                                    /* the status block on the screen */
#define SORTCMB         0x004000    /* Enable Sort and Combine for this device
                                     * Note: tape devices should not use this
                                     * feature.  To use this feature your board
                                     * must have firmware rev 9 or greater for
                                     * the 3500, and any non beta firmware
                                     * for the 3510 controller. */
#define NOREWIND        0x008000    /* This is a no rewind device for tapes*/
#define ONEFM           0x010000    /* This flag terminates tape with 1 EOF
                                       (like sun) instead of two.  Be careful
                                       when using this flag.  If it is desired
                                       to append files to the end of a tape, it
                                       may be hard to determine where that is
                                       as a filemark gets written between files
                                       also.
/****************************************************************************/

/* Manufacturer specific "normal" *ops: */
/* TAPE DEVICES */
#define Archive         (TP_NORMAL)
#define Archivenr       (TP_NORMAL|NOREWIND)
#define Exabyte         (TP_NORMAL|NOOPN_RDCAP|NORESERVE|EXABYTE)
```



```
#define Exabytenr       (TP_NORMAL|NOOPN_RDCAP|NORESERVE|EXABYTE|NOREWIND)
#define ExabyteF        (TP_NORMAL|NOOPN_RDCAP|NORESERVE|EXABYTE|FIXEDBLK)
#define ExabyteFnr      (TP_NORMAL|NOOPN_RDCAP|NORESERVE|EXABYTE|NOREWIND|FIXEDB
#define HPTP            (TP_NORMAL|ONEFM)
#define HPTPnr          (TP_NORMAL|ONEFM|NOREWIND)
#define Kdy9612         (TP_NORMAL|NOOPN_MDSEL|NORESERVE)
#define Kdy9612nr       (TP_NORMAL|NOOPN_MDSEL|NORESERVE|NOREWIND)
#define Patriot         (TP_NORMAL|GEN_MODE|NORESERVE)
#define Patriotnr       (TP_NORMAL|GEN_MODE|NORESERVE|NOREWIND)
#define WANGTEK         (TP_NORMAL|ONEFILEMARK|GEN_MODE)
#define WANGTEKnr       (TP_NORMAL|ONEFILEMARK|GEN_MODE|NOREWIND)

/* DISK DEVICES */
#define Fujitsu         (WD_NORMAL)
#define FujitsuS        (WD_NORMAL|SYNCHRONOUS)
#define HP              (WD_NORMAL)
#define HPS             (WD_NORMAL|SYNCHRONOUS)
#define Micropolis      (WD_NORMAL)
#define MicropolisS     (WD_NORMAL|SYNCHRONOUS)
#define Maxtor          (WD_NORMAL)
#define MaxtorS         (WD_NORMAL|NOOPN_MDSEL|NORESERVE|SYNCHRONOUS)
#define Miniscribe      (WD_NORMAL|NOOPN_MDSEL)
#define Quantum         (WD_NORMAL)
#define Wren            (WD_NORMAL)
#define WrenS           (WD_NORMAL|SYNCHRONOUS)
#define Other           (WD_NORMAL)
#define Generic         (WD_NORMAL)

/* The "normal" unitops: */
#define WD_NORMAL       (0)
#define TP_NORMAL       (0)

/* A byte to tell us what device type the units are...this is the device */
/* type according to the ANSI spec. */
/* Also, this must match the flags field of mb_device. */
/* This means that there is a maximum of 50 devices allowed */
static unsigned char    utype[50];
#define DIR_ACC         0       /* Direct Access device - probably disk */
#define SEQ_ACC         1       /* Sequential Access deive - tape */
#define PRN             2       /* Printer */
#define PROC            3       /* Processor */
#define WORM            4       /* Write Once Read Multiple - optical disk */
#define RDONLYDIR_ACC   5       /* Read Only Direct Access device - optical disk
#define LUN_NOTPRES     0x7F    /* Logical Unit Not Present */
#define FLOPPY          DIR_ACC /* floppy device */
#define DUMMY           0x90    /* Dummy device for use with cs35ut */
static char *type_names[] = {
        "Direct Access",
        "Sequential Access",
        "Printer",
        "Processor",
        "Write Once Read Many",
        "Read Only Direct Access"
};
```

```
DEST            = /etc

EXTHDRS         = /usr/include/ctype.h \
                  /usr/include/curses.h \
                  /usr/include/fcntl.h \
                  /usr/include/sgtty.h \
                  /usr/include/stdio.h \
                  /usr/include/sun/dkio.h \
                  /usr/include/sun/dklabel.h \
                  /usr/include/sys/errno.h \
                  /usr/include/sys/fcntlcom.h \
                  /usr/include/sys/filio.h \
                  /usr/include/sys/ioccom.h \
                  /usr/include/sys/ioctl.h \
                  /usr/include/sys/sockio.h \
                  /usr/include/sys/stat.h \
                  /usr/include/sys/sysmacros.h \
                  /usr/include/sys/ttold.h \
                  /usr/include/sys/ttychars.h \
                  /usr/include/sys/ttycom.h \
                  /usr/include/sys/ttydev.h \
                  /usr/include/sys/types.h \
                  /usr/include/time.h

HDRS            =

DFLAGS          =

CFLAGS          = $(DFLAGS)

LDFLAGS         =

LIBS            = -lcurses -ltermlib

DLIBS           =

LINKER          = cc

MAKEFILE        = Makefile

OBJS            = cs35ut.o

PRINT           = pr

PROGRAM         = cs35ut

SRCS            = cs35ut.c

all:            $(PROGRAM)

$(PROGRAM):     $(OBJS) $(DLIBS)
                @echo -n "Loading $(PROGRAM) ... "
                @$(LINKER) $(LDFLAGS) $(OBJS) $(DLIBS) $(LIBS) -o $(PROGRAM)
                @echo "done"

clean:;         @rm -f $(OBJS)
```



```
/*****************************************************************************
******************************************************************************
****                                                                      ****
****                    Copyright 1987 Ciprico, Inc                       ****
****                                                                      ****
****                        Ciprico Incorporated                          ****
****                          2955 Xenium Lane                            ****
****                         Plymouth, MN 55441                           ****
****                           (612) 559-2034                             ****
****                                                                      ****
****    Module Name:    cs35ut.c                                          ****
****    Package:        Rimfire 3500 Driver for Unix Sun OS               ****
****    Module Rev:     $Revision$                    ****
****    Date:           $Date$              ****
****                                                                      ****
****    Subroutines:    main, help, opendevice, sfm, wfm, rew, ers_tape,  ****
****                    doformat, debug, domap,                           ****
****                    dopartitions, newconf,                            ****
****                    askpartitions, setpartitions, getpartitions,      ****
****                    showstatistics, doslip                            ****
****                                                                      ****
****    Description:                                                      ****
****            This program is a utility to work hand in hand with the   ****
****            Rimfire 3500 SCSI host bus adaptor.  It supports formatting ****
****            and sector mapping.  It allows access to some of the data ****
****            structures used in the Rimfire 3500 device driver for     ****
****            configuring and partitioning purposes.                    ****
****                                                                      ****
******************************************************************************
*****************************************************************************/
```



```
depend:;        @mkmf -f $(MAKEFILE) PROGRAM=$(PROGRAM) DEST=$(DEST)

index:;         @ctags -wx $(HDRS) $(SRCS)

install:        $(PROGRAM)
                @echo Installing $(PROGRAM) in $(DEST)
                @install -s $(PROGRAM) $(DEST)

print:;         @$(PRINT) $(HDRS) $(SRCS)

program:        $(PROGRAM)

tags:           $(HDRS) $(SRCS); @ctags $(HDRS) $(SRCS)

update:         $(DEST)/$(PROGRAM)

$(DEST)/$(PROGRAM): $(SRCS) $(DLIBS) $(HDRS) $(EXTHDRS)
                @make -f $(MAKEFILE) DEST=$(DEST) install

saber_src:      $(SRCS)
                #load $(CFLAGS) $(SRCS)

saber_obj:      $(OBJS)
                #load $(CFLAGS) $(OBJS)
###
cs35ut.o: /usr/include/ctype.h /usr/include/sys/types.h \
        /usr/include/sys/sysmacros.h /usr/include/sys/stat.h \
        /usr/include/fcntl.h /usr/include/sys/fcntlcom.h /usr/include/time.h \
        /usr/include/sys/errno.h /usr/include/sun/dklabel.h \
        /usr/include/sun/dkio.h /usr/include/sys/ioctl.h \
        /usr/include/sys/ttychars.h /usr/include/sys/ttydev.h \
        /usr/include/sys/ttold.h /usr/include/sys/ioccom.h \
        /usr/include/sys/ttycom.h /usr/include/sys/filio.h \
        /usr/include/sys/sockio.h /usr/include/curses.h /usr/include/stdio.h \
        /usr/include/sgtty.h
```



```
/*****************************************************************************
*                                                                           *
*                              Revision History                             *
*                                                                           *
*****************************************************************************

    Revision    Date            Author
        Description of Change
---------------------------------------------------------------------------

    1.1         04/13/87        D. A. Dickey
        Initial B Release for Rimfire 3500
    1.1         08/20/87        D. A. Dickey
        Initial A Release for Rimfire 3500

$Log:   I:\software\drivers\rf3500\sun\2_0\vcs\cs35ut.c_v  $

      Rev 2.0   21 Apr 1989 10:56:40   JMartin
   Initial revision.

    2.0         04/05/89        J. Martin
        Changes for Multiple controllers.  See cs35.c for details.

$Log$

    2.1         09/06/89 Jody Martin

        1. Made changes to driver to support the Rimfire 3523 as follows:

           - Made changes to support standalone utility.
           - Made many changes to the cs35ut utility, and the driver ioctl
             calls.
           - Made changes to support the new install script.
           - The format disk command now supports including the defect list
             during the disk format.
           - Changed the command codes to match as close as possible to the
             Rimfire 32XX rfutil.
           - Added support for selecting a dummy device to change debug
             value through the utility if you can't open another disk.
           - Added various features to existing commands.

*****************************************************************************/
#ifndef lint
static char utsccsid[] = "@(#)cs35ut.c  $Revision$  (75222006) $Date$, Copyright
#endif


#define HOG_PARTITION   /* define if you want a "free space hog" partition */

#ifdef STANDALONE
#define NO_FLOATS       /* Used for floating point failure in standalone */
#endif
```

```
#include <ctype.h>
#include <sys/types.h>
/*#include <sys/ioctl.h>        /* This is included indirectly from curses.h */
#include <sys/stat.h>
#include <sys/sysmacros.h>
#include <sys/mtio.h>
#include <fcntl.h>
#include <time.h>
#include <sys/errno.h>
#include "/sys/sun/dklabel.h"
#include "/sys/sun/dkio.h"
#include "/sys/sundev/cs35if.h"
#include "/sys/sundev/cs35io.h"
/*#include "/sys/sundev/cs35err.h"*/
#include "/sys/sundev/cs35prm.h"
#include "/sys/sundev/cs35flp.h"
#include "/sys/sundev/cs35int.h"

#include <curses.h>

#define DEFDEVICE       "/dev/rrs0a\000       " /* The default device to open at
static int disk = -1;
static char diskname[100];
static char line[256];
struct dk_map diskparts[NDKMAP];
static WINDOW   *verwin, *paramwin;
static int dev_id;

char    inp_char();

/* Mode Sense - Mode Select menu for tape devices */
char *menu[] = {
        "1.  Speed code (hex):",
        "2.  Buffered flag (1 = buffered):",
        "3.  Density code (hex):",
        "4.  Block Length (hex):"
};

/* global variables used in mode sense - mode select command */
int speed, buffered, density, blklen;

/* Pointer to global variables used in mode sense - mode select command */
int *params[4] = { &speed, &buffered, &density, &blklen };

/* Read defect list menu for format command */
char *defect_menu[] = {
        "1.  Manufacturers defect list read from disk",
        "2.  Grown defect list read from disk",
        "3.  Both the Manufacturers and the Grown list"
};

/* User entered defect list selection from the defect_menu */
byte defect_byte2[] = {
        0,      /* Don't include defect list data during format */
        0x10,   /* Include the manufacturers defect list during format */
```



```
        0x8,    /* Include the Grown defect list during format */
        0x18    /* Include the both the Grown and Manufacturers defect list */
};

/* Defect list formats - to be used with the format command */
byte defect_lst_fmt[] = {
        0,      /* Block Format */
        0x4,    /* Index Format */
        0x5     /* Physical Sector Format */
};

char *month[12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };


extern char     *sys_errlist[];
extern int      errno;

#define MEG     (1000L * 1000L) /* 1 megabyte in bytes */
#define MINPART (0)             /* Smallest partition index */
#define MAXPART (NDKMAP)        /* Largest partition index */
#define NBASE   ('a')           /* Partition base */

/*
 *      The "free space hog" values are MINPART .. MAXPART-1 or the following
 */

#define NO_HOG          -1              /* No hog partition */

#ifdef  HOG_PARTITION
#define DEFAULT_HOG     (MINPART+6)     /* Default - 'g' */
#else
#define DEFAULT_HOG     NO_HOG
#endif

int Hog = DEFAULT_HOG;

static char over_lap[MAXPART][(MAXPART * 2) + 1];
/*
 *      Added macros - Note SECTSIZE is 512
 */

/*
 *      Convert blocks to megabytes - unless NO_FLOATS - then it converts
 *      blocks to bytes
 */

#ifdef NO_FLOATS
#define BTOMEG(x)       ((long)((x) * SECTSIZE))
#else
#define BTOMEG(x)       ((float)((x) * SECTSIZE) / MEG)
#endif

/* These global variables were added for "smart" partitioning */
daddr_t Blocks;         /* # total blocks on the drive */
daddr_t Cblocks;        /* # blocks/cylinder */
```



```
#ifdef NO_FLOATS
long Capacity;          /* Disk capacity ( in bytes ) */
#else
float Capacity;         /* Disk capacity ( in Mbytes ) */
#endif

/*
 * This structure contains the labels used by SUN 4.2 to define the
 * configuration and partitions of a particular disk drive.
 * To add a label for a new disk or disk type, duplicate an existing
 * entry and change the appropriate fields of the structure.
 */

