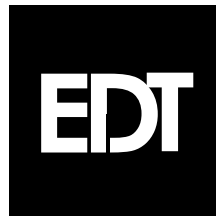


# **SDV**

## **Remote Camera Interface**

### **USER'S GUIDE**

008-00915-00



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

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

© Copyright Engineering Design Team, Inc. 1996. All rights reserved.

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

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

Phone (503) 690-1234

FAX (503) 690-1243

WWW: <http://www.edt.com>

email: [info@edt.com](mailto:info@edt.com)

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

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

Xilinx is a registered trademark of Xilinx, Inc.

Kodak is a trademark of Eastman Kodak Company.

The software described in this manual is based in part on the work of the independent JPEG Group.

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

# Contents

|  |    |
|--|----|
| System Requirements.....                                     | 2  |
| Platform and Operating System.....                           | 2  |
| Memory .....   | 2  |
| Display Hardware .....                                       | 2  |
| Installation.....  | 3  |
| Installing the Hardware .....                                | 3  |
| Installing the Software.....                                 | 4  |
| Installing for SunOS Version 4.1.x (Solaris 1.x).....        | 4  |
| Installing Solaris 2.x (SunOS 5.x) .....                     | 5  |
| Changing the Camera Model.....                               | 6  |
| Customizing the Camera Configuration .....                   | 6  |
| Building the Sample Programs.....                            | 7  |
| Included Files.....  | 7  |
| The SDV Tool.....  | 10 |
| Before Running the SDV Tool.....                             | 10 |
| Running the SDV Tool.....                                    | 10 |
| Using the SDV Tool .....                                     | 11 |
| The Control Panel .....                                      | 12 |
| SDV Programming Interfaces .....                             | 16 |
| X Imaging Library (XIL) Functionality .....                  | 16 |
| The Digital Video Library .....                              | 16 |
| sdv_open(device_name, lock).....                             | 17 |
| sdv_close(sd).....   | 17 |
| sdv_multibuf(sd, nbufs) .....                                | 18 |
| sdv_image(sd).....   | 18 |
| sdv_start_image(sd).....                                     | 19 |
| sdv_wait_image(sd).....                                      | 19 |
| sdv_abort_current(sd) .....                                  | 20 |
| sdv_grab(sd).....  | 20 |
| sdv_grab_average(sd) .....                                   | 21 |
| sdv_camera_type(sd) .....                                    | 21 |
| sdv_set_exposure(sd, value) .....                            | 22 |
| sdv_get_exposure(sd) .....                                   | 23 |
| sdv_set_gain(sd, value).....                                 | 23 |
| sdv_get_gain(sd, gain) .....                                 | 24 |
| sdv_set_blacklvl(sd, value).....                             | 25 |
| sdv_get_blacklvl(sd, blacklvl) .....                         | 25 |
| sdv_enable_lock(sd, flag) .....                              | 26 |
| sdv_setmem(sd, numbufs, bufs) .....                          | 27 |
| sdv_set_remap(sd, remap) .....                               | 28 |
| sdv_overrun(sd) .....  | 29 |
| sdv_picture_timeout(sd, tenths).....                         | 29 |
| sdv_debug(flag) .....  | 30 |
| sdv_write_rasfile(fname, addr, x_size, y_size).....          | 30 |
| sdv_write_rasfile24(fname, addr, x_size, y_size).....        | 31 |
| sdv_write_image(fname, addr, x_size, y_size, istride).....   | 32 |
| sdv_write_image24(fname, addr, x_size, y_size, istride)..... | 33 |

|   |    |
|---|----|
| sdv_serial_read(sd, buf, count) .....   | 33 |
| sdv_serial_write(sd, buf, count) .....  | 34 |
| sdv_serial_command(sd, cmd) .....   | 34 |
| sdv_start_hardware_continuous(sd, frames) .....   | 35 |
| sdv_stop_hardware_continuous(sd) .....  | 35 |
| foi_parity_error .....  | 36 |
| Camera-specific Issues .....  | 37 |
| Kodak MEGAPLUS 1.6 .....  | 37 |
| Kodak MEGAPLUS i Model Cameras (1.4i, 1.6, 1.6i, 1.68i, 4.2i) and ES 1.0 .....  | 37 |
| Troubleshooting .....   | 39 |
| If you encounter problems installing the software: .....  | 39 |
| If you have problems acquiring images using SDV Tool (dv), xwin, or other application programs: .....   | 39 |
| If you encounter problems compiling, linking or running your application program, or if you get<br>"Illegal ioctl" messages on the console when you run your program: ..... | 40 |
| And if all else fails... .....  | 40 |
| Connector Pinout .....  | 42 |
| Registers .....   | 44 |
| FOI_Read_Return Register .....  | 44 |
| FOI_Read_Trigger Register .....   | 44 |
| FOI_Board_ID Register .....   | 44 |
| FOI_Board_Select Register .....   | 44 |
| FOI_Xilinx_Prog Register .....  | 45 |
| Powering Up and Resetting the SDV-RCI .....   | 45 |
| Direction_Control Register .....  | 46 |
| Specifications .....  | 47 |
| Device Data Transfer .....  | 47 |
| Software .....  | 47 |
| Power .....   | 47 |
| Environmental .....   | 47 |
| Physical .....  | 47 |
| Contacting EDT .....  | 48 |
| Appendix A Input and Output .....   | 49 |
| ioctl() Parameters .....  | 49 |
| Asynchronous Input and Output .....   | 51 |
| semtest .....   | 52 |

# Figures

SDV Tool .....12  
SDV Tool Camera Controls pop-up window.....14  
Histogram pop-up window.....15

# Tables

Connector Pinout.....43  
FOI\_Board\_ID Register .....44  
FOI\_Board\_Select Register .....44  
The FOI\_Read\_Trigger Register .....44  
FOI\_Xilinx\_Prog Register .....45  
The Direction\_Control Register.....46

## Overview

The Digital Video Remote Camera Interface (sdvrci) is an external module that implements a high-speed interface between an external digital camera and the SBus Configurable DMA Fiber Optic Interface (SCD-FOI) board, by means of a fiber optic cable. The SDV-RCI can contain up to four daughter boards, each of which can communicate with its own camera. The camera interface consists of thirty-two RS-422-compatible driver/receivers per daughter board connected to a Xilinx RAM-based programmable gate array. These driver/receivers can be assigned as inputs or outputs in groups of four. The Xilinx device can be programmed to implement arbitrary interface protocols by executing a program that downloads a bit pattern from a file to the SDV-RCI board.

This document describes the hardware and software requirements, how to install the Digital Video Remote Camera Interface and driver, and how to use the graphical interface application. It also describes the driver library routines, the structure of the hardware registers, and contains a pinout diagram. It is divided into the following sections:

|                                    |  |
|------------------------------------|--|
| <b>System Requirements</b>         | Describes the hardware and software requirements for the EDT Digital Video Remote Camera Interface   |
| <b>Installation</b>                | Describes how to install the SDV-RCI module and its related software.  |
| <b>The SDV Tool</b>                | Explains how to use the OpenWindows-based interface application for the SDV-RCI board.   |
| <b>SDV Programming Interfaces</b>  | Outlines the basic elements of SDV-RCI applications, and lists and explains the routines available to access the SDV-RCI module with your program. |
| <b>Camera-specific Issues</b>      | Describes issues related to specific camera models.  |
| <b>Troubleshooting</b>             | Provides troubleshooting tips  |
| <b>Connector Pinout</b>            | Provides the pinout diagram for the SDV-RCI cable.   |
| <b>Registers</b>                   | Describes the Digital Video Remote Camera Interface hardware registers.  |
| <b>Specifications</b>              | Lists product specifications   |
| <b>Contacting EDT</b>              | Describes how to access EDT's World Wide Web and FTP sites to obtain technical help, software updates, documentation, and product information.     |
| <b>Appendix A Input and Output</b> | Describes the parameters to the <i>ioctl</i> s.  |

# System Requirements

The Digital Video Remote Camera Interface software is designed to be used in Sun SPARC and UltraSPARC workstations or compatible computers, running SunOS versions 4.1.x (Solaris 1.x) or SunOS 5.x (Solaris 2.x). The SDV-RCI captures data from a variety of cameras, but due to the high speeds and large amounts of data involved, your system needs to meet certain minimum speed and memory requirements.

## Platform and Operating System

Most combinations of camera and platform run without a problem. However, some platforms have proven to be marginal if used with the faster or larger-format cameras. Contact EDT if you have one of the following combinations:

- SPARCstation 4 or 5 under SunOS 4.1.3 or Solaris 2.3 or earlier, and a camera that produces an image larger than 5 MB.
- SPARCstation 1, 1000, Classic, or LX (any operating system), and a camera that runs faster than 10 MB per second.

Documentation for most cameras specifies throughput in terms of MHz, or pixels per second. This is equivalent to MB per second for 8-bit cameras. For 10-, 12-, 14- and 16-bit cameras, multiply the MHz (pixels per second) by two to get data throughput in bytes.

## Memory

The minimum amount of memory required to run applications with the `sdvrci` depends on the camera and the application being used to capture or display the images. As a general rule, in order to capture and display single images using the applications provided, you need at least three times the amount of memory necessary to hold one image of the size generated by your camera, plus the minimum memory required by your operating system and window manager, as well as any other programs you run concurrently.

For example, the Kodak MEGAPLUS 4.2i 10-bit camera produces a 2029 by 2047-pixel image, and uses two bytes per pixel. This equals 8,306,276 bytes (8.3 MB). If you use a SPARCstation 20 running SunOS 5.5 with OpenWindows, which requires a minimum of 32 MB, you need at least  $32 + (8.3 \times 3)$ , or approximately 57 MB of memory. You can enhance the performance of the `sdvrci` interface and display software by using more than the minimum required memory, and by increasing the number of multiple buffers used by the `sdvrci` application. Running with less than the necessary memory results in degraded performance, as the system will be forced to swap memory to disk; in extreme cases this could cause the application or operating system to fail.

## Display Hardware

In order to display images with the `sdvrci` (using the `sdvrci tool` or `xwin` program, for example), you need at least an 8-bit display processor running in either StaticGray, GrayScale or PseudoColor mode. A 24-bit frame buffer running in TrueColor mode (such as the SPARCstation 20 SX) provides optimal display resolution, although it generally requires more memory and results in somewhat slower display times.

# Installation

To install the Digital Video Remote Camera Interface, first connect the box to a power source, a camera, and the host computer. Then install the software driver in the host computer so that applications can access the SDV-RCI.

## Installing the Hardware

The Digital Video Remote Camera Interface is a box containing the remote camera interface board, from one to four daughter boards, a power cable, a camera cable, and either a fiber optic cable or copper coaxial cables. To install the Digital Video Remote Camera Interface:

1. Unpack the Digital Video Remote Camera Interface from the shipping packaging and examine the box. One side has one 80-pin high-density connector per daughter board. The opposite side has a power connector on the left (J1). On the right are connectors for either a fiber optic cable using a standard SC Duplex connector passing 1300 nm light (J4), or a 50- $\Omega$  coaxial cable with standard BNC connectors (J3 and J5).
2. Plug the wall transformer at the end of the power cable into the power outlet in the wall.
3. Plug the other end of the power cable into the power connector J1.
4. On the other side of the box, plug the 80-pin high-density cable into connector P3.
5. Connect the other end of the 80-pin high-density cable into the camera you intend to use with the board.

If you intend to use the Digital Video Remote Camera Interface with the fiber optic cable, proceed with installation as follows:

6. Remove the black plastic dust protectors from both sides of both ends of the fiber optic cable (four in all) and store them in a safe place.
7. Remove the black plastic dust protector from the fiber optic interface J4.
8. Insert both sides of either end of the fiber optic cable into the two holes of the fiber optic interface. To firmly seat the fiber optic cable, ensure that the two tabs at the top of the cable connectors are aligned with the two slots at the top of the interface connector.
9. In the same manner, connect the other end of the fiber optic cable to the SCD-FOI connector in the host computer.

The Digital Video Remote Camera Interface is now ready to use with the fiber optic cable.

If you wish, you can use a 50- $\Omega$  coaxial cable with standard BNC connectors instead. To do so:

1. Perform steps 1–5 as described above.
2. Unscrew the screws holding the top of the Digital Video Remote Camera Interface box and set them aside in a safe place. The top is held on with five screws—two on either side, and one in the center of the back between J1 and J3, 4, and 5.
3. Lift the top of the box off carefully and set it aside.
4. Ground yourself with an antistatic strap.
5. The board is shipped with the jumpers near the transceiver strapped into the position for an optical cable—pin 1 connected to pin 2. These are the pins closest to the edge of the board. To use a coaxial cable, lift the three black plastic cases and place them over pins 2 and 3, one position further from the edge of the board.

6. Replace the top of the box and screw the screws back in.
7. Using one coaxial cable, connect the J3 BNC connector (serial out) to the host computer's serial input port.
8. Using another coaxial cable, connect the J5 BNC connector (serial in) to the host computer's serial output port.

To disconnect the Digital Video Remote Camera Interface, reverse the installation procedure.

## Installing the Software

The software is provided with the board on a floppy disk, and updates are available over the internet. The Digital Video Remote Camera Interface can run on a Sun workstation using either SunOS Version 4.1.x (Solaris 1.x) or SunOS 5.x (Solaris 2.x). The software and installation procedures differ. Both are given below.

**NOTE: If you have had the board for a period of time before installing it, we recommend you get the most current software to ensure you have all the latest enhancements. Call EDT for an updated software package, or follow the instructions in Contacting EDT on page 48 to download it over the Internet.**

### Installing for SunOS Version 4.1.x (Solaris 1.x)

If you are using SunOS Version 4.1.x, be sure your floppy disk is labeled as such, and use the following procedure to install the Digital Video Remote Camera Interface driver:

1. Become root or superuser.
2. Change to the directory in which you wish to install the Digital Video Remote Camera Interface driver.
3. If you are installing from a diskette, place the diskette that came with the Digital Video Remote Camera Interface into the diskette drive. If you are installing from a file downloaded from the internet, copy the file to this directory.
4. The Digital Video Remote Camera Interface driver and related files are included on a diskette in compressed *tar* format. To uncompress and extract the files from the diskette, enter:

```
zcat < /dev/rfd0 | tar xvf -
```

If the software is in a file instead of on a diskette, enter:

```
zcat < EDTsdvtar.Z | tar xvf -
```

5. The above command extracts a number of files. (The list of files distributed is provided in **Included Files** on page 7.) The Digital Video Remote Camera Interface diskette contains versions of the Digital Video Remote Camera Interface driver for a variety of Sun platforms and versions of the Sun operating system. The installation program installs the correct driver based on the host platform and operating system version.
6. To install the driver, enter:

```
make install
```

The makefile provided installs and loads the Digital Video Remote Camera Interface driver.

7. When prompted, select the camera that you are using from the list. If your camera is not on the list, select the number that corresponds to "Pseudo Camera", then complete the installation and contact EDT for help with setting up your particular camera.

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

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

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

```
make load
```

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

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

You can install as up to eight Digital Video Remote Camera Interface boards in your system. Enter the number corresponding to the number of SDV boards you have installed in your system. If you simply type <Return>, one driver is installed.

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

10. If the Digital Video Remote Camera Interface has not been installed inside the host computer, or has been installed incorrectly, the following message appears on the display:

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

If you see this message, go back to **Installing the Hardware** on page 3 and reinstall the module.

To unload the Digital Video Remote Camera Interface driver:

1. Change to the directory in which you placed the Digital Video Remote Camera Interface files, if necessary.
2. Become root or superuser.
3. Enter:

```
make unload
```

## Installing Solaris 2.x (SunOS 5.x)

If you are using Solaris 2.X (SunOS 5.x), make sure your floppy disk is labeled as such, and use the following procedure to install the Digital Video Remote Camera Interface driver:

1. Become root or superuser.
2. Place the diskette that came with the Digital Video Remote Camera Interface into the diskette drive.
3. Tell the Solaris Volume Management to check for a new floppy in the drive, by running

```
volcheck
```

If you are running File Manager, it may bring up a "Format" pop-up at this point, alerting you to the presence of an unformatted floppy. Dismiss this window.

4. At the shell prompt, enter:

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

The `pkgadd` program will prompt you for whether you *really* want to do this once or twice. Don't be discouraged, go ahead and say yes.

5. When prompted, select the camera that you are using from the list. If your camera is not on the list, select the number that corresponds to "Pseudo Camera", complete the installation, and contact EDT for help with setting up your particular camera.

To remove the Digital Video Remote Camera Interface driver:

1. Become root or superuser.
2. Enter:

```
pkgrm EDTsdv
```

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

## Changing the Camera Model

During the installation procedure, you are asked to choose from a selection of camera models. This part of the installation involves running a script, called `sdvcamera`. If you wish to choose a different camera or to reinitialize the device driver after installation, you can run this script manually. To do so:

1. Change to the directory where you installed the SDV-RCI files.
2. Enter:

```
sdvcamera
```

See **Camera-specific Issues** on page 37 for issues concerning specific cameras.

## Customizing the Camera Configuration

In most situations the above procedure is adequate to configure the SDV-RCI board for your particular camera. However, if you wish to use a camera model for which no configuration file has been provided, if you wish to modify the parameters of an existing configuration file, or if you wish to use a system with more than one camera and SDV-RCI board, read the following explanation.

**NOTE: If you wish to change your configuration file, we recommend that you consult EDT.**

The `sdvcamera` script calls two other scripts, `sdvrequest` and `sdvselect`, which together create `sdvload`, yet another script that contains the command line to run the `initcam` program on the selected configuration file. `sdvload` runs automatically whenever you reboot the computer. You can also run it manually when you wish to reinitialize the device driver and `sdvrcl` hardware.

If you use a camera for which a configuration file is not provided by EDT, or wish to modify the parameters in a copy of an existing configuration file, you can do so by changing the name of the configuration file specified in the `sdvload` script. Configuration files reside in the `camera_config` subdirectory and are named with a `.cfg` extension.

By default, `sdvload` initializes the device driver for only one board. If you have more than one camera and SDV-RCI board in the system, add lines to the `sdvload` script to initialize the additional boards. Use the `-u N` command line option to `initcam` to specify which board to initialize with each configuration file. For

example, if you need to initialize two boards, both of which are connected to Kodak Megaplus 1.4 8-bit cameras, `sdvrciload` must contain the following lines:

```
initcam -u 0 -f camera_config/KodakMEGAPLUS14i-8.cfg
initcam -u 1 -f camera_config/KodakMEGAPLUS14i-8.cfg
```

## Building the Sample Programs

To build any of the example programs, enter the command:

```
make file
```

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

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

```
make
```

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

## Included Files

The following table lists the files included on the SDV-RCI software diskette. Some files may not be on your diskette. Specifically, if you specified Solaris 2.x (SunOS 5.x) when you ordered your board, the Solaris 1.x (SunOS 4.x) device drivers will not be included. The reverse is true if you ordered SunOS 4.x. Also, the binary executable files for most of the example programs are not included in the SunOS 4.1.x versions of the software. Contact EDT (or our Web Server) if you need driver disks for both operating systems.