#ifdef SunOS3

static struct dk_label labels[] = {
        /*    apc     gap1    gap2    intrlv   ncyl   acyl    nhead   nsect   bh
        { "edit current label", "",
                0,      0,      0,      0,      0,      0,      0,      0,
                { 0, 0 },       { 0, 0 },       { 0, 0 },       { 0, 0 },
                { 0, 0 },       { 0, 0 },       { 0, 0 },       { 0, 0 },
                DKL_MAGIC, 0
        },
};

struct dk_label label;

#endif /* SunOS3 */

#ifdef SunOS4

static struct dk_label labels[] = {
        /*    apc     gap1    gap2    intrlv   ncyl   acyl    nhead   nsect   bh
        { "edit current label", "", 0, 0,
                0,      0,      0,      0,      0,      0,      0,      0,
                { 0, 0 },       { 0, 0 },       { 0, 0 },       { 0, 0 },
                { 0, 0 },       { 0, 0 },       { 0, 0 },       { 0, 0 },
                DKL_MAGIC, 0
        },
};
struct dk_label label;


#endif /* SunOS4 */

/*
 * This array defines strings which are used as labels when modifying the
 * disk configuration portion of the label.  They should be ordered exactly
 * as they are in the label structure.
 */
static char *fieldnames[] = {           /* starting with dkl_apc */
        "# of alt/cylinder",
        "size of gap1",         "size of gap2",         "interleave factor",
        "# of data cylinders",  "# of alt cylinders",   "# of heads",
        "# of sectors/track",   "label location",       "physical partition #",
        NULL
```



```
};

void print_partitions();
```

```
/*****************************************************************************
*                                                                            *
*       Subroutine:             main                                         *
*                                                                            *
*       Calling Sequence:       rfsdutil                                     *
*                                                                            *
*       Called by:              shell                                        *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:        None                                         *
*                                                                            *
*       Calls Subroutines:                                                   *
*               help, doformat, dogeometry, domap, opendevice,               *
*               dopartitions, showstatistics                                 *
*               doverify.                                                    *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*               This loops reading commands from the user and invokes a      *
*               subroutine to act as required.                               *
*                                                                            *
*****************************************************************************/

main(ac, av)
int ac;
char **av;
{
        /* Initialize screen, and request device to open */
        util_init(ac, av);
        do {
                prompt();                       /* display "cs35ut>" prompt */
                line[0] = inp_char(stdscr);     /* get user cmd entered */
                if (isupper(line[0]))           /* chng cmd to small if caps */
                    line[0] = tolower(line[0]);
                wmove(stdscr, 3, 0);
                wclrtoeol(stdscr);
                waddch(stdscr, ' ');
                wmove(stdscr, 1, 0);
                wclrtoeol(stdscr);
                wmove(stdscr, 1, 0);
                waddstr(stdscr, "Command: ");

                /* Below are the commands that are common to all device
                 * types - disk, tape, and dummy.
                 */
                switch (line[0]) {
                case 'b':
                    waddstr(stdscr, "Debug control");
                    debug();
                    break;
                case 'c':
                    waddstr(stdscr, "Identify Controller");
                    identify();
```



```
                    help();
                    break;
#ifdef notyet
                case 'h':
                    waddstr(stdscr, "General Options");
                    waddstr(stdscr, " - command not implemented");
                    /*General();*/
                    break;
                case 'i':
                    waddstr(stdscr, "Identify Devices");
                    waddstr(stdscr, " - command not implemented");
                    /*identify_dev();*/
                    break;
#endif
                case 'o':
                    waddstr(stdscr, "Open a new device");
                    opendevice(NULL);
                    break;
                case 'q':
                    waddstr(stdscr, "Quit");
                    finish(0);
                    break;
#ifdef notyet
                case 'y':
                    waddstr(stdscr, "Board Statistics");
                    waddstr(stdscr, " - command not implemented");
                    /*brdstat();*/
                    break;
                case 'z':
                    waddstr(stdscr, "Reset Controller");
                    waddstr(stdscr, " - command not implemented");
                    reset();
                    break;
#endif
                default:
                    /* if device is a dummy device, and the command entered
                     * by the user did not match any of the above commands,
                     * then the command is not valid, so return message.
                     */
                    if (dev_id == DUMMY)
                    {
                        help();
                        break;
                    } else
                    /* The following commands are common to both disk and
                     * tape.
                     */
                    {
                        switch (line[0]) {
                        case 'a':
                            waddstr(stdscr, "Load/Start Device");
                            load();
                            help();
                            break;
                        case 'd':
                            waddstr(stdscr, "Read Capacity");
```



```
                            capacity();
                            help();
                            break;
#ifdef notyet
                        case 'j':
                            waddstr(stdscr, "Unit Options");
                            waddstr(stdscr, " - command not implemented");
                            /*unitops();*/
                            break;
                        case 'k':
                            waddstr(stdscr, "Read/Write block");
                            waddstr(stdscr, " - command not implemented");
                            /*rdwrt_blk();*/
                            break;
#endif
                        case 'u':
                            waddstr(stdscr, "Unload/Stop Device");
                            unload();
                            help();
                            break;
#ifdef notyet
                        case 'x':
                            waddstr(stdscr, "Any SCSI command");
                            waddstr(stdscr, " - command not implemented");
                            /*any_scsi();*/
                            break;
#endif
                        default:
                            /* If the device is disk, the following commands
                             * apply, else go on to the tape commands.
                             */
                            if (dev_id == DIR_ACC)
                            {
                                /* The following commands are disk commands
                                 * only.
                                 */
                                switch (line[0]) {
                                case 'f':
                                    waddstr(stdscr, "Format disk");
                                    doformat();
                                    help();
                                    break;
                                case 'l':
                                    waddstr(stdscr, "Choose label");
                                    dolabel();
                                    help();
                                    break;
                                case 'm':
                                    waddstr(stdscr, "Map sectors");
                                    domap();
                                    help();
                                    break;
                                case'n':
                                    waddstr(stdscr, "Mode Sense/Select");
                                    waddstr(stdscr, " - command not implemented"
                                    /*domodsen();*/
```



```
                                    break;
                                case 'r':
                                    waddstr(stdscr, "Read label");
                                    readlabel();
                                    help();
                                    break;
                                case 's':
                                    waddstr(stdscr, "Show label");
                                    showlabel();
                                    help();
                                    break;
                                case 'v':
                                    waddstr(stdscr, "Verify disk format");
                                    doverify();
                                    help();
                                    break;
                                case 'w':
                                    waddstr(stdscr, "Write label to disk");
                                    writelabel();
                                    break;
                                default:
                                    help();
                                }
                            } else
                            if (dev_id == SEQ_ACC)
                            {
                                /* The following commands are tape commands
                                 * only.
                                 */
                                switch (line[0]) {
                                case 'e':
                                    waddstr(stdscr, "Erase tape");
                                    ers_tape();
                                    help();
                                    break;
                                case'n':
                                    waddstr(stdscr, "Mode Sense/Select");
                                    dodensity();
                                    help();
                                    break;
                                case 'r':
                                    waddstr(stdscr, "Rewind");
                                    rew();
                                    help();
                                    break;
                                case 's':
                                    waddstr(stdscr, "Search Filemark");
                                    sfm();
                                    break;
                                case 't':
                                    waddstr(stdscr, "Re-Tension Tape");
                                    reten();
                                    help();
                                    break;
                                case 'w':
                                    waddstr(stdscr, "Write Filemark");
```

```
                                        wfm();
                                        help();
                                        break;
                                    default:
                                        help();
#ifdef notyet
                                        wmove(stdscr, 3, 0);
                                        waddstr(stdscr, "Illegal command -");
#endif

                                    }     /* tape command switch */
                                }     /* separate disk and tape commands */
                            }     /* common disk and tape cmd switch */
                        }
                    }       /*  Common command switch */

                    wmove(stdscr, 1, 0);
                    winsch(stdscr, ' ');
                    winsch(stdscr, 't');
                    winsch(stdscr, 's');
                    winsch(stdscr, 'a');
                    winsch(stdscr, 'L');
            } while (TRUE);
            /*NOTREACHED*/
}

char
inp_char(win)
register WINDOW *win;
{
            register char ch;
            register int x, y;

            do {
                    wrefresh(win);
                    getyx(win, y, x);
                    ch = wgetch(win);
                    if (ch == '\22' || ch == '\14') {
                            wmove(win, y, x);
                            clearok(win, TRUE);
                    } else
                            break;
            } while (TRUE);
            return (ch);
}

clriwin() {
            register int i;
            register int cnt;

            cnt = 11 < LINES ? 11 : LINES;
            cnt -= 4;
            for (i = 4; cnt; cnt--) {
                    wmove(stdscr, i, 0);
                    wprintw(stdscr, "%-*s", COLS, " ");
                    i++;
```



```
            }
}

clrowin() {
            register int i;

            for (i = 11; i < LINES; i++) {
                    wmove(stdscr, i, 0);
                    wprintw(stdscr, "%-*s", COLS, " ");
            }
}

input(win, buf)
register WINDOW *win;
register char *buf;
{
            register char ch;
            register int cnt = 0;
            int cur_x, cur_y;

            getyx(win, cur_y, cur_x);

            while (TRUE) {
                    wrefresh(win);
                    ch = inp_char(win);
                    if (ch == killchar()) {
                            cnt = 0;
                            wmove(win, cur_y, cur_x);
                            wclrtoeol(win);
                            waddch(win, ' ');
                            wmove(win, cur_y, cur_x);
                    } else if (ch == erasechar() && cnt) {
                            int nowcur_y, nowcur_x;
                            getyx(win, nowcur_y, nowcur_x);
                            --nowcur_x;
                            --cnt;
                            wmove(win, nowcur_y, nowcur_x);
                            wclrtoeol(win);
                            waddch(win, ' ');
                            wmove(win, nowcur_y, nowcur_x);
                    } else if (ch == '\r' || ch == '\n') {
                            line[cnt] = '\0';
                            break;
                    } else if (ch != erasechar()) {
                            line[cnt++] = ch;
                            if (waddch(win, ch) == ERR) {
                                    line[cnt] = '\0';
                                    return(ERR);
                            }
                    }
            }
            return(OK);
}

finish(val)
register val;
```



```
{
            wmove(stdscr, LINES-1, 0);
            wrefresh(stdscr);
            delwin(verwin);
            endwin();
            printf("\n");
            exit(val);
}

/*  This routine displays the "cs35ut>" prompt. */
prompt() {
            register char *fmt = "cs35ut> ";

            wmove(stdscr, 2, 0);
            wprintw(stdscr, "%-*s", COLS, fmt);
            wmove(stdscr, 2, strlen(fmt));
            wrefresh(stdscr);
}
```



```
/*************************************************************************
 *                                                                       *
 *      pak                                                              *
 *                                                                       *
 *      Function:                                                        *
 *              Suspends re-writing of screen until user presses a key   *
 *                                                                       *
 *      Inputs:                                                          *
 *              none                                                     *
 *                                                                       *
 *      Outputs:                                                         *
 *              none                                                     *
 *                                                                       *
 *************************************************************************/

pak()
{
            center_line(LINES-1,"<Press Any Key>");
            inp_char(stdscr);
}
```

```
/*****************************************************************************
 *                                                                           *
 *      clrpak                                                               *
 *                                                                           *
 *      Function:                                                            *
 *              Clears the last line of the screen                           *
 *                                                                           *
 *      Inputs:                                                              *
 *              none                                                         *
 *                                                                           *
 *      Outputs:                                                             *
 *              none                                                         *
 *                                                                           *
 *****************************************************************************/

int
clrpak()
{
        move(LINES-1,0); clrtoeol();
}

/*  The util_init routine initializes the screen routines, and calls the
 *  opendevice routine to open a device.
 */
util_init(ac, av)
int ac;
char **av;
{
        register char   *defdev = (char *) NULL;

        if (initscr() == (WINDOW *) ERR) {  /* initialize screen routines */
                error("Can't init curses.");
                exit(-1);
        }

        raw();
        noecho();

        if (ac > 1) {
                defdev = *++av;
                ac--;
        }


        if (opendevice(defdev) < 0) {
                wrefresh(stdscr);
                endwin();
                printf("\n");
                error("Can't open default device.");
                exit(-1);
        }

        help();         /* display the help menu */
        init_verwin();
}
```



```
init_verwin() {
        if ((verwin = newwin(0, 0, 11, COLS-50)) == (WINDOW *) ERR) {
                closedevice();
                endwin();
                printf("\n");
                error("Can't init verify window.");
                exit(-1);
        }
        scrollok(verwin, TRUE);
}

error(str)
register char *str;
{
        fprintf(stderr, "%s\n", str);
        fflush(stderr);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             help                                         *
*                                                                            *
*       Calling Sequence:       help()                                       *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:        None                                         *
*                                                                            *
*       Calls Subroutines:      None                                         *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*             This subroutine prints a short description of the              *
*             commands available.  There are different menus for disk        *
*             devices, tape devices, and dummy device.                       *
*                                                                            *
*****************************************************************************/

help() {
        clrowin();
        clriwin();
        wmove(stdscr, 9, 0);
        waddstr(stdscr, "Commands available are:");

        /* Display disk menu for a disk device */
        if (dev_id == DIR_ACC)
        {
                wmove(stdscr, 11, 0);
                waddstr(stdscr, "a - load/start device\n");
                waddstr(stdscr, "b - debug control\n");
                waddstr(stdscr, "c - identify controller\n");
                waddstr(stdscr, "d - read capacity\n");
/*              waddstr(stdscr, "e - read defect list\n");*/
                waddstr(stdscr, "f - format the drive\n");
/*              waddstr(stdscr, "h - general options\n");
                waddstr(stdscr, "i - identify devices\n");
                waddstr(stdscr, "j - unit options\n");
                waddstr(stdscr, "k - read/write block\n"); */
                waddstr(stdscr, "l - Choose/Edit a disk label\n");
                waddstr(stdscr, "m - map sectors\n");
                wmove(stdscr, 11, COLS/2);
/*              waddstr(stdscr, "n - mode sense/select\n");*/
                waddstr(stdscr, "o - open a disk device\n");
                wmove(stdscr, 12, COLS/2);
                waddstr(stdscr, "q - quit\n");
                wmove(stdscr, 13, COLS/2);
                waddstr(stdscr, "r - read and display label\n");
                wmove(stdscr, 14, COLS/2);
                waddstr(stdscr, "s - show label\n");
                wmove(stdscr, 15, COLS/2);
```