For a complete and up-to-date list of the included files, see the *readme* file.

|                                |   |
|--------------------------------|---|
| camera.h                       | The header file for SDV-RCI camera controls.  |
| camera_config/<br>camera.rbt   | Camera-specific firmware; see the <i>sdvrci</i> addendum for a specific camera model.   |
| camera_config/<br>camera.cfg   | Camera-specific configuration files, selected by <i>sdvcamera</i> . Each file contains a description of the camera that gets passed in to the device driver. Includes height, width, depth, features available, driver specific flags and methods, and the pathname of the firmware ( <i>.rbt</i> ) file to download to the hardware. |
| camera_config/<br>hardware.rbt | Firmware file specific to the hardware. This file is named in the <i>camera_config/camera.cfg</i> file for the camera in use, and is downloaded when <i>initcam</i> is invoked (by <i>sdvrcicamera</i> and also on boot).   |
| scdfoi.rbt                     | Xilinx firmware for the SCD-FOI board.  |
| dv                             | General purpose, OpenWindows-based application for capturing, displaying, and saving images.  |
| drv_ioctl.h                    | Header file containing IOCTL definitions  |
| initcam<br>initcam.c           | Program (executable and source) for configuring the device driver and downloading the <i>sdvrci</i> firmware. The <i>sdvrcicamera</i> script runs <i>initcam</i> with the selected configuration file. On boot, the file <i>sdvrciload</i> is invoked   |
| INSTALL                        | SunOS 4.1.3 installation script used by the makefile. Do not invoke directly—use <i>make install</i> to install the software.   |
| libsdv.a                       | SDV-RCI C interface library. Link with any application program that makes calls to <i>sdvrci</i> library routines.  |

|                              |  |
|------------------------------|--|
| makefile                     | Makefile for installing, loading, and unloading the sdvrcl device driver, and making the example programs. Used with the <i>make</i> command to automatically install the driver or compile the example programs.  |
| makeras<br>makeras.c         | Program (executable and source) to convert a raw data file to a Sun raster format file.  |
| pseudo                       | Shell script that configures the sdvrcl board for Pseudo Camera operation at a given width, height and depth. Useful for diagnostics and testing without an actual camera device.  |
| README                       | ASCII file containing last-minute information about the sdvrcl software.   |
| sdv                          | Executable SDV-RCI device driver for Solaris Version 2.x.  |
| sdv.h                        | SDV-RCI device driver header file defining <i>ioctl</i> s.   |
| sdv.o.sun4c                  | Executable SDV-RCI device driver for SunOS 4.1.x on a Sun 4C architecture such as a SPARCStation 1, 1+, 2, or IPC.   |
| sdv.o.sun4m                  | Executable SDV-RCI device driver for SunOS 4.1.x on a Sun 4M architecture such as a SPARCStation 5, 10, 20, LX, Classic, or an Ultra 1 or 2.   |
| sdv_reg.h                    | SDV-RCI device driver header file defining registers.  |
| sdvcamera                    | Shell script for choosing the camera and downloading the firmware.   |
| sdvselect                    | Shell script invoked by sdvrcicamera. Not meant to be run directly.  |
| sdvload                      | Shell script created by <i>sdvrclselect</i> (by means of <i>sdvrcicamera</i> ). Contains the command line to run <i>initcam</i> to initialize the device driver and download the firmware to the sdvrcl device. Once the sdvrcl software has been installed, this script runs automatically every time the system is booted. It can also be run by hand to reinitialize the device driver and SDV-RCI hardware to a known state. |
| sdvrequest                   | Shell script invoked by sdvrcicamera. Not meant to be run directly.  |
| sdvlib.c                     | Source code for SDV-RCI C interface library ( <i>libsdvrcl.a</i> ).  |
| sdvlib.h                     | Header file for the SDV-RCI C interface library.   |
| serial_test<br>serial_test.c | Used to send control commands to the Kodak AIA Serial device. Only applicable to Kodak MEGAPLUS 'i' model cameras. See your <i>Kodak MEGAPLUS Camera User's Manual</i> for an explanation of the valid serial commands.  |
| setdebug<br>setdebug.c       | Simple program that sets the debug level for the SDV-RCI. Sets tells the SDV-RCI device driver to output debug information to the console. Enter <i>setdebug -H</i> to find the command line options. Because the debugging output can be rather cryptic, this program is generally useful only when working with EDT technical support to optimize an application or diagnose problems.   |
| speedtest<br>speedtest.c     | A simpler example program illustrating asynchronous I/O read requests to the sdvrcl using ring-buffering and the driver handshaking mechanisms. <i>speedtest.c</i> can use the internal loopback mode to generate data.  |
| take<br>take.c               | Example/diagnostic program that reads an image or images, and optionally writes to a raw or Sun Raster format file. Includes a number of options that can be useful in optimizing performance and diagnosing problems with the camera and interface. Does not have any display capabilities.   |
| watchstat<br>watchstat.c     | Diagnostic program that displays information on current sdvrcl bus activity.   |
| xwin<br>xwin.c               | An example program that captures and displays 8 bit images in a simple X window, using <i>sdvrclib</i> calls and XLib. A good starting place for your application.   |

|                 |   |
|-----------------|---|
| xil/display     | An example program that shows the use of the X Imaging Library interface for the sdrvci under Solaris. See the Sun AnswerBook documentation for the X Imaging Library.  |
| xil/display.c   |   |
| xil/xiliosdv.so | Dynamically linked X Imaging Library (XIL) pipeline library file for the SDV-RCI. The installation procedure copies this file to the SUNWits/Graphics-sw/xil/lib/pipelines directory, which enables you to use <i>xil_create_from_device()</i> on the SDV-RCI device. See the Sun AnswerBook documentation for detailed documentation on the X Imaging Library. |

## The SDV Tool

The SDV tool, or *dv*, is a general purpose application that incorporates calls to the SDV-RCI device driver in an X Windows application program. The program provides a graphical user interface for capturing and displaying video images and storing them to disk for further image manipulation. The SDV tool also provides access to selected controls on applicable camera devices, including shutter speed, shutter lock, shutter trigger, gain, black level, continuous shutter repeat and image update, panning and zooming.

### Before Running the SDV Tool

To run the SV tool, you must be running an X Window manager such as OpenLook. If you are running under the Solaris 2.x (SunOS 5.x) operating system, you need to make sure that the dynamically linked X imaging library (XIL) path and Openwin home paths have been added to your `$LD_LIBRARY_PATH` environment variable in order to run the `sdvrci` Tool or other XIL based application. Assuming that the XIL files are in their default location, one way to do this is to add the following lines in your `.login` file *after* any existing `LD_LIBRARY_PATH`:

```
setenv XILHOME /opt/SUNWits/Graphics-sw/xil
setenv OPENWINHOME /usr/openwin
setenv LD_LIBRARY_PATH "$XILHOME/lib:$OPENWINHOME/lib:$LD_LIBRARY_PATH"
```

Then source your `.login` file again:

```
source .login
```

If you are using a camera that has a front panel mode switch with a **Computer** setting for example, some Kodak MEGAPLUS cameras), set the knob to this setting. If this knob is set otherwise, the SDV tool displays the video picture, but shutter and exposure settings are as set on the camera control unit instead of *dv*, and you will probably not be able to trigger the shutter using the application program.

### Running the SDV Tool

Invoke the SDV tool by entering:

```
dv
```

If you are running on a StaticGray or 24-bit TrueColor display, *dv* displays images from monochrome cameras in 256 shades of gray. If you are using an 8-bit Grayscale or PseudoColor display, *dv* allocates the first free 128 color map entries in the display and rescales images for display in 128 shades of gray. Under most conditions, this results in good display definition with no color map flashing. However, some situations have specific requirements; therefore, several options available on the *dv* command line allow you to customize the color map configuration. If, for example, you want *dv* to use the maximum number of color map entries beyond those already in use by the window manager or other applications, enter:

```
dv -A
```

For best performance, run the window manager in StaticGray mode.

The complete list of options available with *dv* is as follows:

- a** Use all colormap entries (normally uses first available 128).<sup>a</sup>
- A** Use all colormap entries in default colormap that are not already used by the window manager or other applications.<sup>a</sup>

- e N** Specify the number of color map entries to use, starting at the first unused one, or at the one specified by the **-p** flag if present.<sup>a</sup>
- c** Do not allocate any color map cells. colors will still be remapped per the default or as specified by the **-p** and/or **-e** flags. Use this flag if you have grayscale colormap cells already allocated in the default colormap from another program that you want dv to use.<sup>a</sup>
- d** Output debug information.
- f** Fake open—don't access the sdvrci card, create grayscale image internally.
- h** Use high 8 hardware color map filter (8-bit devices only).<sup>b</sup>
- i** Start up with h/w image polarity set to inverted (negative).
- l** Use low 8 hardware color map filter (8-bit devices only).<sup>b</sup>
- n N** Specify number of multiple buffers to use in ring-buffering, for improved performance when more memory is available (default 1).
- p N** Specify the number of entries to preserve at the bottom of the color map for use by the window manager or other applications. default is to start at first entry that is not already in use.<sup>a</sup>
- r** Use rainbow colormap (useful with infrared cameras).
- t** Disable acquire timeout (default is to block on acquire).
- u N** Use unit /dev/sdvrciN (default is /dev/sdvrci0).
- U** Display option list. does not invoke the program.

a. Only apply with 8-bit `PseudoColor` or `GrayScale` displays

b. Only useful with `StaticGray` displays—likely to cause flashing otherwise

## Using the SDV Tool

The *dv* main window consists of a control panel, a focus window, and a full image window. The control panel is at the upper left. The focus window is the small rectangle at the upper right. The full image window is the large rectangle below the control panel and focus window. The tool appears as shown in Figure 1.