```
                waddstr(stdscr, "u - unload media\n");
                wmove(stdscr, 16, COLS/2);
                waddstr(stdscr, "v - verify format\n");
                wmove(stdscr, 17, COLS/2);
                waddstr(stdscr, "w - write the disk label\n");
                wmove(stdscr, 18, COLS/2);
/*              waddstr(stdscr, "x - any SCSI command\n"); */
                wmove(stdscr, 19, COLS/2);
/*              waddstr(stdscr, "y - board statistics\n"); */
                wmove(stdscr, 20, COLS/2);
/*              waddstr(stdscr, "z - reset controller\n"); */
                wmove(stdscr, 21, COLS/2);
                wmove(stdscr, 22, COLS/2);
        } else
        /* Display tape menu for a tape device */
        if (dev_id == SEQ_ACC)
        {
                wmove(stdscr, 11, 0);
                waddstr(stdscr, "a - load/start device\n");
                waddstr(stdscr, "b - debug control\n");
                waddstr(stdscr, "c - identify controller\n");
                waddstr(stdscr, "d - read capacity\n");
                waddstr(stdscr, "e - erase tape\n");
/*              waddstr(stdscr, "h - general options\n");
                waddstr(stdscr, "i - identify devices\n");
                waddstr(stdscr, "j - unit options\n");
                waddstr(stdscr, "k - read/write block\n"); */
                waddstr(stdscr, "n - mode sense/select\n");
                wmove(stdscr, 11, COLS/2);
                waddstr(stdscr, "o - open a disk device\n");
                wmove(stdscr, 12, COLS/2);
                waddstr(stdscr, "q - quit\n");
                wmove(stdscr, 13, COLS/2);
                waddstr(stdscr, "r - rewind tape\n");
                wmove(stdscr, 14, COLS/2);
                waddstr(stdscr, "s - search filemark\n");
                wmove(stdscr, 15, COLS/2);
                waddstr(stdscr, "t - retension tape\n");
                wmove(stdscr, 16, COLS/2);
                waddstr(stdscr, "u - unload media\n");
                wmove(stdscr, 17, COLS/2);
                waddstr(stdscr, "w - write filemark\n");
                wmove(stdscr, 18, COLS/2);
/*              waddstr(stdscr, "x - any SCSI command\n");
                wmove(stdscr, 19, COLS/2);
                waddstr(stdscr, "y - board statistics\n");
                wmove(stdscr, 20, COLS/2);
                waddstr(stdscr, "z - reset controller\n"); */
        } else
        {
                /* Display dummy device menu */
                wmove(stdscr, 11, 0);

                waddstr(stdscr, "b - debug control\n");
                waddstr(stdscr, "c - identify controller\n");
/*              waddstr(stdscr, "h - general options\n");
```

```
                waddstr(stdscr, "i - identify devices\n"); */
                waddstr(stdscr, "o - open a disk device\n");
                waddstr(stdscr, "q - quit\n");
/*              waddstr(stdscr, "y - board statistics\n");
                waddstr(stdscr, "z - reset controller\n"); */
        }

        wrefresh(stdscr);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             opendevice                                   *
*                                                                            *
*       Calling Sequence:       opendevice()                                 *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:        None                                         *
*                                                                            *
*       Calls Subroutines:      None                                         *
*                                                                            *
*       Public/Global Variables:                                             *
*                                                                            *
*                               line - buffer for reading user input         *
*                               disk - file descriptor of device file opened.*
*                                                                            *
*       Description:                                                         *
*               This subroutine reads a line from the user which it then     *
*               attempts to open as device file.  If a device file is already*
*               open, it is closed.  If the open fails, "disk" is set to -1. *
*                                                                            *
*****************************************************************************/

closedevice() {
        close(disk);
        disk = -1;
        wmove(stdscr, 0, 0);
        wprintw(stdscr, "Open Device: %-30s  Size: %-10u", "(None)", 0);
}

opendevice(dname)
register char *dname;
{
        register onceonly = FALSE;
        struct stat statb;
        int part, size;

        if (disk != -1) {
                closedevice();
        }

        for (;;) {
                wmove(stdscr, 4, 0);
                wprintw(stdscr, "%-50s", "New Device: ");
                wmove(stdscr, 4, 12);
                line[0] = '\0';
                wrefresh(stdscr);
                if (dname == (char *) NULL || onceonly)
                        input(stdscr, line);
                else {
                        strcpy(line, dname);
                        waddstr(stdscr, line);
```



```
                        onceonly = TRUE;
                }
                wrefresh(stdscr);
                wmove(stdscr, 5, 0);
                if ((disk = open(line, O_RDWR)) < 0) {
                        if (line[0] == '\0' || line[0] == '\33') {
                                disk = -1;
                                return(disk);
                        }
                        wprintw(stdscr, "Can't open: %-40s\n", line);
                        wprintw(stdscr, "open: %-40s\n", sys_errlist[errno]);
                        continue;
                }

                wclrtoeol(stdscr);
                break;
        }
        strcpy(diskname, line);

        /* Get the device type from the rfdinfo array, thru this ioctl call */
        if (ioctl(disk, RFIOCGDEVID, &dev_id) < 0) {
                wprintw(stdscr, "RFIOCGDEVID: %s", sys_errlist[errno]);
                return(-1);
        }

        /* If dummy device, just display open device, size, and help menu. */
        if (dev_id == DUMMY)
        {
                wmove(stdscr, 4, 0);
                wprintw(stdscr, "DUMMY Device");
                wrefresh(stdscr);
                size = 0;
                wmove(stdscr, 0, 0);
                wprintw(stdscr, "Open Device: %-30s  Size: %-10u", diskname, siz
                help();
                return(disk);
        }

        wmove(stdscr, 0, 0);
        if (getpartitions(diskparts) < 0) {
                wmove(stdscr, 13, 0);
                wprintw(stdscr, "Can't get partitions from %s\n", diskname);
                closedevice();
                return(-1);
        }

#ifdef notyet
        /* Try to read the label if this is a disk */
        if(dev_id == DIR_ACC)
        {
                lseek(disk, 0L, 0);
                if (read(disk, (char *)&label, sizeof(struct dk_label)) < 0) {
                        waddstr(stdscr, "Disk label was not readable\n");
                }
        }
#endif
```



```
        if (diskname[strlen(diskname) - 3] == 's')
                part = diskname[strlen(diskname) - 1] - 'a';
        else
                part = 0;
        size = diskparts[part].dkl_nblk;
        wprintw(stdscr, "Open Device: %-30s  Size: %-10u", diskname, size);

        help();
        return(disk);
}
```

```
/*****************************************************************************
*                                                                            *
*       Subroutine:             doformat                                     *
*                                                                            *
*       Calling Sequence:       doformat()                                   *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:                                                     *
*                       fmt - disk format structure to pass to the driver    *
*                       lastcyl - last cylinder requested                    *
*                                                                            *
*       Calls Subroutines:  ioctl - to pass the format command to the driver *
*                                                                            *
*       Public/Global Variables:                                             *
*                       disk - file descriptor of disk device to format      *
*                       line - buffer for user input                         *
*                                                                            *
*       Description:                                                         *
*               This subroutine asks the user what portion of the disk to    *
*               format and then proceeds to format the specified area on a   *
*               cylinder by cylinder basis.                                  *
*                                                                            *
*****************************************************************************/

doformat()
{

        format fmt;
        register struct dk_label *lp;
        dword   *lba, *len;
        read_cap cap;
        int i;
        int deflst_sel;
        int list_fmt;
        int good_deflist = 0;
        int time;

        clriwin();
        clrowin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                return;
        }

        lp = &label;
        lba = (dword *)&cap;
        len = (dword *)cap.blklen;
        *lba = 0;

        errno = 0;
        if (ioctl(disk, RFIOCRDCAP, &cap) < 0) {
```



```
                wprintw(stdscr, "RFIOCRDCAP: %s\n", sys_errlist[errno]);
                waddstr(stdscr, "Getting # of blocks from disk geometry\n");
                *lba = (int)lp->dkl_ncyl * lp->dkl_nhead * lp->dkl_nsect;
        }

        /* A read defect list command will not be done if def_lst_type = 0,
         * initialize it to zero.
         */
        fmt.def_lst_type = 0;


        /* time is set equal to the approximate amount of time to format
         * the specific device.  This is just an approximate value.
         * It is determined by multiplying the number of cylinders
         * by the number of heads (number of tracks), and multiplying
         * that value by the amount of time it takes for two revolutions,
         * (.0333333333).
         */
#ifdef NO_FLOATS
        time = (int) ((lp->dkl_ncyl * lp->dkl_nhead / 30) / 60);
#else
        time = (int) ((.033333333 * lp->dkl_ncyl * lp->dkl_nhead) / 60);
#endif

        waddstr(stdscr, "Enter 'y' to include a defect list during format: ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] == 'y') {
                waddstr(stdscr, "Yes");         /* Usr wants defect list */
                waddstr(stdscr, "\nEnter choice of defect list from menu below:
                /* Display defect list menu options */
                for(i=0; i < 3; i++) {
                        wmove(stdscr, 8 + i, 0);
                        wprintw(stdscr, "%-45.45s  \n", defect_menu[i]);
                }

                /* Move curser to top of menu and wait for entry by user */
                while(TRUE)
                {
                    wmove(stdscr, 6, 46);
                    input(stdscr, line);            /* Get user input */
                    sscanf(line, "%d", &deflst_sel);    /* chng from ascii  to #
                    /* Check for valid menu selection */
                    if(deflst_sel > 0 || deflst_sel < 4)
                    {
                        /* Set read defect list cmd blk - byte 2 */
                        fmt.def_lst_type = defect_byte2[deflst_sel];
                        wmove(stdscr, 7, 0);
                        wclrtoeol(stdscr);
                        break;
                    }

                    wmove(stdscr, 7, 0);
                    waddstr(stdscr, "Illegal entry, try again\n");
                    wrefresh(stdscr);
                }
```



```
        } else
        {
                waddstr(stdscr, "No");
        }

        wmove(stdscr, 12, 0);
        waddstr(stdscr, "What interleave? ");
        input(stdscr, line);
        fmt.interleave = atoi(line);
        wmove(stdscr, 14, 0);
        wprintw(stdscr, "Formatting blocks 0 to %u in approximately %d minutes.\
        wprintw(stdscr, "This time will vary according to disk manufacturer\n");
        waddstr(stdscr, "Are you sure? ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                pak();
                return;
        }

        waddstr(stdscr, "Yes");
        wmove(stdscr, 16, 0);
        waddstr(stdscr, "Formatting:\n");
        /*clrowin();*/
        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, RFIOCFMT, &fmt) < 0) {
                wprintw(stdscr, "RFIOCFMT: %s", sys_errlist[errno]);
                pak();
                return(-1);
        }
        wmove(stdscr, 16, 12);
        waddstr(stdscr, "Done");

        wmove(stdscr, LINES - 3, 0);
        waddstr(stdscr, "Do you want to write the label to the disk? (y/n):");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                pak();
                return;
        }

        waddstr(stdscr, "Yes");
        lseek(disk, 0L, 0);
        if (write(disk, (char *)&label, sizeof(struct dk_label)) < 0) {
                wmove(stdscr, 8, 0);
                wprintw(stdscr, "Write: %s", sys_errlist[errno]);
                pak();
                return;
        }
        pak();
}

capacity()
{
```



```
        dword   *lba, *len;
        read_cap        cap;

        clriwin();
        clrowin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        lba = (dword *)&cap;
        len = (dword *)cap.blklen;

        errno = 0;
        if (ioctl(disk, RFIOCRDCAP, &cap) < 0) {
                wprintw(stdscr, "RFIOCRDCAP: %s", sys_errlist[errno]);
                pak();
                return(-1);
        }

        wmove(stdscr, 10, 0);
        waddstr(stdscr, "                          Dec          Hex\n");
        wprintw(stdscr, "Last Logical block  %5d       %5x\n", *lba, *lba);
        wprintw(stdscr, "Block length        %5d       %5x\n", *len, *len);
        pak();
        return(0);
}

getblock(which, bp)
char *which;
register int *bp;
{
        int done, block;

        clriwin();
        wmove(stdscr, 5, 0);
        waddstr(stdscr, "'q' to quit\n");

        if(!(strcmp(which, "Number of")))
                wprintw(stdscr, "Enter %s Blocks:\n", which);
        else
                wprintw(stdscr, "Enter %s Block:\n", which);

        *bp = 0;
        done = FALSE;
        while (!done) {
                wmove(stdscr, 6, strlen(which) + 16);
                wclrtoeol(stdscr);
                wmove(stdscr, 6, strlen(which) + 17);
                input(stdscr, line);
                wmove(stdscr, 8, 0);
                wclrtoeol(stdscr);
                if (line[0] != '\0') {
                        if (!isdigit(line[0]) && line[0] != '-') {
```

```
                                if ((line[0] | 040) == 'q')
                                        return(FALSE);
                        } else {
                                if (sscanf(line, "%d", &block) == 1) {
                                        *bp = block;
                                        return(TRUE);
                                }
                        }
                        wprintw(stdscr, "Illegal input: %s", line);
                }
        }
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             doverify                                     *
*                                                                            *
*       Calling Sequence:       doverify()                                   *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:                                                     *
*                       vfy - disk verify structure to pass to the driver    *
*                       lastcyl - last cylinder requested                    *
*                                                                            *
*       Calls Subroutines:  ioctl - to pass the format command to the driver *
*                                                                            *
*       Public/Global Variables:                                             *
*                   disk - file descriptor of disk device to format          *
*                   line - buffer for user input                             *
*                                                                            *
*       Description:                                                         *
*               This subroutine asks the user what portion of the disk to    *
*               verify and then proceeds to verify the specified area on a   *
*               cylinder by cylinder basis.                                  *
*                                                                            *
*****************************************************************************/

doverify()
{
        register char *cp;
        int scyl, ecyl, shd, ehd;
        int didverify = FALSE, domaps = FALSE;
        int numtimes, sblock, eblock;
        extern char *strchr();
        dword   *lba, *len;
        read_cap        cap;

        clriwin();
        clrowin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        lba = (dword *)&cap;
        len = (dword *)cap.blklen;
        *lba = 0;

        errno = 0;
        if (ioctl(disk, RFIOCRDCAP, &cap) < 0) {
                wprintw(stdscr, "RFIOCRDCAP: %s", sys_errlist[errno]);
                pak();
                return(-1);
```



```
        }

        wmove(stdscr, 7, 0);
        waddstr(stdscr, "How many passes do you want to make? ");
        input(stdscr, line);
        if (line[0] == '\0' || (line[0] | 040) == 'q')
                return;
        numtimes = atoi(line);
        if (numtimes < 1)
                numtimes = 1;
        wmove(stdscr, 8, 0);
        wprintw(stdscr, "OK, doing %d verify passes.\n", numtimes);
        wmove(stdscr, 9, 0);
        waddstr(stdscr, "Do you want to map bad sectors as they are discovered?
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                domaps = FALSE;
        } else {
                waddstr(stdscr, "Yes");
                domaps = TRUE;
        }
        wmove(stdscr, 10, 0);
        waddstr(stdscr, "Ready to verify disk, enter 'y' to proceed: ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                pak();
                return;
        }

        sblock = 0;
        eblock = *lba;

        clriwin();
        wmove(stdscr, 5, 0);
        waddstr(stdscr, "Enter 'y' to verify the whole disk: ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                wmove(stdscr, 11, 0);
                wclrtoeol(stdscr);
                waddch(stdscr, ' ');
                wmove(stdscr, 11, 0);
                if (getblock("Starting", &sblock) != TRUE) {
                        pak();
                        return;
                }
                wmove(stdscr, 11, 0);
                wclrtoeol(stdscr);
                waddch(stdscr, ' ');
                wmove(stdscr, 11, 0);
                if (getblock("Ending", &eblock) != TRUE) {
                        pak();
                        return;
                }
                wmove(stdscr, 11, 0);
```



```
                wclrtoeol(stdscr);
                waddch(stdscr, ' ');
                wmove(stdscr, 11, 0);
                if (sblock > eblock) {
                        wmove(stdscr, 5, 0);
                        waddstr(stdscr, "The start location must be before the e
                        pak();
                        return;
                }
        }
        clriwin();
        wmove(stdscr, 5, 0);
        wprintw(stdscr, "Verifying Blocks %u to %u\nAre you sure? ", sblock, ebl
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                pak();
                return;
        }
        waddstr(stdscr, "Yes");
        wmove(stdscr, 10, 0);
        waddstr(stdscr, "Verifying:\n");
        clrowin();
        wclear(verwin);
        if (ver_blocks(&sblock, &eblock, domaps, numtimes) != TRUE)
                return;
        wmove(stdscr, 10, 12);
        waddstr(stdscr, "Done");
        pak();
}
```

```
ver_map(ver)
register struct dk_vfy *ver;
{
        struct dk_mapr map;

        map.dkm_fblk = ver->dkv_badblk;
        map.dkm_nblk = 1;
        errno = 0;
        if (ioctl(disk, RFIOCMAP, &map) < 0) {
                wprintw(stdscr, "RFIOCMAP: %s", sys_errlist[errno]);
                wrefresh(verwin);
                return;
        }
        wprintw(verwin, "Block %d mapped.\n", map.dkm_fblk);
        wrefresh(verwin);
}
```



```
ver_blocks(sbp, ebp, domaps, numtimes)
register int *sbp, *ebp, domaps, numtimes;
{
        register int i;
        struct dk_mapr map;
        struct dk_vfy ver;
        int nsects;
        struct dk_geom geom;

        if (*sbp > *ebp)
                return(FALSE);

        wmove(stdscr, 13, 0);

        if (dev_id != DIR_ACC)
        {
                wprintw(stdscr, "Verify is only implemented for Floppies and nor
                return(FALSE);
        }

        errno = 0;
        if (ioctl(disk, DKIOCGGEOM, &geom) < 0) {
                wmove(stdscr, 13, 0);
                wprintw(stdscr, "DKIOCGGEOM: %s", sys_errlist[errno]);
                return(FALSE);
        }

        nsects = geom.dkg_nsect;

        if (nsects == 0) {
                wmove(stdscr, 11, 0);
                waddstr(stdscr, "What kind of drive is this?  One with zero sect
                waddstr(stdscr, "Assuming 35 Sectors Per Track.\n");
                nsects = 35;
        }

        while (*sbp <= *ebp) {
                wmove(stdscr, 13, 0);
                wprintw(stdscr, "Block %4d", *sbp);
                wrefresh(stdscr);
                for (i = numtimes; i; i--) {
                        ver.dkv_blkno = *sbp;
                        if (nsects > (*ebp - *sbp + 1))
                                nsects = (*ebp - *sbp) + 1;
                        ver.dkv_nblk = nsects;
                        ver.dkv_error = 0;
                        ver.dkv_badblk = 0;
                        errno = 0;
                        if (ioctl(disk, RFIOCVFY, &ver) < 0) {
                                wprintw(verwin, "RFIOCVFY: %s", sys_errlist[errn
                                wrefresh(verwin);
                                return(FALSE);
                        }
                        if (ver.dkv_error != 0) {
                                wprintw(verwin, "Verify failed: block=%d (0x%x)\
```



```
                                        ver.dkv_badblk, ver.dkv_error);
                                wrefresh(verwin);
                                if (domaps)
                                        ver_map(&ver);
                        }
                }
                *sbp += nsects;
        }
        (*sbp)--;
        wmove(stdscr, 13, 0);
        wprintw(stdscr, "Block %4d", *sbp);
        return(TRUE);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             debug                                        *
*                                                                            *
*       Calling Sequence:       debug()                                      *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:  dbg_var - the debug variable from the driver.      *
*                                                                            *
*       Calls Subroutines: ioctl - to pass the ioctl request to the driver
*                                                                            *
*       Public/Global Variables:                                             *
*               disk - file descriptor of disk device to query for the cache *
*               line - buffer for user input                                 *
*                                                                            *
*       Description:                                                         *
*               This routine prints out the debug variable and then          *
*               queries the user for a new value.                            *
*                                                                            *
*****************************************************************************/

debug()
{
        unsigned long dbg_var;

        clriwin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "Please use 'o' to open a disk device file");
                return;
        }

        errno = 0;
        if (ioctl(disk, RFIOCGDEBUG, &dbg_var) < 0) {
                wprintw(stdscr, "Get debug: %s", sys_errlist[errno]);
                return;
        }
        wprintw(stdscr, "The current debug variable setting in hex is: (%x)\n",
        waddstr(stdscr, "Do you want to change it? ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                return;
        }
        waddstr(stdscr, "Yes");

        wmove(stdscr, 7, 0);
        wprintw(stdscr, "Debug value in hex: (%x) = ", dbg_var);
        input(stdscr, line);
        sscanf(line, "%x", &dbg_var);

        wmove(stdscr, 8, 0);
```

```
        errno = 0;
        if (ioctl(disk, RFIOCSDEBUG, &dbg_var) < 0) {
                wprintw(stdscr, "Set debug: %s", sys_errlist[errno]);
                return;
        }
        wprintw(stdscr, "Debug value changed to (%x)\n", dbg_var);
}
```



```
/*****************************************************************************
*                                                                            *
*       Subroutine:             domap                                        *
*                                                                            *
*       Calling Sequence:       domap()                                      *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Calling Parameters:     None                                         *
*                                                                            *
*       Local Variables:                                                     *
*                       geom - geometry information for the current device   *
*                       map - structure to pass map command to the driver    *
*                       sblk, eblk - Starting and End block to map           *
*                       tblk - Block to map To                               *
*                       lblk - Last block on disk                            *
*                       cp - used to step through the user input             *
*                                                                            *
*       Calls Subroutines:                                                   *
*               ioctl - to get configuration information from the driver     *
*                       and to pass the map command to the driver            *
*                                                                            *
*       Public/Global Variables:                                             *
*                               line - buffer for user input                 *
*                                                                            *
*       Description:                                                         *
*               This subroutine asks the user for a range of blocks          *
*               to map and passes the information to the driver.             *
*                                                                            *
*****************************************************************************/

domap()
{
        struct dk_mapr map;
        daddr_t sblock, eblock, nblock;
        register char *cp;
        extern char *strchr();

        clriwin();
        clrowin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        wmove(stdscr, 11, 0);
        if (getblock("Starting", &sblock) != TRUE)
                return;
        wmove(stdscr, 17, 0);
        wprintw(stdscr, "Starting block: %d\n", sblock);
        if (getblock("Number of", &nblock) != TRUE)
                return;
        wmove(stdscr, 18, 0);
```



```
        wprintw(stdscr, "Number of blocks: %d\n", nblock);

        eblock = sblock + nblock;

        clriwin();
        wmove(stdscr, 5, 0);
        wprintw(stdscr, "Enter 'y' to map blocks %d through %d  ", sblock, ebloc
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                pak();
                return;
        }
        waddstr(stdscr, "Yes");
        map.dkm_fblk = sblock;
        map.dkm_nblk = nblock;

        errno = 0;
        if (ioctl(disk, RFIOCMAP, &map) < 0) {
                wmove(stdscr, 7, 0);
                wprintw(stdscr, "RFIOCMAP: %s", sys_errlist[errno]);
        }
        pak();
}
```



```
/*****************************************************************************
 *                                                                           *
 *       Subroutine:             showpartitions                              *
 *                                                                           *
 *       Calling Sequence:       showpartitions(map)                         *
 *                                                                           *
 *       Called by:              main                                        *
 *                                                                           *
 *       Calling Parameters:                                                 *
 *                       map - pointer to the partition mapping information  *
 *                       current_line - starting line number for printing    *
 *                                                                           *
 *       Local Variables:                                                    *
 *                                                                           *
 *       Calls Subroutines:      None                                        *
 *                                                                           *
 *       Public/Global Variables:None                                        *
 *                                                                           *
 *       Description:                                                        *
 *               This subroutine displays the partition mapping information  *
 *               passed to it.                                               *
 *                                                                           *
 *****************************************************************************/

int
showpartitions(map, current_line)
        register struct dk_map *map;
        register int current_line;
{
        register struct dk_label *lp;
        register int i, j, n, test;
        register daddr_t end_cyl;
        char print_line[200];
        int o_lap;
        int MASK;
        char olap[(MAXPART * 2) +1];

        lp = &label;
        Cblocks = (long)lp->dkl_nhead * lp->dkl_nsect;
        for (i = LINES < 24 ? LINES : 23; i > 14; i--) {
                move(i, 0);
                wclrtoeol(stdscr);
        }

        /* Find overlap for each partition */
        for ( i = 0; i < MAXPART; i++)
        {
                /* Fill overlap string array with spaces initially*/
                for( n = 0; n < ((MAXPART * 2) + 1); n++){
                        olap[n] = ' ';
                }

                /* find overlaps for this partition */
                o_lap = overlap(i);
```

```
                for ( j = 0, MASK = 1, n = 0; j < MAXPART; j++ )
                {
                        if(o_lap & MASK)
                        {
                                olap[n++] = NBASE + j;
                                olap[n++] = ' ';
                        }
                        MASK = MASK << 1;
                }
                olap[MAXPART * 2] = NULL;
                strcpy(over_lap[i], olap);

        }

        center_line(current_line,
                "    starting    ending          size       ");
        current_line++;
        center_line(current_line,
                "Partition   cylinder   cylinder   blocks /  Mbytes
        current_line++;
        for (i = MINPART; i < MAXPART; i++) {
                if ( Cblocks == 0 )
                        end_cyl = (daddr_t)0;
                else
                        end_cyl = (daddr_t)(map->dkl_cylno +
                                        (map->dkl_nblk / Cblocks) - 1);
#ifdef NO_FLOATS
{
                long tmptmp = BTOMEG((long)map->dkl_nblk);
                char rem[100];
                sprintf(rem, "%02ld", ((tmptmp % MEG) * 100) / MEG);
                sprintf(print_line,
                        "          %c %c       %5ld      %5ld    %8ld /%5ld.%.2s
                        NBASE + i, (i == Hog) ? '*' : ' ',
                        (long)map->dkl_cylno,
                        (end_cyl == (long)-1 ? (long)0 : end_cyl),
                        (long)map->dkl_nblk,
                        tmptmp / MEG, rem,
                        over_lap[i]);
}
#else   /* !NO_FLOATS */
                sprintf(print_line,
                        "          %c %c       %5ld      %5ld     %8ld / %7.2f
                        NBASE + i, (i == Hog) ? '*' : ' ',
                        (long)map->dkl_cylno,
                        (end_cyl == (long)-1 ? (long)0 : end_cyl),
                        (long)map->dkl_nblk,
                        BTOMEG((long)map->dkl_nblk),
                        over_lap[i]);
#endif /* NO_FLOATS */
                center_line(current_line, print_line);
                map++;
                current_line++;
        }
        return(current_line);
}
```



```
/*****************************************************************************
 *                                                                           *
 *      Subroutine:             askpartitions                                *
 *                                                                           *
 *      Calling Sequence:       askpartitions(map)                           *
 *                                                                           *
 *      Called by:              dolabel                                      *
 *                                                                           *
 *      Calling Parameters:                                                  *
 *                      lp - label                                           *
 *                                                                           *
 *      Calls Subroutines:      get_number                                   *
 *                                                                           *
 *      Public/Global Variables:                                             *
 *                              line - user input buffer                     *
 *                                                                           *
 *      Description:                                                         *
 *              This subroutine allows the user to edit the partition        *
 *              table passed to it.  It steps through each entry,            *
 *              displays it, and accepts replacement entries if specified.   *
 *                                                                           *
 *****************************************************************************/

askpartitions(lp)
        register struct dk_label *lp;
{
        register int i, first;
        register char *ptr;
        register struct dk_map *current_partition;
        daddr_t mb_til_end;     /* Mbytes left to allocate */
        daddr_t start_block;    /* starting block # for a partition */
        int ret;                /* generic integer function return value */
        register int done;      /* Loop terminater */
        char partition;         /* current partition name */
        int inp;
#ifdef  HOG_PARTITION
        extern void new_hog();
#endif

        clriwin();
        clrowin();

        /* If the # of heads, # of cylinders, or # of sectors per track
         * are 0, it doesn't make sense to change the partitions.
         * Display an error message to the screen and return.
         */
        if (lp->dkl_nhead == 0 || lp->dkl_nsect == 0 || lp->dkl_ncyl == 0)
        {
                wmove(stdscr, LINES - 4, 0);
                waddstr(stdscr, "    WARNING:  Incomplete geometry information.
                waddstr(stdscr, "    # of heads, # of cylinders, or # of sectors
                waddstr(stdscr, "    the l command and correct this problem.");
                wrefresh(stdscr);
                pak();
                return;
```