### The Image Windows

The *full image* (larger) window displays either the full image or that portion of the image that can fit in the window, depending on the scaling and window size. When you first click on the **Expose** button, *sdvrci* selects a scaling value that allows the image to be fit into the window's default size. You can drag the corners of the application window with the mouse to enlarge or reduce the displayable image size. You can pan around the image by holding the middle mouse button down and moving the mouse, in effect dragging the image inside the window to bring the desired portion into view. It is often undesirable or impossible to display the entire image in full resolution in the full image window. For this reason, the *focus* window is provided to display a small portion of the image in full or double resolution. This is useful when focusing the camera, or for enhanced viewing of a specific portion of the image. To select a portion of the image to display in the focus window, move the mouse cursor to the desired spot in the full image window and click the left mouse button. Alternately, hold the left mouse button down within the full image window and move the cursor around to pan the image within the focus window.

**NOTE:** If you are using a camera that has a CCD size smaller than 198 x 128, the focus window and full image windows are reversed. The entire image is displayed in the upper right hand corner in 100% resolution, and an enlarged image (200% or more) is displayed in the main window. In this mode the pan and cursor focus operations are disabled because they don't apply.

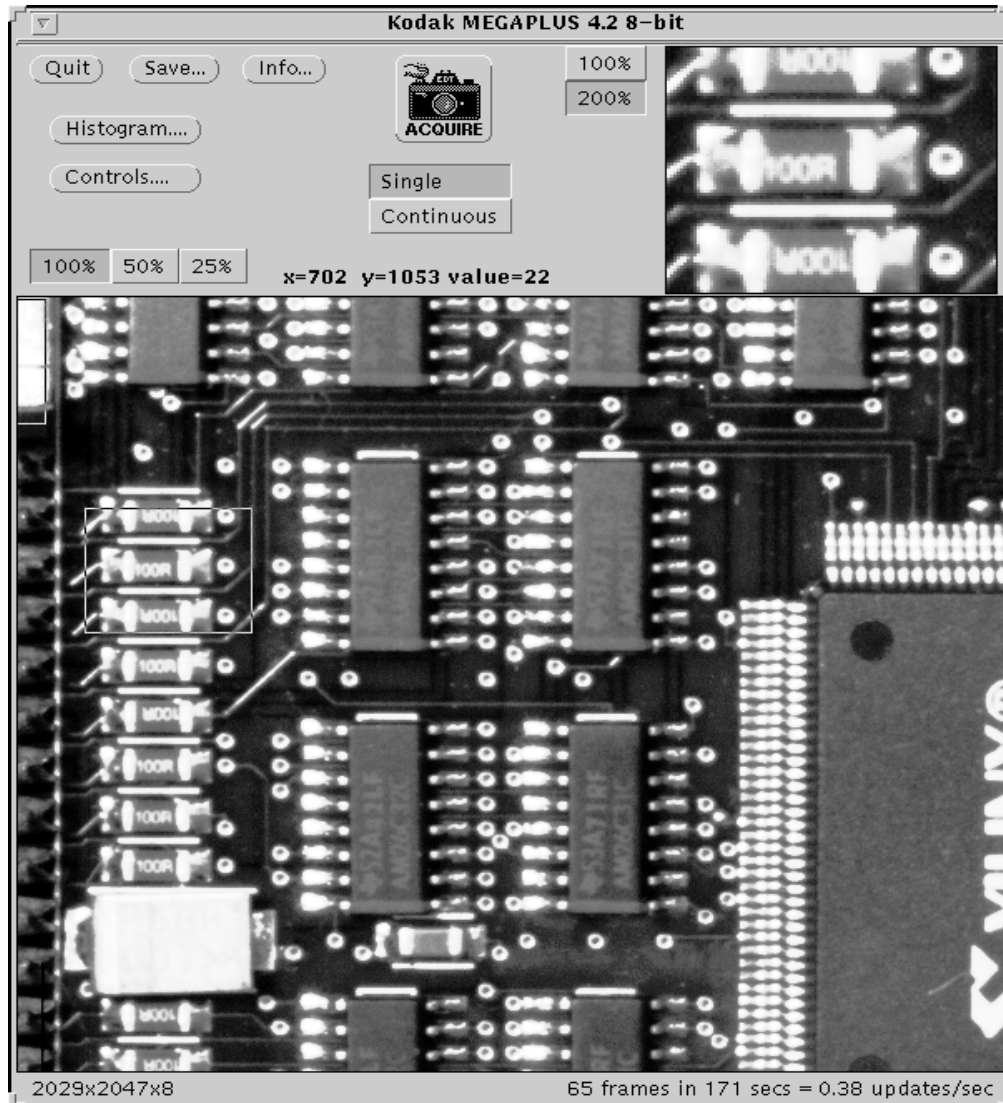


Figure 1. SDV Tool

## The Control Panel

From the top left to the bottom right, the sdvrci tool buttons work as follows:

**Save...** Invokes a popup window that prompts you for a file name and a file format setting. Four file formats are provided: raw, Sun raster, TIFF, and JPEG.

If **raw** format is selected, the file is saved as a stream of bytes, each byte representing a value (0–255) for the pixel (8 bit cameras), or 0-65535 (10-, 12-, 14- and 16-bit cameras). If you are using a camera that has more than 8 bits per pixel, saving in other than **raw** format will result in loss of all but the top 8 bits of data. No coordinate information is saved in this format, so any program that manipulates such a file must get the dimensions for the image some other way.

**Sun Raster** format is the standard Sun raster file format, as documented in Sun man page *rasterfile (5)*, or the Solaris OpenWindows *Reference Manual*, or the Solaris *User's Guide* in the section on Image Tool.

**TIFF** (Tag Image File Format) is a format that typically describes images that come from scanners, frame-grabbers, and photo-retouching programs. The TIFF specification is available from Aldus Corporation. TIFF images saved by the **sdvrci** Tool do not use any of the compression options available with some TIFF versions.

**JPEG** is an internationally recognized compressed image format, as established by the Joint Photographic Experts Group, for continuous-tone still images in both grayscale and color. The **sdvrci** uses the Baseline “lossy” method. For further information, see “The JPEG Still Picture Compression Standard” by Gregory K. Wallace, *Communications of the ACM*, April, 1991. Also see the *Solaris XIL 1.0 Imaging Library Programmer's Guide*, Chapter 14, “JPEG Baseline Sequential Compressor.” The SunOS 4.1 version is based in part on the work of the independent JPEG group.

Clicking on the **Save** button in this window saves the entire image as received from the camera, in the selected format, to the specified file name. You can specify an entire file path if required. Clicking on **Cancel** cancels the save operation.

|                     |  |
|---------------------|--|
| <b>Info...</b>      | Brings up a pop-up window with information on the device driver and camera as configured   |
| <b>Acquire</b>      | Grabs the current image and displays it on the screen.   |
| <b>Mode</b>         | Determines whether the display updates continuously ( <b>Continuous</b> ) or only when the <b>Acquire</b> button is pressed ( <b>Single</b> ). If the camera control unit has a <b>Mode</b> knob (for example, some Kodak MEGAPLUS models), it should be set to <b>Computer</b> for this to work properly.   |
| <b>Scaling</b>      | Determines how the image is scaled in the full image window. Since the image size (in pixels) of most camera devices is larger than the default full image window, the image by default is normally scaled to less than 100% in order to display the full image within the image window. You can enlarge the image window by dragging a corner of the application window. You can display the image at different resolutions by selecting a different value for the <b>Scaling</b> setting. You can pan the image around within the window by holding down the middle mouse button and dragging. |
| <b>Focus</b>        | On Solaris 2.x only, determines how the focus window's subimage is displayed. If 100% is selected, the subimage is displayed in full resolution. If 200% is selected, the tool uses bi-linear interpolation to display the image in twice the normal resolution.   |
| <b>Controls....</b> | Brings up a pop-up window with camera and display controls (see Figure 2).   |

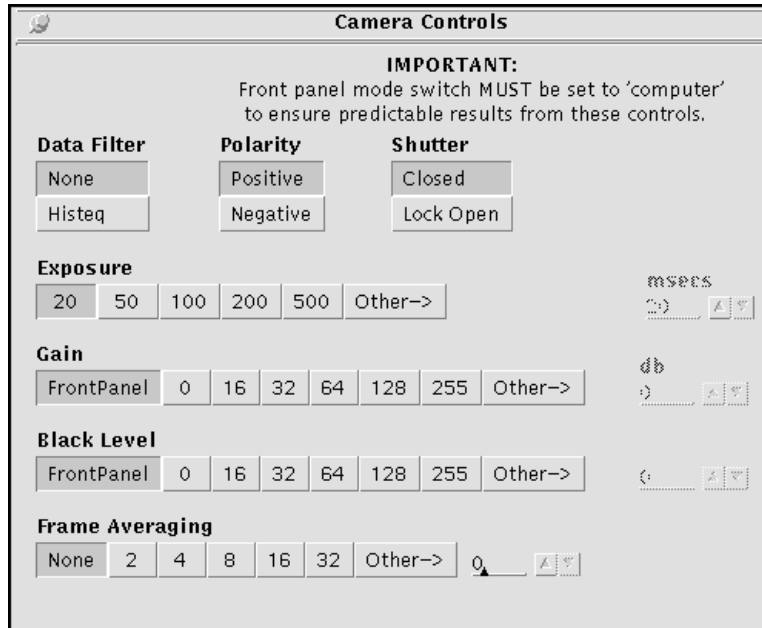


Figure 2. SDV Tool Camera Controls pop-up window

|                                 |  |
|---------------------------------|--|
| <b>Display Filter</b>           | <b>None</b> —no display filtering. <b>Histeq</b> perform a histogram equalization algorithm to rescale the pixels for display so that they cover the range of possible display values (0–256).   |
| <b>Polarity</b>                 | Sets the hardware polarity bit to select whether the image data is normal ( <b>Positive</b> ) or inverted ( <b>Negative</b> ).   |
| <b>Shutter</b> <sup>1</sup>     | Selects whether the shutter is normally <b>Closed</b> or locked <b>Open</b>  |
| <b>Exposure</b> <sup>1</sup>    | Determines the shutter speed in milliseconds. If <b>Front Panel</b> is selected, or if the <b>Mode</b> knob on the camera control unit is set to <b>Computer</b> , then the shutter speed is the speed selected on the front of the CCU. |
| <b>Gain</b> <sup>1</sup>        | Selects the gain on cameras that have this feature.  |
| <b>Black Level</b> <sup>1</sup> | Selects the black level on cameras that have this feature.   |
| <b>Frame Averaging</b>          | Selects the number of frames to be taken on each acquire and <b>Averaged</b> , and the results displayed. Selecting <b>None</b> results in only one acquire and no averaging.  |
| <b>Histogram...</b>             | Brings up a pop-up window with a histogram showing the distribution of the pixel values in the current image.  |

<sup>1</sup>Applies only to camera devices that have these software selectable controls

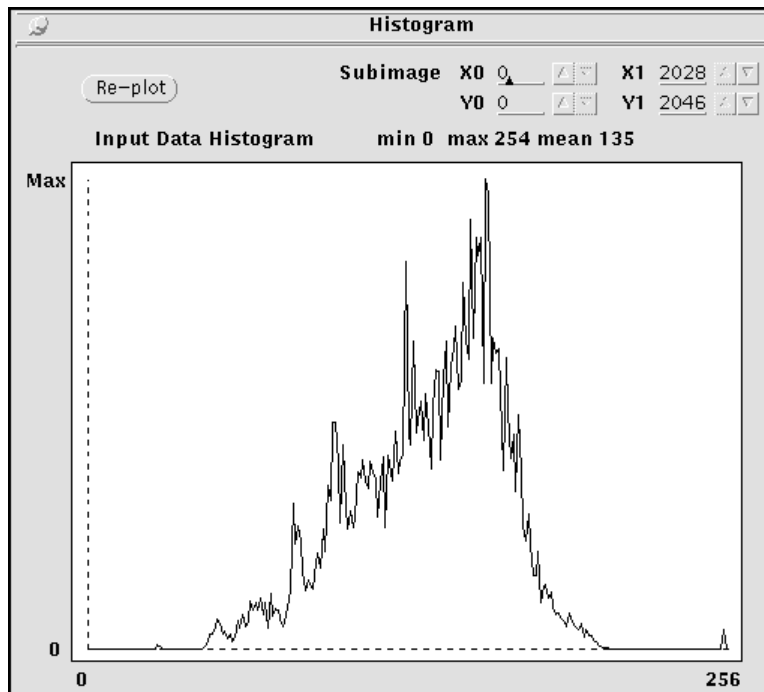


Figure 3. Histogram pop-up window

## SDV Programming Interfaces

Two paths exist for programmers to interface with the SDV-RCI: the SDV Digital Video Library, provided with the SDV-RCI software, and pipelined X Imaging Library (XIL) calls, provided by Solaris 2.x. Even if you choose not to use the pipelined XIL functionality to open the device and acquire images, you can still use XIL routines to manipulate and display images acquired by the regular SDV Digital Video Library routines.

### X Imaging Library (XIL) Functionality

The `sdvrci` under Solaris 2.x includes an X Imaging Library interface. To use it for grabbing and displaying an image, use `xil_create_from_device()` and `xil_copy()`, passing the string `iosdvrci` as the device argument. An example of this is available in the file `sdvrci_display.c` in the `/opt/EDTsdvrci/xil` subdirectory. `xil_create_from_device()` and `xil_copy()` are standard routines supplied by Solaris. See the Sun AnswerBook documentation for more information about these routines.

### The Digital Video Library

The SDV digital video library provides a C language interface to the SDV-RCI device driver. Routines for image capture, save, and device control are included.

All routines access a specific device, whose handle is created and returned by the `sdv_open()` routine. `sdvrci` applications typically include the following elements:

1. The preprocessor statement

```
#include "sdvlib.h"
```

2. A call to `sdv_open()`, such as:

```
struct sdvDev *sd = sdv_open("/dev/sdv0", sdv_LOCKDEV);
```

3. Other library routine calls, such as `sdv_image()` (which acquires an image and returns a pointer to it), as in:

```
caddr_t image = sdv_image(sd) ;
```

4. A call to `sdv_close()` to close the device before ending the program, as in:

```
sdv_close(sd) ;
```

5. The `-lsdv` option to the compiler, to link the library file `libsdv.a` with your program

See the makefile and example programs provided for examples of compiling code using the digital video library routines.

The `sdvDev` device status struct is defined in the file `sdvlib.h`. It includes elements that describe the camera as passed in from the `camera.cfg` file being used. It is recommended that you use the information in this structure to determine things such as width, height, depth, shutter speed min and max, instead of hard-coding these values into your program; this ensures that your program will not have problems if the camera device in use or its corresponding `camera.cfg` file is changed in the future.

**sdv\_open(device\_name, lock)****Description**

Opens the device, the first step in accessing the hardware. Allocates the memory for a device status struct, as defined in *sdvlib.h*, and host memory required to store a captured image.

If multiple SDV devices are installed, they are named *sdv0*, *sdv1*, *sdv2*, etc.

**Arguments**

*dev\_name*      The pathname of the device to be opened, which should exist. If NULL, attempts to open the default device `"/dev/sdv0"`.

*lock*            Defined values are SDV\_LOCKDEV or SDV\_NOLOCKDEV. SDV\_LOCKDEV locks the device with a nonblocking exclusive lock (see *flock (2)* in the SunOS documentation). SDV\_NOLOCKDEV does not check for a lock condition; therefore, the device can be opened even if it has previously been accessed.

**Returns**

A pointer to the *SdvDev* data structure, if successful. This structure holds the addresses of the host memory where captured images are stored, and a file descriptor for the open device.

NULL if unsuccessful.

**Example**

```
struct SdvDev *dv = sdv_open("/dev/sdv0", SDV_LOCKDEV);
```

**Syntax**

```
#include "sdvlib.h"
SdvDev *
sdv_open(dev_name, lock)
char *dev_name;
int lock;
```

**sdv\_close(sd)****Description**

Closes the specified device, unlocks it if it has been locked, and frees the device struct and image memory.

**Arguments**

*sd*              The pointer to the device structure.

**Returns**

0 if successful, -1 if unsuccessful.

**Example**

```
sdv_close(sd);
```

**Syntax**

```
#include "sdvlib.h"
int
sdv_close(sd)
SdvDev *sd;
```

**sdv\_multibuf(sd, nbufs)****Description**

Sets the number of multiple buffers to use in ring buffer continuous mode (see `sdv_image()`, `sdv_start_image()`, `sdv_wait_image()`). Additional buffers will speed acquisition and display. The number of buffers is limited only by the amount of host memory available.

**Arguments**

*sd*                   The pointer to the device structure.  
*nbufs*                The number of buffers to allocate.

**Returns**

0 if successful, -1 if unsuccessful

**Example**

```
sdv_multibuf(sd, 3);
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_multibuf(sd, nbufs)
SdvDev            *sd;
int                nbufs;
```

**sdv\_image(sd)****Description**

Initiate image acquisition if not already started, then wait for and return the address of the next available image. This routine essentially is the same as doing an `sdv_start_image()` followed by `sdv_wait_image()`, but as such doesn't allow you to process a previous image while waiting for the next one like you can do if you use the two calls separately.

**Arguments**

*sd*                   device struct, returned from `sdv_open`

**Returns**

Address of the next available image buffer that has been acquired.

**Example**

```
caddr_t p = sdv_image(sd);
```

**Syntax**

```
#include "sdvlib.h"

caddr_t sdv_image(sd)
SdvDev *sd;
```

**sdv\_start\_image(sd)****Description**

Initializes image acquisition. Returns without waiting for acquisition to complete. Used with `sdv_wait_image()`, which waits for the image to complete and returns a pointer to it.

**Arguments**

`sd`                    device struct, returned from `sdv_open`

**Returns**

0 if success, nonzero if ioctl fails

**Syntax**

```
#include "sdvlib.h"
sdv_start_image(sd)
SdvDev *sd;
```

**sdv\_wait\_image(sd)****Description**

Wait for the image started by `sdv_start_image()`. Will return immediately if the image started by the last call to `sdv_start_image()` is already complete.

Use `sdv_start_image()` to start image acquisition, and `sdv_wait_image()` to wait for it to complete. `sdv_wait_image()` returns the address of the image. You can start a second image while processing the first if you have used `sdv_multibuf()` to allocate 2 or more separate image buffers.