```
        }

        /*
         *      Have the user enter the "free space hog" partition
         */
#ifdef  HOG_PARTITION
        print_partitions(lp, 5);
        for ( ;; ) {
                wmove(stdscr, LINES - 2, 0);
                wclrtoeol(stdscr);
                waddstr(stdscr, "Enter the free space hog partition [n<one>] : "
                inp = inp_char(stdscr);
                if ( (inp == '\n') || (inp == '\r') )
                        break;
                if ( (isalpha(inp)) && (isupper(inp)) )
                        inp = tolower(inp);

                if ( inp == (int)'n' ) {
                        /* User doesn't want a free space hog partition */
                        Hog = NO_HOG;
                        break;
                }
                else {
                        Hog = inp - NBASE;
                        if ( (Hog >= MINPART) && (Hog <= MAXPART) ) {
                                /* Got a good one */
                                break;
                        }
                        else
                                printw("Invalid partition '%c'", (char)inp);
                }
        }
        new_hog();
#endif  /* HOG_PARTITION */

        for ( ;; ) {
                print_partitions(lp, 5);
                move(LINES - 3, 0);
                clrtoeol();
                first = TRUE;

                /*
                 *      Have the user pick a partition to change
                 */

                do {
                        if (!first) {
                                wmove(stdscr, LINES - 1, 0);
                                waddstr(stdscr, "Illegal partition, try again");
                        }
                        first = FALSE;
                        wmove(stdscr, LINES - 2, 0);
                        wclrtoeol(stdscr);
                        waddstr(stdscr, "Partition to change, <CR> when done: ")
                        line[0] = inp_char(stdscr);
                        if (line[0] == '\n' || line[0] == '\r')
```

```
                        return;
                if (isalpha(line[0])) {
                        if (isupper(line[0]))
                                line[0] = tolower(line[0]);
                }
        } while (line[0] < (NBASE + MINPART)
                        || line[0] > (NBASE + (MAXPART - 1)));

        partition = line[0];
        wmove(stdscr, LINES - 2, 0);
        clrtoeol();
        wmove(stdscr, LINES - 1, 0);
        clrtoeol();

        /* Point to the correct partition in the label */

        current_partition = &lp->dkl_map[partition - NBASE];

        /* Print out the partitions */

        print_partitions(lp, 5);
        move(LINES - 3, 0);
        clrtoeol();
        move(LINES - 3, 0);
        printw("Partition %c", partition);

        /* Loop until partition is "approved" by user */

        for ( done = 0; !done; ) {

                /*******************/
                /* Partition loop */
                /*******************/

                /* Pick a starting cylinder */

                for ( ;; ) {
                        move(LINES - 2, 0);
                        clrtoeol();
                        move(LINES - 2, 0);
                        printw("Enter the starting cylinder (0-%d, q<uit
                                        lp->dkl_ncyl - 1);
                        if ( (ret = get_number(&(current_partition->dkl_
                                continue; /* Error occured */

                        if ( ret == 3 ) {
                                /* User entered 'q' to quit */
                                done = 1;
                                break;
                        }

                        if ( (current_partition->dkl_cylno >= 0)
                                        && (current_partition->dkl_cylno
                                break;
                }
```



```
                /* Print out the partitions */
                print_partitions(lp, 5);
                move(LINES - 3, 0);
                printw("Partition %c", partition);
                start_block = (daddr_t)current_partition->dkl_cylno * Cb

                /* Pick a size */

                for ( ; !done ; ) {
                        /* Calculate Mbytes from start_block to end */
                        mb_til_end = (daddr_t)(((Blocks - start_block)
                                * SECTSIZE + (MEG-1)) / MEG);
                        move(LINES - 2, 0);
                        clrtoeol();
                        move(LINES - 2, 0);
                        printw("Enter the size in megabytes (0-%ld, r<es
                        /* current_partition->dkl_nblk gets filled by ge
                         * user entered Mbyte value.  If the user entere
                         * disk, it is set to mb_til_end.
                         */
                        if ( (ret = get_number(&(current_partition->dkl_
                                continue; /* Error occured */
                        if ( ret == 2 )
                                /* Entered 'return', no change */
                                break;
                        if ( ret == 3 ) {
                                /* Entered 'quit' */
                                done = 1;
                                break;
                        }

                        /* If user entered 'r' for rest of disk, calcula
                         * nblk by using total number of blocks - starti
                         */
                        if ( current_partition->dkl_nblk == mb_til_end )
                                current_partition->dkl_nblk = Blocks - s
                        }
                        else if ( (current_partition->dkl_nblk >= 0)
                                        && (current_partition->dkl_nblk
                                current_partition->dkl_nblk = ((((curren
                                        * MEG)/SECTSIZE +Cblocks - 1)/Cb

                                if (current_partition->dkl_nblk > (Block
                                        current_partition->dkl_nblk = Bl
                        } else
                                continue;

                        /* Print out the partitions */

                        print_partitions(lp, 5);
                        move(LINES - 3, 0);
                        printw("Partition %c", partition);
                        break;
                }
                move(LINES - 2, 0);
                clrtoeol();
```



```
                        for ( ; !done ; ) {
                                /* Loop until user likes the partition */

#ifdef  HOG_PARTITION
                                new_hog();
#endif
                                print_partitions(lp, 5);
                                move(LINES - 3, 0);
                                printw("Partition %c", partition);
                                move(LINES - 2, 0);
#ifdef NO_FLOATS
{
                                long tmptmp = BTOMEG(current_partition->dkl_nblk
                                char rem[100];
                                sprintf(rem, "%02ld", ((tmptmp % MEG) * 100) / M
                                printw("%ld blocks (%ld.%.2s Mbytes) OK (y/+/-/n
                                        current_partition->dkl_nblk,
                                        tmptmp / MEG, rem);
}
#else   /* !NO_FLOATS */
                                printw("%ld blocks (%.2f Mbytes) OK (y/+/-/n)? "
                                        current_partition->dkl_nblk,
                                        BTOMEG(current_partition->dkl_nblk));
#endif  /* NO_FLOATS */
                                ptr = line;
                                *ptr = inp_char(stdscr);
                                if ( (*ptr == 'n') || (*ptr == 'N')) {
                                        /* User doesn't like partition - wants t
                                        break;
                                }
                                else if ((*ptr == '\n') || (*ptr == 'Y') || (*pt
                                        /* User likes this one - done */
                                        done = 1;

                                        /* Recalculate the "hog" partition */
#ifdef  HOG_PARTITION
                                        new_hog();
#endif
                                        break;
                                }
                                else if ('+' == *ptr) {
                                        /* Increase by one cylinder block and re
                                        if ((current_partition->dkl_nblk + Cbloc
                                                current_partition->dkl_nblk += C
                                        }
                                }
                                else if ('-' == *ptr) {
                                        /* Decrease by one cylinder block and re
                                        if ((current_partition->dkl_nblk - Cbloc
                                                current_partition->dkl_nblk -= C
                                }
                        }
                }
        }
        /*NOTREACHED*/
```



```
}
```

```
/************************************************************************
 *                                                                      *
 *      print_partitions                                                *
 *                                                                      *
 *      Function:                                                       *
 *              Prints out the partition table plus added info          *
 *                                                                      *
 *      Inputs:                                                         *
 *              lp      - pointer to the label                          *
 *              start_line - line to start printing at                  *
 *                                                                      *
 *      Outputs:                                                        *
 *              void                                                    *
 *                                                                      *
 ************************************************************************/

void
print_partitions(lp, start_line)
        register struct dk_label *lp;
        int start_line;
{
        register int current_line;      /* Used to display more information
                                         * after the usual partition info */
        char display_line[200];         /* Local storage for display line */


        /* First display the partition table */

        current_line = showpartitions(lp->dkl_map, start_line);
        current_line++;

        /* Print out some additional info */

#ifdef NO_FLOATS
{
        char rem[100];
        sprintf(rem, "%02ld", ((Capacity % MEG) * 100) / MEG);
        sprintf(display_line,
                "Disk capacity: %5ld.%.2s Mbytes      Total cylinders: %5d",
                        Capacity / MEG, rem, lp->dkl_ncyl);
}
#else
        sprintf(display_line,
                "Disk capacity: %7.2f Mbytes      Total cylinders: %5d",
                        Capacity, lp->dkl_ncyl);
#endif  /* NO_FLOATS */
        center_line(current_line, display_line);
        current_line++;

        refresh();
}
```



```
/************************************************************************
*                                                                       *
*       Subroutine:             setpartitions                           *
*                                                                       *
*       Calling Sequence:       setpartitions()                         *
*                                                                       *
*       Called by:              dopartitions                            *
*                                                                       *
*       Calling Parameters:     None                                    *
*                                                                       *
*       Local Variables:                                                *
*                       i - looping                                     *
*                       file - to open the device file for each partition       *
*                       cp - pointer to partition character of device file name*
*                                                                       *
*       Calls Subroutines:                                              *
*                       ioctl - to set partition information for each partition *
*                                                                       *
*       Public/Global Variables:                                        *
*                       diskname - device file name of device currently in use  *
*                                                                       *
*       Description:                                                    *
*               This subroutine passes the partition information from   *
*               "map" to the device driver.                             *
*                                                                       *
************************************************************************/

setpartitions(map)
register struct dk_map *map;
{
        register int i;
        register int file;
        register char *cp;
        char savechar;

        if (diskname[strlen(diskname) - 3] != 's') {
                wmove(stdscr, 8, 0);
                waddstr(stdscr, "Partitions are only implemented for normal disk
                return;
        }
        cp = &diskname[strlen(diskname)-1];
        savechar = *cp;
        for (i = 0; i < NDKMAP; i++) {
                *cp = 'a' + i;
                if ((file = open(diskname, 2)) < 0)
                        continue;
                errno = 0;
                if (ioctl(file, DKIOCSPART, &map[i]) < 0) {
                        wmove(stdscr, 4, 0);
                        wprintw(stdscr, "\nDKIOCSPART: %s", sys_errlist[errno]);
                }
                close(file);
        }
        pak();
        *cp = savechar; /* back to the first partition */
```



```
        opendevice(diskname);   /* So as to refresh the possible size change. */
        return;
}
```



```
/************************************************************************
*                                                                       *
*       Subroutine:             getpartitions                           *
*                                                                       *
*       Calling Sequence:       getpartitions()                         *
*                                                                       *
*       Called by:              dopartitions                            *
*                                                                       *
*       Calling Parameters:     None                                    *
*                                                                       *
*       Local Variables:                                                *
*                       i - looping                                     *
*                       file - to open the device file for each partition       *
*                       cp - pointer to partition character of device file name *
*                                                                       *
*       Calls Subroutines:                                              *
*                       ioctl - to set partition information for each partition  *
*                                                                       *
*       Public/Global Variables:                                        *
*                       diskname - device file name of device currently in use  *
*                                                                       *
*       Description:                                                    *
*               This subroutine gets the partition information from     *
*               the device driver and places it in the array of         *
*               structures pointed to by "map".                         *
*                                                                       *
************************************************************************/

getpartitions(map)
register struct dk_map *map;
{
        register int i, max;
        register int file;
        register char *cp;
        char savepart;
        int good = -1;

        cp = &diskname[strlen(diskname)-1];
        savepart = *cp;
        max = diskname[strlen(diskname) - 3] == 's' ? NDKMAP : 1;
        if (max == 1) {
                errno = 0;
                if (ioctl(disk, DKIOCGPART, &map[0]) < 0) {
                        wmove(stdscr, 4, 0);
                        wprintw(stdscr, "\nDKIOCGPART: %s\n", sys_errlist[errno]
                        return(-1);
                } else
                        return;
        }

        for (i = 0; i < max; i++) {
                *cp = 'a' + i;
                if ((file = open(diskname, 2)) < 0) {
                        map[i].dkl_cylno =  map[i].dkl_nblk = 0;
                        continue;
```

```
                }
                errno = 0;
                if (ioctl(file, DKIOCGPART, &map[i]) < 0) {
                        wmove(stdscr, 4, 0);
                        wprintw(stdscr, "\nDKIOCGPART: %s\n", sys_errlist[errno]
                        *cp = savepart;
                        return(-1);
                }
                good = 0;
                close(file);
        }
        *cp = savepart;
        return(good);
}

/*****************************************************************************
*                                                                            *
*       Tape Positioning Subroutines:   sfm(), reten(), load(), unload(),    *
*                                       ers_tape(), wfm(), rew()             *
*                                                                            *
*       Calling Sequence:       routine()                                    *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Calls Subroutines:      ioctl                                        *
*                                                                            *
*       Description:                                                         *
*                The subroutines listed above are all for tape devices       *
*                for positioning the tape, erasing, or writing filemarks.    *
*                                                                            *
*****************************************************************************/

/* Search a filemark - this routine will search forward one filemark. */
sfm()
{
        struct mtop mt;

        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                return;
        }

        waddstr(stdscr, "Enter number of filemarks to search:  ");
        wrefresh(stdscr);
        input(stdscr, line);
        mt.mt_count = atoi(line);
        mt.mt_op = MTFSF;

        errno = 0;
        if (ioctl(disk, MTIOCTOP, &mt) < 0) {
                wprintw(stdscr, "RFIOCSFM: %s\n", sys_errlist[errno]);
                return(-1);
        }
```



```
        return(0);
}

/* Retension the tape */
reten()
{
        struct mtop mt;

        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        mt.mt_op = MTRETEN;
        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, MTIOCTOP, &mt) < 0) {
                wprintw(stdscr, "RFIOCRETEN: %s\n", sys_errlist[errno]);
                pak();
                return(-1);
        }
        return(0);
}

/* This routine causes the SCSI load command to be issued to the drive. */
load()
{
        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, RFIOCLOAD, 0) < 0) {
                wprintw(stdscr, "RFIOCLOAD: %s\n", sys_errlist[errno]);
                pak();
                return(-1);
        }
        return(0);
}

/* This routine causes the SCSI unload command to be issued to the drive. */
unload()
{
        struct mtop mt;

        clriwin();
```



```
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        mt.mt_op = MTOFFL;
        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, MTIOCTOP, &mt) < 0) {
                wprintw(stdscr, "RFIOCUNLOAD: %s\n", sys_errlist[errno]);
                pak();
                return(-1);
        }
        return(0);
}


/* This routine causes the tape to be erased from the current tape position
 * to the end of tape. */
ers_tape()
{
        struct mtop mt;

        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        wmove(stdscr, 11, 0);
        waddstr(stdscr, "Enter 'y' if you are ready to start the erase:  ");

        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                pak();
                return;
        }

        waddstr(stdscr, "Yes");

        mt.mt_op = MTERASE;
        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, MTIOCTOP, &mt) < 0) {
                wprintw(stdscr, "RFIOCERASE: %s\n", sys_errlist[errno]);
                pak();
                return(-1);
        }
        return(0);
```



```
}

/* This routine will cause one filemark to be written to tape. */
wfm()
{
        struct mtop mt;

        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        mt.mt_op = MTWEOF;
        mt.mt_count = 1;
        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, MTIOCTOP, &mt) < 0) {
                wprintw(stdscr, "WFM: %s\n", sys_errlist[errno]);
                pak();
                return(-1);
        }
        return(0);
}

/* This routine causes the tape to be rewound.*/
rew()
{
        struct mtop mt;

        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        mt.mt_op = MTREW;
        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, MTIOCTOP, &mt) < 0) {

                wprintw(stdscr, "REWIND: %s\n", sys_errlist[errno]);
                pak();
                return(-1);
        }
        return(0);
}

/*****************************************************************************
*                                                                            *
```

```
*       Subroutine:     identify(),                                             *
*                                                                               *
*       Calling Sequence:       identify()                                      *
*                                                                               *
*       Called by:              main                                            *
*                                                                               *
*       Calls Subroutines:      ioctl                                           *
*                                                                               *
*       Description:                                                            *
*               This routine issues the Rimfire 35XX identify command which     *
*               returns the firmware revision of the adapter.                   *
*                                                                               *
********************************************************************************/

identify()
{
        RETID   id;

        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                pak();
                return;
        }

        wrefresh(stdscr);
        errno = 0;
        if (ioctl(disk, RFIOCIDENT,&id) < 0) {
                waddstr(stdscr, "RFIOCIDENT: %s", sys_errlist[errno]);
                pak();
                return(-1);
        }

        wprintw(stdscr, "\tFW  rev: %02d\n", id.fwrev & 0xFF);
        wprintw(stdscr, "\tEng rev: %02d\n", id.engrev & 0xFF);
        wprintw(stdscr, "\tFW date: %s %2d,%2d\n",
                month[id.month - 1], id.day, id.year);
        pak();
        return(0);
}
/********************************************************************************
*                                                                               *
*       Subroutine:             showlabel                                       *
*                                                                               *
*       Calling Sequence:       showlabel()                                     *
*                                                                               *
*       Called by:              main                                            *
*                                                                               *
*       Calling Parameters:     None                                            *
*                                                                               *
*       Local Variables:                                                        *
*                       lp - pointer to label                                   *
*                       wp - used to step through configuration info            *
*                       cksum - used to calculate the checksum of the label     *
```



```
*                       i - looping                                             *
*                                                                               *
*       Calls Subroutines:  showpartitions - display partition information      *
*                                            from the label                     *
*                                                                               *
*       Public/Global Variables:                                                *
*                               label - the current label                       *
*                                                                               *
*       Description:                                                            *
*               This subroutine displays the current label.                     *
*                                                                               *
********************************************************************************/

showlabel()
{
        register struct dk_label *lp = &label;
        register int i;
        register unsigned short *wp;
        unsigned short cksum;
        register int field_count, current_line, shift, start_line;

        clriwin();
        clrowin();

        current_line = 4;
        wmove(stdscr, current_line, 0);
        wprintw(stdscr, "<%s>\n", lp->dkl_asciilabel);
        current_line++;

        cksum = 0;
        for (i = 0, wp = (unsigned short *)lp; i < sizeof(*lp)/2-1; i++)
                cksum ^= *wp++;
        if (cksum != lp->dkl_cksum)
                wprintw(stdscr, "Checksum error in label, was 0x%04x, should be
                        cksum, lp->dkl_cksum);
        else
                waddstr(stdscr, "Checksum in label is ok.\n");
        current_line++;

        if (lp->dkl_magic != DKL_MAGIC)
                wprintw(stdscr, "Wrong magic number in label: 0x%04x\n", lp->dkl
        else

                waddstr(stdscr, "Magic number in label is ok.\n");
        current_line++;
        for ( field_count = 0; fieldnames[field_count] != NULL; field_count++ )
                ;
        wp = &lp->dkl_apc;
        start_line = current_line;
        shift = 0;
        for (i = 0; fieldnames[i] != NULL; i++) {
                if ( current_line == (start_line + (field_count / 2)) ) {
                        current_line = start_line;
                        shift = COLS / 2;
                }
                wmove(stdscr, current_line, shift);
```



```
                wprintw(stdscr, "%-20s %d\n", fieldnames[i], *wp++);
                current_line++;
        }
        showpartitions(lp->dkl_map, (start_line + (field_count / 2)) + 1);
        pak();
}
/********************************************************************************
*                                                                               *
*       Subroutine:             readlabel                                       *
*                                                                               *
*       Calling Sequence:       readlabel()                                     *
*                                                                               *
*       Called by:              main                                            *
*                                                                               *
*       Calling Parameters:     None                                            *
*                                                                               *
*       Local Variables:        None                                            *
*                                                                               *
*       Calls Subroutines:      None                                            *
*                                                                               *
*       Public/Global Variables:                                                *
*                       disk - file descriptor of device file to read label from *
*                       label - disk label structure to read label into         *
*                                                                               *
*       Description:                                                            *
*               This subroutine attempts to read the label from the device file *
*               currently open.                                                 *
*                                                                               *
********************************************************************************/

readlabel()
{
        wmove(stdscr, 5, 0);
        wclrtoeol(stdscr);
        wmove(stdscr, 6, 0);
        wclrtoeol(stdscr);
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "Please use 'o' to open a disk device file");
                return;
        }
        lseek(disk, 0L, 0);
        if (read(disk, (char *)&label, sizeof(struct dk_label)) < 0) {
                wprintw(stdscr, "Read: %s", sys_errlist[errno]);
                return;
        }
        if (label.dkl_magic != DKL_MAGIC)
                wprintw(stdscr, "Warning: Bad magic number (0x%x) in label", lab
        showlabel();
}
```



```
/********************************************************************************
*                                                                               *
*       Subroutine:             writelabel                                      *
*                                                                               *
*       Calling Sequence:       writelabel()                                    *
*                                                                               *
*       Called by:              main                                            *
*                                                                               *
*       Calling Parameters:     None                                            *
*                                                                               *
*       Local Variables:        None                                            *
*                                                                               *
*       Calls Subroutines:      None                                            *
*                                                                               *
*       Public/Global Variables:                                                *
*                       disk - file descriptor of device file to write label to *
*                       label - disk label structure to write label from        *
*                                                                               *
*       Description:                                                            *
*               This subroutine attempts to write the current label to          *
*               the device file currently open.                                 *
*                                                                               *
********************************************************************************/

writelabel()
{
        clriwin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "Please use 'o' to open a disk device file");
                return;
        }
        waddstr(stdscr,"Ready to write label, enter 'y' to proceed: ");
        line[0] = inp_char(stdscr);
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                return;
        }
        waddstr(stdscr, "Write");
        lseek(disk, 0L, 0);
        if (write(disk, (char *)&label, sizeof(struct dk_label)) < 0) {
                wmove(stdscr, 8, 0);
                wprintw(stdscr, "Write: %s", sys_errlist[errno]);
                return;
        }
        return;
}
/********************************************************************************
*                                                                               *
*       Subroutine:             dolabel                                         *
*                                                                               *
*       Calling Sequence:       dolabel()                                       *
*                                                                               *
*       Called by:              main                                            *
*                                                                               *
```

```
*       Calling Parameters:     None                                    *
*                                                                       *
*       Local Variables:                                                *
*                       wp - used to step through configuration info    *
*                       lp - pointer to label                           *
*                       type - type of label chosen                     *
*                       i - looping                                     *
*                                                                       *
*       Calls Subroutines:                                              *
*                        ioctl - to pass the label command to the driver*
*                        askpartitions - allow user to modify partition *
*                                        tables in the label            *
*                        showpartitions - display partitions in the label*
*                                                                       *
*       Public/Global Variables:                                        *
*            fieldnames - names of the disk configuration fields in the label *
*                                                                       *
*       Description:                                                    *
*             This subroutine displays the available standard labels    *
*             and allows the user to select one or edit whichever label *
*             is currently selected.                                    *
*                                                                       *
************************************************************************/

dolabel()
{
        read_cap cap;
        dword *lba;
        dword ucyl;
        register int i;
        register unsigned short *wp;
        register struct dk_label *lp;
        int type;
        int     x, y;
        page_3 p3;
        page_4 p4;
        inq_data inqdat;
        int offset;
        dword ncyl;
        int nhead;
        unsigned short apc, acyl, nsect, intrlv;
        char ascii_lab[128];
        int current_line, start_line, shift, field_count;

        clriwin();
        clrowin();

        p3.page_code = 3;
        p4.page_code = 4;
        y = 11;
        x = 0;
        wmove(stdscr, 11, 0);
        wclrtoeol(stdscr);
        wmove(stdscr, 11, 0);
        waddstr(stdscr, "0 - edit current label\n");
        waddstr(stdscr, "1 - get disk geometry from the disk thru SCSI mode sens
```



```
        wrefresh(stdscr);

choose:
        wmove(stdscr, 4, 0);
        wclrtoeol(stdscr);
        wmove(stdscr, 4, 0);
        waddstr(stdscr, "Please enter your choice: ");
        input(stdscr, line);
        if (line[0] == 'q')
                return;
        if (line[0] == '\0'
         || (type = atoi(line)) < 0
         || type > 1) {
                wmove(stdscr, 5, 0);
                waddstr(stdscr, "Bad choice, please try again.");
                goto choose;
        }
        lp = &label;
        clrowin();
        if (type == 0) {
label:
                wmove(stdscr, 6, 0);
                waddstr(stdscr, "Please enter an ascii label for the disk (less
                wprintw(stdscr, "Press RETURN to keep:\n<%s>\n<", lp->dkl_asciil
                wclrtoeol(stdscr);
                wmove(stdscr, 9, 1);
                input(stdscr, line);
                waddch(stdscr, '>');
                wrefresh(stdscr);
                if (line[0] != '\0') {
                        if (strlen(line) > 127) {
                                wmove(stdscr, 10, 0);
                                waddstr(stdscr, "Label is too long, please try a
                                goto label;
                        } else
                                strcpy(lp->dkl_asciilabel, line);
                }

                newconf(lp);

                /*
                 *      Calculate some drive characteristics
                 */

                Blocks = (long)lp->dkl_nhead * lp->dkl_ncyl * lp->dkl_nsect;
                Cblocks = (long)lp->dkl_nhead * lp->dkl_nsect;
                if ( Blocks <= 0 )
                        Blocks = 1;
                if ( Cblocks <= 0 )
                        Cblocks = 1;
                Capacity = BTOMEG(Blocks);

                clriwin();
                clrowin();

                /*
```



```
                 *      Determine where, on the screen, the partitions
                 *      should be printed
                 */

                for ( field_count = 0; fieldnames[field_count] != NULL; field_co
                        ;
                wp = &lp->dkl_apc;
                shift = 0;
                start_line = current_line = 4;
                for (i = 0; fieldnames[i] != NULL; i++) {
                        if ( current_line == (start_line + (field_count / 2)) )
                                current_line = start_line;
                                shift = COLS / 2;
                        }
                        wmove(stdscr, current_line, shift);
                        wprintw(stdscr, "%-20s %d", fieldnames[i], *wp++);
                        current_line++;
                }
                showpartitions(lp->dkl_map, (start_line + (field_count / 2)) + 1
                askpartitions(lp);

        } else {

                /* Issue a mode sense for page 4 to get the number of heads. */
                if (ioctl(disk, RFIOCMDSEN4, &p4))
                {
                        wprintw(stdscr, "RFIOCMDSEN4: %s\n", sys_errlist[errno])
                        wrefresh(stdscr);
                        pak();
                        return;
                }

                ncyl = p4.ncyl_b1;
                ncyl = (ncyl << 8) | p4.ncyl_b2;
                nhead = p4.nhead;

                /* Issue a mode sense for page 3 to get number of sectors
                 * per track, interleave value, and alternates per cylinder
                 */
                if (ioctl(disk, RFIOCMDSEN3, &p3))
                {
                        wprintw(stdscr, "RFIOCMDSEN3: %s\n", sys_errlist[errno])
                        wrefresh(stdscr);
                        pak();
                        return;
                }

                if(p3.alttpzone != 0)
                        apc = p3.altspzone / p3.alttpzone;
                else
                        apc = 0;
                apc = apc * nhead;
                if(p3.alttpvol)
                        acyl = p3.alttpvol / nhead;
                nsect = p3.spt;
                /* If number of sectors is equal to 0, give it a
```



```
                   value of 35 and warn the user that this is an
                   assumed value. */
                if(nsect == 0)
                {
                        nsect = 35;
                        wmove(stdscr, LINES - 2, 0);
                        waddstr(stdscr, "WARNING:  Number of sectors per track f
                        wrefresh(stdscr);
                        pak();
                }

                intrlv = p3.interleave;

                /* Issue a read capacity command to calculate the number of
                 * useable cylinders.
                 * useable cyl = ((read cap blk#) / (# of hds * sec/track)).
                 */

                /* lba = number of blocks after read capacity cmd issued. */
                lba = (dword *)cap.nblk;

                if (ioctl(disk, RFIOCRDCAP, &cap))
                {
                        /* Useable cylinders = total number of cylinders */
                        ucyl = ncyl;
                } else
                {
                        /* Calculate the number of useable cylinders. */
                        ucyl = (*lba / (nhead * nsect));
                }

                /* Issue an inquiry command to get the ascii device manufacturer
                 * and device model number to include in the ascii section of
                 * the label.
                 */
                if (ioctl(disk, RFIOCINQ, &inqdat))
                {
                        wprintw(stdscr, "RFIOCINQ: %s\n", sys_errlist[errno]);
                        wrefresh(stdscr);
                        pak();
                        return;
                }

                inqdat.vend_uniq[10] = 0;
                inqdat.vend_uniq[18] = 0;

                sprintf(ascii_lab, "%s-%s cyl %d alt %d hd %d sec %d apc %d", &i

                strcpy(lp->dkl_asciilabel, ascii_lab);
                lp->dkl_apc = apc;
                lp->dkl_gap1 = 0;
                lp->dkl_gap2 = 0;
                lp->dkl_intrlv = intrlv;
                lp->dkl_ncyl = ucyl;
                lp->dkl_acyl = acyl;
                lp->dkl_nhead = nhead;
```