**Arguments**

`sd`                    device struct, returned from `sdv_open`

**Returns**

0 if success, nonzero if ioctl fails

**Syntax**

```
#include "sdvlib.h"

caddr_t
sdv_wait_image(sd)
SdvDev *sd;
```

**Example**

```
/* see also image.c example program */
sdv_multibuf(sd, 2);
sdv_start_image(sd);
while(1) {
    caddr_t image ;

    image = sdv_wait_image(sd); /* returns latest image */
    sdv_start_image(sd); /* start acquisition of next image */

    /* process and/or display image previously acquired here */
    printf("got image\n") ;
}
}
```

**sdv\_abort\_current(sd)****Description**

Aborts the current buffer and advances the DMA engine to the next buffer, without waiting for the current acquisition to complete. Works only with ring buffer mode acquisition\* (*sdv\_image()*, *sdv\_start\_image()*, *sdv\_wait\_image()*).

**Arguments**

*sd*                    device struct, returned from *sdv\_open*

**Returns**

0 if successful, -1 if unsuccessful.

**Syntax**

```
#include "sdvlib.h"

int sdv_abort_current(sd)
SdvDev *sd;
```

**sdv\_grab(sd)****Description**

Grab a frame from selected input, put in memory starting at *sd->image*. If continuous (see *sv\_enable\_continuous*), updates the memory continuously, grabbing a new frame each time one is available from the camera. Primarily for use when acquiring single images. Should not be used when ring buffer mode functions (*sdv\_image()*, *sdv\_start\_image()*, *sdv\_wait\_image()*) are in use.

**NOTE: *sdv\_grab()* is an older way of doing image acquisition and is included here primarily for backwards compatibility. Use *sdv\_start\_image()* and *sdv\_wait\_image()*, or just *sdv\_image()* if you are designing a new application.**

**Arguments**

*sd*                    The pointer to the device structure.

**Returns**

0 if successful, -1 if unsuccessful.

**Example**

```
sdv_grab(sd);
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_grab(sd)
SdvDev *sd;
```

**sdv\_grab\_average(sd)****Description**

Grab count frames from selected input, put in memory an average of the frames

**Arguments**

*sd*                   The pointer to the device structure.  
*count*                Number of frames to grab and average

**Returns**

Pointer to a buffer containing the averaged data. Uses a single buffer. This routine does not use ring buffering and allocates a separate buffer for the data, so if you are using ring buffering, the first use of this routine will result in a decrease of available memory by the size of one image.

**Example**

```
unsigned char *p = sdv_grab_average(sd);
```

**Syntax**

```
#include "sdvlib.h"

unsigned char *
sdv_grab_average(sd)
SdvDev *sd;
```

**sdv\_camera\_type(sd)****Description**

Get the name of the camera device in use

**Arguments**

*sd*                   The pointer to the device structure.

**Returns**

The name of the camera, as specified in the camera config file that was loaded by the initcam program on boot or subsequently by hand.

**Example**

```
printf("Camera Name: %s\n", sdv_camera_type(sd));
```

**Syntax**

```
#include "sdvlib.h"

char *
sdv_camera_type(sd)
SdvDev *sd;
```

**Files**

```
$SDVHOME/camera_config/*.cfg
```

**sdv\_set\_exposure(sd, value)****Description**

Sets the exposure time on the digital video device. Only applicable with camera devices that have software programmable exposure time. If the camera model is one that has a computer selectable front panel mode, then setting *value* to 0 will result in gain being selected from the front panel switch.

**Arguments**

*sd*                   The pointer to the device structure.

*value*                Exposure time, in milliseconds. The valid range depends on the camera. The minimum shutter speed, maximum shutter speed, and front panel availability, as set in the camera\_config file for the particular device, are available from the *sd->methods->shutter\_speed\_min*, *sd->methods->shutter\_speed\_max*, and *sd->methods->shutter\_speed\_frontp* structure elements, respectively.

**Returns**

0 if successful, -1 if unsuccessful.

**Examples**

```
sdv_set_exposure(sd, 5); /* milliseconds */
sdv_set_exposure(sd, 0) ;/* set exposure from camera front panel (only for
                           cameras with s/w selectable front panel mode) */
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_set_exposure(sd, value)
SdvDev *sd;
int value;
```

**sdv\_get\_exposure(sd)****Description**

Gets the exposure time on the digital video device. Only applicable with camera devices that have software programmable exposure time.

**Arguments**

*sd*                   The pointer to the device structure.

**Returns**

Exposure time. Units are milliseconds.

**Example**

```
int exposure = sdv_get_exposure(sd);
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_get_exposure(sd)
SdvDev *sd;
```

**sdv\_set\_gain(sd, value)****Description**

Sets the gain on the input device. Only applicable on devices with extended control capabilities, such as the Kodak Megaplug 1.6. If the camera model is one that has a computer selectable front panel mode, then setting *value* to 0 will result in gain being selected from the front panel switch.

**Arguments**

*sd*                   The pointer to the device structure.

*value*               Gain value. The valid range and presence of a front panel are device dependent, and can be found in the *methods.gain\_max*, *methods.gain\_min*, and *methods.gain\_frontp* elements in the SdvDev structure, defined in *sdvlib.h* and *camera.h*.

**Returns**

0 if successful, -1 if unsuccessful.

**Example**

```
sdv_set_gain(sd, 0); /* neutral gain */
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_set_gain(sd, value)
SdvDev *sd;
int value;
```

**sdv\_get\_gain(sd, gain)****Description**

Gets the gain on the video device. Only applicable on devices with extended control capabilities, such as the Kodak Megaplug 1.6. If the camera model is one that has a computer selectable front panel gain mode, then setting *value* to 0 will result in gain being selected from the front panel switch.

**Arguments**

*sd*                   The pointer to the device structure.  
*gain*                 Pointer to location to store gain value.

**Returns**

Gain value. The valid range is -128 to 128. The actual valid range is device-dependent. Value is also stored in *gain* pointer.

**Example**

```
sdv_get_gain(sd, &gain);
```

**Syntax**

```
#include "sdvlib.h"  
  
int  
sdv_get_gain(sd, gain)  
SdvDev *sd;  
int *gain;
```

**sdv\_set\_blacklvl(sd, value)****Description**

Sets the black level (offset) on the input device. Only applicable on devices with extended control capabilities, such as the Kodak Megaplug 1.6. If the camera model is one that has a computer selectable front panel mode, then setting *value* to 0 will result in gain being selected from the front panel switch.

**Arguments**

*sd*                   The pointer to the device structure.

*value*                Black level value. The valid range and presence of a front panel are device dependent, and can be found in the *methods.offset\_max*, *methods.offset\_min*, and *methods.offset\_frontp* elements in the *SdvDev* structure, defined in *sdvlib.h* and *camera.h*.

**Returns**

0 if successful, -1 if unsuccessful.

**Example**

```
sdv_set_blacklvl(sd, 0); /* neutral blacklevel */
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_set_blacklvl(sd, value)
SdvDev *sd;
int value;
```

**sdv\_get\_blacklvl(sd, blacklvl)****Description**

Gets the black level (offset) on the video device. Only applicable on devices with extended control capabilities, such as the Kodak Megaplug 1.6.

**Arguments**

*sd*                    The pointer to the device structure.

*blacklvl*            Pointer to location to hold blacklevel value.

**Returns**

Black level value. Also stored in location pointed to by *blacklvl*.

**Example**

```
sdv_get_blacklevel(sd, &blacklvl);
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_get_gain(sd, blacklvl)
SdvDev *sd;
int *blacklvl;
```

**sdv\_enable\_lock(sd, flag)****Description**

Enables or disables the shutter lock. Only applicable with camera devices that have software programmable shutter lock.

**Arguments**

*sd*                   The pointer to the device structure.

*flag*                 A value of 1 locks the shutter open. A value of 0 disables the shutter lock. The shutter is normally closed except during exposures.

**Returns**

0 if successful, -1 if unsuccessful.

**Example**

```
sdv_enable_lock(sd, 1); /* lock shutter */
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_enable_lock(sd, flag)
SdvDev *sd;
int flag;
```

**sdv\_setmem(sd, numbufs, bufs)****Description**

Frees any memory already allocated by `sdv_open()` and `sdv_multibuf()` and sets the image buffers to user allocated buffers. Normally `sdv_open()` and `sdv_multibuf()` allocate their own buffers, however, some applications require that their own buffers be used. This routine is provided to allow you to do this.

Setting *buf* to 0 disables copying.

**Arguments**

|                |                                      |
|----------------|--------------------------------------|
| <i>sd</i>      | The pointer to the device structure. |
| <i>numbufs</i> | the number of buffers                |
| <i>bufs</i>    | pointer to pointers to bufs          |

**Returns**

0 if successful, -1 if unsuccessful.

**Example**

```
sdv_setmem(sd, 2, bufs); /* address of buffer */
```

**Syntax**

```
#include "sdvlib.h"

int
sdv_setmem(sd, numbufs, bufs)
SdvDev *sd;
long size ;
caddr_t *bufs ;
```

## **sdv\_set\_remap(sd, remap)**

### **Description**

Sets the hardware to remap the upper or lower eight values to avoid conflict with the window manager color palette, which often uses the bottom or top entries for basic screen colors.

**NOTE:** While this function can make it easier to display grayscale images on a color display using window managers such as OpenLook that use the lower 8 or fewer color map values, it can result in some loss of resolution in the data. If full data integrity is required, turn off hardware remapping by setting *remap* to `FILTER_OFF` before capturing data to be saved to a file or displayed on another medium.

### **Arguments**

*sd*                   The pointer to the device structure.

*remap*                Either or both of `FILTER_F0_ON` or `FILTER_00_ON`, or `FILTER_OFF`.