```
		lp->dkl_nsect = nsect;

		/*
		 *	Now Calculate some drive characteristics
		 */

		Blocks = (long)lp->dkl_nhead * lp->dkl_ncyl * lp->dkl_nsect;
		Cblocks = (long)lp->dkl_nhead * lp->dkl_nsect;
		if ( Blocks <= 0 )
			Blocks = 1;
		if ( Cblocks <= 0 )
			Cblocks = 1;
		Capacity = BTOMEG(Blocks);

		/*
		 *	Determine where, on the screen, the partitions
		 *	should be printed
		 */

		for ( field_count = 0; fieldnames[field_count] != NULL; field_co
			;
		start_line = current_line = 4;
		wmove(stdscr, current_line, 0);
		wprintw(stdscr, "<%s>", lp->dkl_asciilabel);
		current_line++;
		wp = &lp->dkl_apc;
		start_line = current_line;
		shift = 0;
		for (i = 0; fieldnames[i] != NULL; i++) {
			if ( current_line == (start_line + (field_count / 2)) )
				current_line = start_line;
				shift = COLS / 2;
			}
			wmove(stdscr, current_line, shift);
			wprintw(stdscr, "%-20s %d", fieldnames[i], *wp++);
			current_line++;
		}
		showpartitions(lp->dkl_map, (start_line + (field_count / 2)) + 1
		askpartitions(lp);
		showlabel();
	}

#ifdef notyet
	wmove(stdscr, LINES - 2, 0);
	waddstr(stdscr, "Enter 'y' to write this partition table to the kernel:
	line[0] = inp_char(stdscr);
	if (line[0] != 'y') {
		waddstr(stdscr, "No");
		pak();
		return;
	}
	waddstr(stdscr, "Yes");
	wmove(stdscr, current_line, 0);

	setpartitions(lp->dkl_map);
#endif
```



```
	lp->dkl_magic = DKL_MAGIC;
	lp->dkl_cksum = 0;
	for (i = 0, wp = (unsigned short *)lp; i < sizeof(struct dk_label)/2-1;
		lp->dkl_cksum ^= *wp++;
	return;
}
/*****************************************************************************
*                                                                            *
*	Subroutine:		newconf                                      *
*                                                                            *
*	Calling Sequence:	newconf(lp)                                  *
*                                                                            *
*	Called by:		dolabel                                      *
*                                                                            *
*	Calling Parameters:	lp - pointer to label structure              *
*                                                                            *
*	Local Variables:                                                     *
*			wp - used to step through configuration info         *
*			i - looping                                          *
*                                                                            *
*	Calls Subroutines:                                                   *
*			ioctl - to pass the label command to the driver      *
*                                                                            *
*	Public/Global Variables:                                             *
*		fieldnames - names of the disk configuration fields in the label *
*                                                                            *
*	Description:                                                         *
*		This subroutine allow the user to change the current         *
*		setting of disk label.                                       *
*                                                                            *
*****************************************************************************/

newconf(lp)
register struct dk_label *lp;
{
	int	item, i;
	register unsigned short *wp;

	clriwin();
	clrowin();
	while (1) {
		wmove(stdscr, 11, 0);
		wp = &lp->dkl_apc;
		for (i = 0; fieldnames[i] != NULL; i++, wp++) {
			wmove(stdscr, 11 + i, 0);
			wprintw(stdscr, "%ld - %s <%d>\n", i + 1, fieldnames[i],
		}
		wmove(stdscr, 5, 0);
		wclrtoeol(stdscr);
		wmove(stdscr, 5, 0);
		wprintw(stdscr, "Enter item number to change, <CR> when done: ")
		input(stdscr, line);
		wmove(stdscr, 6, 0);
		wclrtoeol(stdscr);
		if (line[0] == '\0')
			break;
```



```
		item = atoi(line);

		wmove(stdscr, 8, 0);
		wclrtoeol(stdscr);
		wmove(stdscr, 7, 0);
		switch (item) {
		case 1:
			wprintw(stdscr, " 1.  # of spare sectors per cylinder:
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_apc = atoi(line);
			break;
		case 2:
			wprintw(stdscr, " 2.  Size of gap1:   %u\n", lp->dkl_gap
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_gap1 = atoi(line);
			break;
		case 3:
			wprintw(stdscr, " 3.  Size of gap2:   %u\n", lp->dkl_gap
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_gap2 = atoi(line);
			break;
		case 4:
			wprintw(stdscr, " 4.  Interleave factor:   %u\n", lp->dk
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_intrlv = atoi(line);
			break;
		case 5:
			wprintw(stdscr, " 5.  Number of cylinders:  %u\n", lp->d
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_ncyl = atoi(line);
			break;
		case 6:
			wprintw(stdscr, " 6.  Number of alt cylinders:  %u\n", l
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_acyl = atoi(line);
			break;
		case 7:
			wprintw(stdscr, " 7.  Number of heads:  %u\n", lp->dkl_n
```



```
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_nhead = atoi(line);
			break;
		case 8:
			wprintw(stdscr, " 8.  # of sectors/track:    %u\n", lp->
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_nsect = atoi(line);
			break;
		case 9:
			wprintw(stdscr, " 9.  Label location:  %u\n", lp->dkl_bh
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_bhead = atoi(line);
			break;
		case 10:
			wprintw(stdscr, " 10.  Physical partition #:  %u\n", lp-
			waddstr(stdscr, "new value or <CR> for no change: ");
			input(stdscr, line);
			if (line[0] == '\0')
				break;
			lp->dkl_ppart = atoi(line);
			break;
		default:
			wmove(stdscr, 6, 0);
			wprintw(stdscr, "Invalid item %d.\n", item);
			break;
		}
	}
}
/*****************************************************************************
*                                                                            *
*	Subroutine:		dogeometry                                   *
*                                                                            *
*	Calling Sequence:	dogeometry()                                 *
*                                                                            *
*	Called by:		main                                         *
*                                                                            *
*	Calling Parameters:	None                                         *
*                                                                            *
*	Local Variables:                                                     *
*			geometry - geometry information to pass to the driver*
*                                                                            *
*	Calls Subroutines:                                                   *
*				ioctl - to get geometry info from the driver *
*					and to pass new information to it    *
*                                                                            *
*	Public/Global Variables:                                             *
*				label - current label                        *
```

```
*                               line - buffer for user input                    *
*                                                                               *
*       Description:                                                            *
*            This subroutine asks whether to use geometry information as it     *
*            exists in the kernel or from the current label.  It then displays  *
*            the geometry and asks whether to send it to the kernel.            *
*                                                                               *
********************************************************************************/

dogeometry()
{
        struct dk_geom geometry;

        clriwin();
        clrowin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk before modifying the geo
                return;
        }
        waddstr(stdscr, "Enter 'k' to get disk geometry from the kernel,\n");
        waddstr(stdscr, "anything else will get it from the label: ");
        line[0] = inp_char(stdscr);
        if (line[0] == 'k') {
                waddstr(stdscr, "Kernel");
                if (ioctl(disk, DKIOCGGEOM, &geometry) < 0) {
                        wmove(stdscr, 7, 0);
                        wprintw(stdscr, "Get geometry: %s", sys_errlist[errno]);
                        return;
                }
        } else {
                waddstr(stdscr, "Label");

                geometry.dkg_ncyl = label.dkl_ncyl;
                geometry.dkg_acyl = label.dkl_acyl;
                geometry.dkg_bcyl = 0;
                geometry.dkg_nhead = label.dkl_nhead;
                geometry.dkg_bhead = label.dkl_bhead;
                geometry.dkg_nsect = label.dkl_nsect;
                geometry.dkg_intrlv = label.dkl_intrlv;
                geometry.dkg_gap1 = label.dkl_gap1;
                geometry.dkg_gap2 = label.dkl_gap2;
                geometry.dkg_apc = label.dkl_apc;
        }

        showgeometry(&geometry);
}
/*******************************************************************************
*                                                                              *
*       Subroutine:             showgeometry                                   *
*                                                                              *
*       Calling Sequence:       showgeometry(gp)                               *
*                                                                              *
*       Called by:              dogeometry, domap                              *
*                                                                              *
*       Calling Parameters:   gp - pointer to geometry information to display  *
```



```
*                                                                              *
*       Local Variables:        None                                           *
*                                                                              *
*       Calls Subroutines:      None                                           *
*                                                                              *
*       Public/Global Variables:None                                           *
*                                                                              *
*       Description:                                                           *
*                   This subroutine displays the disk geometry information     *
*                   passed to it.                                              *
*                                                                              *
********************************************************************************/

showgeometry(gp)
register struct dk_geom *gp;
{
        register int i;

        wmove(stdscr, 11, 0);
        wprintw(stdscr, "# of alternates per cylinder is %d\n", gp->dkg_apc);
        wmove(stdscr, 12, 0);
        wprintw(stdscr, "# of data cylinders is %d\n", gp->dkg_ncyl);
        wmove(stdscr, 13, 0);
        wprintw(stdscr, "# of alternate cylinders is %d\n", gp->dkg_acyl);
        wmove(stdscr, 14, 0);
        wprintw(stdscr, "cyl offset (for fixed head area) is %d\n", gp->dkg_bcyl
        wmove(stdscr, 15, 0);
        wprintw(stdscr, "# of heads is %d\n", gp->dkg_nhead);
        wmove(stdscr, 16, 0);
        wprintw(stdscr, "head offset (for Larks, etc.) is %d\n", gp->dkg_bhead);
        wmove(stdscr, 17, 0);
        wprintw(stdscr, "# of sectors per track is %d\n", gp->dkg_nsect);
        wmove(stdscr, 18, 0);
        wprintw(stdscr, "# of spare sectors per track is %d\n", gp->dkg_extra[0]
        wmove(stdscr, 19, 0);
        wprintw(stdscr, "interleave factor is %d\n", gp->dkg_intrlv);
        wmove(stdscr, 20, 0);
        wprintw(stdscr, "gap 1 size is %d\n", gp->dkg_gap1);
        wmove(stdscr, 21, 0);
        wprintw(stdscr, "gap 2 size is %d\n", gp->dkg_gap2);
}

/*******************************************************************************
*                                                                              *
*       Subroutine:             reset                                          *
*                                                                              *
*       Calling Sequence:       reset()                                        *
*                                                                              *
*       Called by:              main                                           *
*                                                                              *
*       Local Variables:        None                                           *
*                                                                              *
*       Calls Subroutines:      ioctl                                          *
*                                                                              *
*       Public/Global Variables:None                                           *
*                                                                              *
```



```
*       Description:                                                           *
*                   This subroutine resets the controller via ioctl call,      *
*                   then sets up the adapter by General Options, Unit Options, *
*                   and Start Command list.                                    *
*                                                                              *
********************************************************************************/

reset()
{
        int errco;

        clriwin();
        clrowin();
        wmove(stdscr, 11, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                return;
        }

        wrefresh(stdscr);

        wmove(stdscr, 11, 0);
        waddstr(stdscr, "The adapter will be reset even if devices are currently
        waddstr(stdscr, "Do not continue if there are mounted file systems or an
        waddstr(stdscr, "currently being accessed!!!!\n");
        waddstr(stdscr, "Do you want to continue, y or n? ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                return;
        }
        waddstr(stdscr, "Yes");

        wmove(stdscr, 15, 0);
        waddstr(stdscr, "Are you sure? ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] != 'y') {
                waddstr(stdscr, "No");
                return;
        }
        waddstr(stdscr, "Yes");
        wmove(stdscr, 15, 0);

        errno = 0;
        if (ioctl(disk, RFIOCRESET, 0)) {
                wprintw(stdscr, "RFIOCRESET: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
                return;
        }
        waddstr(stdscr, "RETURNDED from RFIOCRESET\n");
        wrefresh(stdscr);

        errno = 0;
        if (ioctl(disk, RFIOCGENOPT, 0)) {
                wprintw(stdscr, "RFIOCGENOPT: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
```



```
                return;
        }
        waddstr(stdscr, "RETURNDED from RFIOCGENOPT\n");
        wrefresh(stdscr);

        errno = 0;
        if (ioctl(disk, RFIOCUNITOPT, 0) < 0) {
                wprintw(stdscr, "RFIOCUNITOPT: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
                return;
        }
        waddstr(stdscr, "RETURNDED from RFIOCUNITOPT\n");
        wrefresh(stdscr);

        errno = 0;
        if (ioctl(disk, RFIOCCMDLST, 0)) {
                wprintw(stdscr, "RFIOCMDLST: %s\n", sys_errlist[errno]);
                return;
        }

        disk = -1;
        waddstr(stdscr, "Calling opendevice\n");
        wrefresh(stdscr);
        if (opendevice(diskname) < 0) {
                wrefresh(stdscr);
                endwin();
                printf("\n");
                error("Can't open  device after adapter reset.");
                exit(-1);
        }

        waddstr(stdscr, "Returned from opendevice -returning\n");
        wrefresh(stdscr);

        return(0);
}

/*******************************************************************************
*                                                                              *
*       Subroutine:             domodsen                                       *
*                                                                              *
*       Calling Sequence:       domodsen()                                     *
*                                                                              *
*       Called by:              main                                           *
*                                                                              *
*       Local Variables:        None                                           *
*                                                                              *
*       Calls Subroutines:      ioctl                                          *
*                                                                              *
*       Public/Global Variables:None                                           *
*                                                                              *
*       Description:                                                           *
*                   This subroutine displays the disk geometry information     *
*                   passed to it.                                              *
*                                                                              *
********************************************************************************/
```

```
domodsen()
{
        page_3 p3;
        page_4 p4;
        inq_data inqdat;
        int offset;
        dword ncyl;
        int nhead;
        unsigned short apc, acyl, nsect, intrlv;
        char ascii_lab[128];
        struct dk_label *lp;

        p3.page_code = 3;
        p4.page_code = 4;

        clriwin();
        clrowin();