If `FILTER_00_ON` is specified, this function remaps the lowest sixteen values to the next-to-lowest 8 values; that is, 0 and 1 map to 8, 2 and 3 map to 9, 4 and 5 map to 10, etc. Similarly, `FILTER_F0_ON` causes the uppermost sixteen values to be remapped to the next-to-uppermost eight values.

`FILTER_OFF` disables hardware remapping.

### **Returns**

0 if successful, -1 if unsuccessful

### **Example**

```
sdv_set_remap(sd, FILTER_00_ON); /* remap lower values */
```

### **Syntax**

```
#include "sdvlib.h"

int
sdv_set_remap(sd, remap)
SdvDev *sd;
int remap;
```

**sdv\_overrun(sd)****Description**

Determines whether there was data overrun (partial frame) on the last acquire.

**Arguments**

*sd*                    device struct, returned from `sdv_open`

**Returns**

Number of bytes of data remaining from last acquire. 0 indicates no overrun.

**Syntax**

```
#include "sdvlib.h"

int
sdv_overrun(sd)
SdvDev *sd;
```

**sdv\_picture\_timeout(sd, tenths)****Description**

Sets length of time to wait for data on acquisition before timing out. A default value for this based on the size of the image produced by the camera device in use is calculated and set by `sdv_open()`. This routine will override that value. A value of 0 tells the driver to wait forever for the amount of data requested.

**Arguments**

*sd*                    device struct, returned from `sdv_open`

*tenths*                number of tenths of a second to wait for timeout, or 0 to block waiting for data

**Returns**

0 if successful, nonzero on failure

**Syntax**

```
#include "sdvlib.h"

int
sdv_picture_timeout(sd, tenths)
SdvDev *sd;
int tenths ;
```

**sdv\_debug(flag)****Description**

Sets the debug level of the sdv library. This results in debug output being output to the screen by sdv library calls. The same thing can be accomplished by setting the SDVDEBUG environment variable to 1. See also the SDVx\_DEBUG\_LEVEL and SDVx\_DEBUG\_INTR ioctl calls, and the program setdebug.c for information on using the device driver debug flags.

**Arguments**

*flag*                    flags debug output on (nonzero) or off (zero).

**Returns**

0 if successful, nonzero on failure

**Syntax**

```
#include "sdvlib.h"

int
sdv_debug(flag)
int flag ;
```

**sdv\_write\_rasfile(fname, addr, x\_size, y\_size)****Description**

Outputs a 1 band, 8 bit image to a Sun Raster format file

**Arguments**

*fname*                    the name of the output file  
*addr*                     the address of the image data (8 bits per pixel)  
*x\_size*                    width in pixels of image  
*y\_size*                    height in pixels of image

**Returns**

0 on success, -1 on failure

**Syntax**

```
#include "sdvlib.h"

sdv_write_rasfile(fname, addr, x_size, y_size)
char                    *fname;
caddr_t                addr;
int                     x_size;
int                     y_size;
```

**sdv\_write\_rasfile24(fname, addr, x\_size, y\_size)****Description**

Outputs a 3 band, 8 bit image to a Sun Raster format file

**Arguments**

|               |  |
|---------------|--|
| <i>fname</i>  | the name of the output file                              |
| <i>addr</i>   | the address of the image data (24 bits per pixel -- BGR) |
| <i>x_size</i> | width in pixels of image                                 |
| <i>y_size</i> | height in pixels of image                                |

**Returns**

0 on success, -1 on failure

**Syntax**

```
#include "sdvlib.h"

sdv_write_rasfile(fname, addr, x_size, y_size)
char             *fname;
caddr_t         addr;
int             x_size;
int             y_size;
```

**sdv\_write\_image(fname, addr, x\_size, y\_size, istride)****Description**

Outputs a 1 band, 8 bit image to a Sun raster format file, with stride. This function can be used to output a partial image. For example, to output the top left 100x100 pixels of the image, you should specify *x\_size* of 100 and *y\_size* \* 100, and an *istride* of the actual width of the image

**NOTE: Note: Use *sdv\_write\_rasfile* to output a full image in the most efficient way.**

**Arguments**

|                |  |
|----------------|--|
| <i>fname</i>   | the name of the output file  |
| <i>addr</i>    | the address of the image data (8 bits per pixel)                                       |
| <i>x_size</i>  | width in pixels of image   |
| <i>y_size</i>  | height in pixels of image  |
| <i>istride</i> | number of pixels to skip from one the end of one scanline to the beginning of the next |

**Returns**

0 on success, -1 on failure

**Syntax**

```
#include "sdvlib.h"

sdv_write_image(fname, addr, x_size, y_size,istride)
char          *fname ;
caddr_t       addr ;
int           x_size ;
int           y_size ;
int           istride ;
```

**sdv\_write\_image24(fname, addr, x\_size, y\_size, istride)****Description**

Outputs a 3 band, 8 bit image to a Sun raster format file, with stride. This function can be used to output a partial image. For example, to output the top left 100x100 pixels of the image, you should specify *x\_size* of 100 and *y\_size* \* 100, and an *istride* of the actual width in pixels of the image

**NOTE: Note: Use *sdv\_write\_rasfile24* to output a full image in the most efficient way.**

**Arguments**

|                |  |
|----------------|--|
| <i>fname</i>   | the name of the output file  |
| <i>addr</i>    | the address of the image data (24 bits per pixel - BGR)                                |
| <i>x_size</i>  | width in pixels of image   |
| <i>y_size</i>  | height in pixels of image  |
| <i>istride</i> | number of pixels to skip from one the end of one scanline to the beginning of the next |

**Returns**

0 on success, -1 on failure

**Syntax**

```
#include "sdvlib.h"

sdv_write_image24(fname, addr, x_size, y_size, istride, pixel_stride)
char          *fname ;
caddr_t       addr ;
int           x_size ;
int           y_size ;
int           istride ;
int           pixel_stride ;
```

**sdv\_serial\_read(sd, buf, count)****Description**

Does a serial read over the RS-422 lines.

**Arguments**

|              |   |
|--------------|---|
| <i>sd</i>    | device struct, returned from <i>sdv_open</i>                          |
| <i>buf</i>   | pointer to data buffer -- must be pre-allocated to <i>count</i> bytes |
| <i>count</i> | number of bytes to be read  |

**Returns**

0 if success, nonzero if ioctl fails

**Syntax**

```
#include "sdvlib.h"

sdv_serial_read(sd, buf, count)
SdvDev *sd;
unsigned char *buf ;
int count;
```

**sdv\_serial\_write(sd, buf, count)****Description**

Does a serial write over the RS-422 lines. This command is applicable only with cameras that use the serial AIA interface protocol, such as the Kodak 'i' cameras and the Hamamatsu C4880.

**Arguments**

*sd*                    device struct, returned from `sdv_open`  
*buf*                    pointer to data buffer -- must be at least *count* bytes  
*count*                  number of bytes to write

**Returns**

0 if success, nonzero if ioctl fails

**Syntax**

```
#include "sdvlib.h"

sdv_serial_write(sd, buf, count)
SdvDev *sd;
unsigned char *buf ;
int count;
```

**sdv\_serial\_command(sd, cmd)****Description**

Sends a serial command to the camera. This is essentially the same as `sdv_serial_write` except that it is not necessary to specify the length of the string.

This command is applicable only with cameras that use the serial AIA interface protocol, such as the Kodak 'i' cameras and the Hamamatsu C4880. The programmer is responsible for verifying that the commands sent are valid for the camera in use. See your camera's User's Guide for a list of valid commands for your camera.

**Arguments**

*sd*                    device struct, returned from `sdv_open`  
*cmd*                    string containing a serial command to be sent to the camera

**Returns**

0 if success, nonzero if ioctl fails

**Example**

```
sdv_serial_command(sd, "DEF ON"); /* set defect correction ON */
```

**Syntax**

```
#include "sdvlib.h"

sdv_serial_command(sd, cmd)
SdvDev *sd;
char *cmd ;
```

**sdv\_start\_hardware\_continuous(sd, frames)****Description**

Useful when you need to capture every frame of a camera when there is no delay between the end of one image and the start of another. Called just before capturing one or more frames in hardware continuous mode. In this mode, the driver initializes hardware continuous mode by setting the hardware continuous bit in the data path register and then flushing the buffers. Then when the first read is performed by the application the camera will produce a continuous stream of data. No further fifo resets are performed until hardware continuous mode end automatically or via `sdv_stop_hardware_continuous()`. `sdv_stop_hardware_continuous()` is used to end hardware continuous mode if it is not done automatically by the driver.

**Arguments**

*sd* device struct, returned from `sdv_open`

*frames* the number of frames to capture before the driver automatically shuts off hardware continuous mode. If zero, hardware continuous mode will remain in effect until `sdv_stop_continuous()` is called.

**Returns**

void

**Syntax**

```
#include "sdvlib.h"

sdv_start_hardware_continuous(sd, frames)
SdvDev *sd;
unsigned int frames ;
```

**sdv\_stop\_hardware\_continuous(sd)****Description**

Stops hardware continuous mode See `start_hardware_continuous()` for further description.

**Arguments**

*sd* device struct, returned from `sdv_open`

**Returns**

void

**Syntax**

```
#include "sdvlib.h"

sdv_stop_hardware_continuous(sd)
SdvDev *sd;
```

## **foi\_parity\_error**

### **Description**

Checks to determine if a parity error has occurred since the last time this routine was called and returns 0 if not, 1 if so, and -1 if the routine is not supported for a particular device or an illegal argument was provided.

### **Arguments**

*sd*                    device struct, returned from `sdv_open`

### **Returns**

0 if no error; 1 if parity error; -1 if the routine is not supported or if an illegal argument was provided.