#ifdef notyet
        if (ioctl(disk, RFIOCMDSEN4, &p4))
        {
                wprintw(stdscr, "RFIOCMDSEN4: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
                return;
        }

        ncyl = p4.ncyl_b1;
        ncyl = (ncyl << 8) | p4.ncyl_b2;
        nhead = p4.nhead;

        wmove(stdscr, 11, 0);
        wprintw(stdscr, "# of cyl = %d  # of hds = %d\n", ncyl, p4.nhead);
        wrefresh(stdscr);

#endif
        if (ioctl(disk, RFIOCMDSEN3, &p3))
        {
                wprintw(stdscr, "RFIOCMDSEN3: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
                return;
        }

        if(p3.alttpzone != 0)
                apc = p3.altspzone / p3.alttpzone;
        else
                apc = 0;
        apc = apc * nhead;
        if(p3.alttpvol)
                acyl = p3.alttpvol / nhead;
        nsect = p3.spt;
        intrlv = p3.interleave;

        wmove(stdscr, 13, 0);
        wprintw(stdscr, "apc = %d  nsect = %d intrlv = %d\n", apc, nsect, intrlv
```



```
        wrefresh(stdscr);


        if (ioctl(disk, RFIOCINQ, &inqdat))
        {
                wprintw(stdscr, "RFIOCINQ: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
                return;
        }
        inqdat.vend_uniq[10] = 0;
        inqdat.vend_uniq[18] = 0;

        wprintw(stdscr, "%s  %s\n", &inqdat.vend_uniq[3], &inqdat.vend_uniq[11])
        wrefresh(stdscr);

        sprintf(ascii_lab, "%s-%s cyl %d alt %d hd %d sec %d apc %d", &inqdat.ve

        wprintw(stdscr, "%s\n", ascii_lab);
        wrefresh(stdscr);

        label.dkl_apc = apc;
        label.dkl_gap1 = 0;
        label.dkl_gap2 = 0;
        label.dkl_intrlv = intrlv;
        label.dkl_ncyl = ncyl;
        label.dkl_acyl = acyl;
        label.dkl_nhead = nhead;
        label.dkl_nsect = nsect;

        lp = &label;
}
/*****************************************************************************
*                                                                            *
*       Subroutine:             dodensity                                    *
*                                                                            *
*       Calling Sequence:       dodensity()                                  *
*                                                                            *
*       Called by:              main                                         *
*                                                                            *
*       Local Variables:        None                                         *
*                                                                            *
*       Calls Subroutines:      ioctl                                        *
*                                                                            *
*       Public/Global Variables:None                                         *
*                                                                            *
*       Description:                                                         *
*                   This subroutine displays the disk geometry information   *
*                   passed to it.                                            *
*                                                                            *
*****************************************************************************/

dodensity()
{
        mode_sel parm_list;
        int menu_select;
        int first;
```



```
        int i;


        clriwin();
        clrowin();

        if (ioctl(disk, RFIOCMDSEN, &parm_list))
        {
                wprintw(stdscr, "RFIOCMDSEN: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
                pak();
                return;
        }

        speed = (parm_list.byte2 & 0x0F);
        buffered = ((parm_list.byte2 >> 4) & 0x07);
        density = parm_list.density_code;
        blklen = ((parm_list.blklen[0] << 16) |
                  (parm_list.blklen[1] << 8) |
                  parm_list.blklen[2]);

        /* Display mode select menu */
        for(i=0; i < 4; i++) {
                wmove(stdscr, 13 + i, 0);
                wprintw(stdscr, "%-36.36s  %x\n", menu[i], *params[i]);
        }

        wmove(stdscr, 18, 0);
        wprintw(stdscr, "Number of blocks = %02x%02x%02x\n", parm_list.nblk[0],
        wrefresh(stdscr);

again:
        do
        {
                first = 1;
                do
                {
                        if (!first)
                        {
                                wmove(stdscr, 10, 0);
                                waddstr(stdscr, "Illegal Entry, try again - ");
                        }
                        first = FALSE;
                        wmove(stdscr, 10, 32);
                        waddstr(stdscr, "Number to change, <CR> when done: ");
                        line[0] = inp_char(stdscr);
                        if (line[0] == '\r' || line[0] == '\n')
                                goto done;
                        if (isalpha(line[0]))
                        {
                                if (isupper(line[0]))
                                        line[0] = tolower(line[0]);
                        }
                } while (line[0] < '1' || line[0] > '4');

                menu_select = (line[0] - '1');
```



```
                wmove(stdscr, 13 + menu_select, 38);
                input(stdscr, line);
                sscanf(line, "%x", params[menu_select]);

        } while(TRUE);

done:

        parm_list.byte0 = 0;
        speed &= 0xf;
        buffered &= 0x7;
        parm_list.byte2 = ((buffered << 4) | speed);

        parm_list.nblk[0] =  parm_list.nblk[1] = parm_list.nblk[2] = 0;
        parm_list.density_code = density;

        if (ioctl(disk, RFIOCMDSEL, &parm_list))
        {
                wprintw(stdscr, "RFIOCMDSEL: %s\n", sys_errlist[errno]);
                wrefresh(stdscr);
                pak();
                return;
        }
        pak();
}

/*****************************************************************************/

defect()
{

        def_list def_lst;
        dword   *lba, *len;
        read_cap        cap;
        int i;
        int j;
        int deflst_sel;
        int list_fmt;
        int good_deflist = 0;

        clriwin();
        clrowin();
        wmove(stdscr, 5, 0);
        if (disk == -1) {
                waddstr(stdscr, "You must open the disk first.");
                return;
        }
        /* Don't read defect list unless user wants to */
        def_lst.list[0] = 0;

        lba = (dword *)&cap;
        len = (dword *)cap.blklen;
        *lba = 0;

        errno = 0;
```

```
        if (ioctl(disk, RFIOCRDCAP, &cap) < 0) {
                wprintw(stdscr, "RFIOCRDCAP: %s", sys_errlist[errno]);
                return(-1);
        }

        waddstr(stdscr, "Enter 'y' to include a defect list during format: ");
        line[0] = inp_char(stdscr) | 040;
        if (line[0] == 'y') {
                waddstr(stdscr, "Yes");         /* Usr wants defect list */
                waddstr(stdscr, "\nEnter choice of defect list from menu below:
                /* Display defect list menu options */
                for(i=0; i < 3; i++) {
                        wmove(stdscr, 8 + i, 0);
                        wprintw(stdscr, "%-45.45s  \n", defect_menu[i]);
                }

                /* Move curser to top of menu and wait for entry by user */
                while(TRUE)
                {
                    wmove(stdscr, 6, 46);
                    wclrtoeol(stdscr);
                    input(stdscr, line);            /* Get user input */
                    sscanf(line, "%x", &deflst_sel);    /* chng from ascii to #
                    /* Check for valid menu selection */
                    if(deflst_sel > 0 && deflst_sel < 4)
                    {
                        /* Set read defect list cmd blk - byte 2 */
                        list_fmt = defect_byte2[deflst_sel];
                        wmove(stdscr, 7, 0);
                        wclrtoeol(stdscr);
                        break;
                    }

                    wmove(stdscr, 7, 0);
                    waddstr(stdscr, "Illegal entry, try again\n");
                    wrefresh(stdscr);
                }

                /* Read the defect list, trying all 3 defect list formats
                 * until one works.  Some drives may only support one format.
                 */
                for(j=0; j < 3; j++)
                {
                        def_lst.list[0] = (list_fmt | defect_lst_fmt[i]);
                        if (ioctl(disk, RFIOCRDDEF, &def_lst) >= 0)
                        {
                                good_deflist = 1;
                                break;
                        }

                }

                if(!good_deflist)
                {
                        waddstr(stdscr, "Read defect list command failed\n");
                        wprintw(stdscr, "RFIOCRDDEF: %s", sys_errlist[errno]);
```



```
                        wrefresh(stdscr);
                        return(-1);
                }

        } else
        {
                waddstr(stdscr, "No");
        }
}
/*****************************************************************************
 *                                                                           *
 *      get_number                                                           *
 *                                                                           *
 *      Function:                                                            *
 *              Function to read a number from the user                      *
 *                                                                           *
 *      Inputs:                                                              *
 *              num             - address of number for storing              *
 *              special         - Special non-numeric input character        *
 *              value           - value to set "num" to if "special"         *
 *                                is seen                                    *
 *              error_line      - line number of a line to use for           *
 *                                displaying errors                          *
 *                                                                           *
 *      Outputs:                                                             *
 *              -1      - error                                              *
 *              0       - number was read                                    *
 *              1       - "special" was seen                                 *
 *              2       - empty line                                         *
 *              3       - user specified quit ('q')                          *
 *                                                                           *
 *****************************************************************************/

int
get_number(num, special, value, error_line)
        daddr_t *num;   /* place to store number */
        char special;   /* non-numeric input character */
        daddr_t value;  /* value to set to *num if 'special' is typed */
        int error_line; /* Error line number */
{
        register char *ptr;
        char *buff = line;
        char u_special;


        /* Get a line of input */

        input(stdscr, buff);

        /*
         *      If empty line and the "special" character is not new-line,
         *      return an error. If empty line and "special" is new-line,
         *      return "special seen".
         */

        if ( buff[0] == '\0' ) {
```



```
                if ( special == '\n' ) {
                        *num = value;
                        return(1);
                }
                return(2);
        }

        /*
         *      We only use upper-case, convert "special", and check to see
         *      if "special" was entered
         */

        u_special = buff[0];

        if ( islower((int)u_special) )
                u_special = (char)toupper((int)u_special);

        if ( special == u_special ) {
                *num = value;
                return(1);
        }
        else if ( u_special == 'Q' )
                return(3);
        else {
                /*
                 *      Maybe a "real" number, check each character to be sure
                 *      that only digits were entered
                 */

                for ( ptr = buff; *ptr; ptr++ ) {
                        if ( !(isdigit((int)*ptr)) ) {
                                move(error_line, 0);
                                clrtoeol();
                                move(error_line, 0);
                                printw("'%c' is not numeric", *ptr);
                                refresh();
                                *num = 0;
                                return(-1);
                        }
                }
                /* Convert the string to an integer */

                *num = atoi(buff);
        }
        /* User entered a number */

        return(0);
}
/* center_line:
 Center a message on stdscrn at the given line.  Assumes no new-lines,
 returns, tabs, or other unusual characters in the message */

center_line(line,message)
int line;
char message[];
{
```



```
    move(line,0);
    clrtoeol();
    move(line,(COLS-strlen(message))/2);
    addstr(message);
} /* center_line */

/* Overlap:
 * Check to see if the partition passed is overlapping with other
 * partitions.  Return the overlapping partitions.
 */
overlap(partition)
        int partition;          /* Partition to check */
{
        register struct dk_map *map;
        register int i, j;
        register check_start, check_end;        /* starting and ending blocks
                                                 * for passed in partition */
        register int start = 0;
        register int multiplier = label.dkl_nhead * label.dkl_nsect;
        register int overlap = 0;


        /*
         *      Calculate the partition to check's parameters
         */

        map = &label.dkl_map[partition];
        check_start = map->dkl_cylno * multiplier;
        check_end = check_start + map->dkl_nblk;

        /* If partition is 0 megabytes, don't check overlaps */
        if((check_start == 0) && (check_end == 0))
        {
                return(overlap);
        }

        /*
         *      Look through the partition map and find all overlapping
         *      partitions
         */

        for ( i = 0, j=0, map = label.dkl_map; i < NDKMAP; i++, map++ ) {
                /* Skip ourselves */
                if ( i == partition )
                        continue;

                /* Does it overlap ?? */
                start = map->dkl_cylno * multiplier;
                if ( (start > check_start) && (start < check_end) )
                        overlap |= (1 << i);    /* Yes */
                else if ( ((start + map->dkl_nblk) > check_start)
                            && ((start + map->dkl_nblk) < check_end) )
                        overlap |= (1 << i);
                else if ( (start <= check_start) &&
                                (( start + map->dkl_nblk) >= check_end) )
                        overlap |= (1 << i);
```

```
        }

        return(overlap);
}
```



```
/************************************************************************
 *                                                                      *
 *      new_hog                                                         *
 *                                                                      *
 *      Function:                                                       *
 *              This function is used to calculate the "hog" partition  *
 *      values. The "hog" partition is defined as the partition used to *
 *      hold all blocks from the last assigned block to the end of      *
 *      volume.                                                         *
 *                                                                      *
 *      Inputs:                                                         *
 *              none                                                    *
 *                                                                      *
 *      Outputs:                                                        *
 *              void                                                    *
 *                                                                      *
 ************************************************************************/

#ifdef  HOG_PARTITION

static void
new_hog()
{
        register struct dk_map *cpart = label.dkl_map;
        register struct dk_map *hog = &label.dkl_map[Hog];
        register int last_cyl = 0;
        register int i, zero;
        register long end_cyl;

        if ( Hog == NO_HOG )
                return;


        /* If the partition table is all zero's, don't calculate
         * the free hog.  Else you will kill cs35ut with a core
         * file and an Arithmetic exeption.
         */
        for ( i = MINPART, zero = TRUE; i < MAXPART; i++)
        {
                if (cpart->dkl_nblk > 0)
                {
                        zero = FALSE;
                        break;
                }
        }
        if (zero)
                return;

        /*
         *      Find the last assigned cylinder
         */

        for ( i = MINPART; i < MAXPART; i++, cpart++ ) {

                /*
```



```
                 *      Skip the hog partition and any that have the entire disk
                 *      allocated.
                 */

                if ( (i == Hog) || (cpart->dkl_nblk == Blocks) )
                        continue;

                /*
                 *      Calculate the last cylinder used by this partition. If
                 *      it is the largest so far, remember it
                 */

                end_cyl = (long)(cpart->dkl_cylno +
                                        (cpart->dkl_nblk / Cblocks) - 1);
                if ( end_cyl > last_cyl )
                        last_cyl = end_cyl;
        }

        /*
         *      The "hog" starts at the last assigned (cylinder + 1)
         *      and is (Blocks - start) in length
         */

        last_cyl++;
        if ( last_cyl >= label.dkl_ncyl ) {
                /* Must already have a hog partition */
                hog->dkl_cylno = (daddr_t)0;
                hog->dkl_nblk = (daddr_t)0;
        }
        else {
                hog->dkl_cylno = (daddr_t)last_cyl;
                hog->dkl_nblk = (daddr_t)(Blocks - (hog->dkl_cylno * Cblocks));
        }
}
#endif  /* HOG_PARTITION */
```