### **Syntax**

```
#include "sdvlib.h"

foi_parity_error(sd)
SdvDev *sd;
```

## Camera-specific Issues

This section provides installation, protocol, and operating instructions for specific camera models where necessary, as well as describing extension libraries for external camera control.

### Kodak MEGAPLUS 1.6

Some Kodak MEGAPLUS cameras with a separate Camera Control Unit (CCU) have an RS-232/RS-422 switch on the back of the CCU. This switch must be set to RS-422 in order for the SDV-RCI to be able to control the camera. Also, the front panel **Mode** switch must be set to **Computer** and the **Shutter** switch must be **On**.

### Kodak MEGAPLUS i Model Cameras (1.4i, 1.6, 1.6i, 1.68i, 4.2i) and ES 1.0

#### *Installation*

When installing the SDV-RCI software, and when rebooting the system, turn the ES and i model cameras **on**. If the camera was off when the software was installed or the machine was rebooted (or if you're not sure):

1. Turn the camera on.
2. Run *sdvload*.
3. Then run the required applications.

#### *Interface Protocol*

The Kodak ES and i model cameras are configured at the factory for either RS-232 or RS-422 control. Because the SDV-RCI board controls the camera through the RS-422 lines, you must specify RS-422 control when ordering your camera from Kodak. If your camera does not operate correctly with the EDT SDV-RCI module (for example, the camera does not click when an acquire is requested, or it clicks continuously all the time), it is likely that the camera is configured for RS-232 control.

To determine whether this is the case, you can use the *serial\_test* program included with the *sdvrci* software. This program tests whether the board is able to talk to the camera over RS-422 lines, and can also configure specific camera parameters. To run the test:

1. Install the SDV-RCI software.
2. Change to the directory where the software was installed (normally */opt/EDTsdv* or */var/EDTsdv*).
3. Enter:

```
serial_test "STS?"
```

Example 1 shows a sample response, although some parameter values may be slightly different.

```
Read 78 bytes:
```

```
DEF OF
GAE 6
BKE 0
MDE TR
SHE ON
EXE 20
TRM P
TRE 1
STP P
```

### Example 1. serial\_test Output

Example 1 shows a clean status: the board is able to communicate with the camera. If the camera is connected properly and the hardware and software installation was done according to the instructions, but the test instead results in garbage characters output to the screen, then your camera is probably configured for RS-232 control. If so, contact your Kodak representative and arrange to reconfigure it for RS-422 mode.

#### ***Sending Commands to the Camera***

You can also use the *serial\_test* program to send to the camera any of the function and query commands described in the *Kodak MEGAPLUS Camera User's Manual*. For example, to turn on Defect Conceal, enter:

```
serial_test "DEF ON"
```

See **sdv\_serial\_command(sd, cmd)** on page 34 for instructions on sending commands to the camera from a C program.

**NOTE:** Application programs such as *initcam*, *dv*, and *take* modify some camera parameters.

#### ***Kodak ES 1.0/SC Operating Mode***

The ES 1.0/SC is configurable for either dual-channel or single-channel output. Either will work—however, unless you have a relatively fast CPU and display hardware, the speed advantages possible with dual-channel mode (approximately 30 frames per second vs. 15 frames per second) can be negated by the extra time the software takes to merge the interlaced fields. Therefore, unless you are running on an UltraSPARC or a SPARCstation 20/SX in static gray visual mode, we recommend that you choose *Kodak MEGAPLUS ES 1.0/SC (single ch.)* when you install the software. (If the software is already installed, you can change your choice by once more running *sdvrcicamera*.)

Single-channel mode is not available on the ES 1.0 (non-SC), so choose "*Kodak MEGAPLUS ES 1.0*" for this camera.

#### ***Kodak ES 1.0 and ES 1.0/SC Extended Functions***

The ES 1.0 and ES 1.0/SC cameras have some functionality that is not available on other Kodak cameras, such as black level and gain balance (**BKB** and **GAB**). EDT provides no corresponding options or buttons in the *dv* or *take* applications to set these parameters. If you want to change these settings, use the *serial\_test*. For example, to set the black level balance to 100, enter:

```
serial_test "BKB 100"
```

## Troubleshooting

While every effort has been made to ensure that your installation and use of the SDV-RCI is as painless and trouble-free as possible, there is always a chance that problems will occur. Check the following if you encounter any difficulties.

### If you encounter problems installing the software:

Make sure the software release disk is the correct one for your version of the Operating System. SunOS 4.1.3 software will not install properly on Solaris 2.x systems, and vice-versa.

Make sure you have the latest release of the software. See **Contacting EDT** on page 48 for instructions on downloading the latest release from our *WWW* or *ftp* site.

Make sure you are logged in as *root*.

Do not try to install using the File Manager. Solaris 2.x software diskettes are in *pkgadd* format. SunOS 4.1.x diskettes are in compressed *tar* format. Neither of these formats is recognized by File Manager.

**NOTE: In earlier releases, the installation procedure specified using File Manager as a method to alert the OS to the presence of a diskette. This is no longer necessary as it has been replaced by a call to *volcheck*.**

Do not try to extract and install files by hand and circumvent the installation process. The installation procedure updates system files and creates links as well as creating new files.

### If you have problems acquiring images using SDV Tool (dv), xwin, or other application programs:

Make sure the camera device is on and is receiving power.

Check all interface cable connections. Turn off the camera device, unplug the cable at both ends, reattach it, and turn the camera device back on. It is not necessary to power down the SPARCstation before connecting or disconnecting the *sdvrci* interface cable.

Make sure the SDV-RCI software has been installed and the driver is running. To do so, on SunOS 4.1.3, enter:

```
modstat
```

and checking to see whether the resulting line or lines of installed device drivers includes one with a *mod name* of EDT, *sdv*.

On Solaris 2.x, enter:

```
modinfo | egrep "Module | sdv" .
```

If this fails, change to the directory where the EDT SDV-RCI software was installed and enter:

```
make unload load
```

Then rerun the *modinfo* or *modstat* command. If it still fails, reinstall the *sdvrci* software as explained in **Installing the Software** on page 4.

Make sure you have an up-to-date version of the operating system, and that it has been upgraded to the current patch level.

If the camera you are using has a mode switch that has a computer controlled setting (for example, the **Computer** setting of the **MODE** knob on some Kodak MEGAPLUS cameras), make sure the knob is set to this setting.

Make sure you don't have a marginal combination of platform and camera. Symptoms include broken, slanted, or otherwise corrupted images, or unreasonably slow acquire times. See **Platform and Operating System** on page 2.

Make sure you have modified your `LD_LIBRARY_PATH` as described in **Before Running the SDV Tool** on page 10 (Solaris 2.x only).

Make sure you have configured the driver for the camera you are using. The name of the camera appears at the top of the `sdvrci` (`dv`) main window. If it is incorrect, you may or may not be to acquire images, but they are likely to be partial or skewed, or you may suffer long acquire times. If you find that you have chosen the wrong camera:

1. Exit the application.
2. Log in as root.
3. Enter:  

```
sdvcamera
```
4. Select the correct camera (see **Changing the Camera Model** on page 6).

If the camera fails to respond for any reason, cycle power on the camera and try again. If it still doesn't work, run `sdvload`. This script reinitializes the driver with the camera information selected when you ran `sdvcamera`.

### **If you encounter problems compiling, linking or running your application program, or if you get "Illegal ioctl" messages on the console when you run your program:**

If you have updated the `sdvrci` software, make sure you have updated all copies and links to header (`.h`) files, then recompile all program modules that have SDV library calls in them and relink the application.

If you are using the XIL library, make sure you have modified your `LD_LIBRARY_PATH` as described in **Before Running the SDV Tool** on page 10

### **And if all else fails...**

Contact EDT through one of the following channels:

|        |                    |
|--------|--------------------|
| e-mail | tech@edt.com       |
| WWW    | http://www.edt.com |
| Fax    | (503) 690-1243     |
| Phone  | (503) 690-1234     |

Include the following in any correspondence with EDT:

- Model of SPARCstation in use.
- Operating system version: for example, Solaris 2.5, SunOS 4.1.4).
- Window manager being used: for example, OpenWindows, Motif Window Manager (mwm).
- Amount of memory in the system.

- Camera in use.
- Application program that shows the problem: for example, *dv*, *xwin*, other.
- Model of SBus expansion box (if any).
- SDV-RCI software version or date. Look on the back of the software diskette to find the date that the disk was manufactured. Use the *modstat* (SunOS 4.1.x) or *modinfo* (Solaris 2.x) command to determine the software version.

## Connector Pinout

The Digital Video Remote Camera Interface uses a high-density 80-pin I/O connector.

The high-density mating connector is an AMP connector, AMP part number 749111-7, with a straight-shielded backshell (AMP P/N 749196-1) or right angle backshell (AMP P/N 749205-1).

The following pinout diagram describes the connection from the sdvrci board to the cable. The connection from the cable to the camera is camera-dependent. Contact Engineering Design Team, Inc. for connector pinouts for cabling to specific cameras.

***NOTE:*** Do not connect your own circuits to the unused pins, as they may be internally connected to the SDV-RCI. Also, some unused pins may be used only when connected to specific cameras.

| AMP | Signal   | AMP | Signal      |
|-----|----------|-----|-------------|
| 1   | Ground   | 41  | Ground      |
| 2   | not used | 42  | not used    |
| 3   | MSB4+    | 43  | MSB0+       |
| 4   | MSB4-    | 44  | MSB0-       |
| 5   | MSB5+    | 45  | MSB1+       |
| 6   | MSB5-    | 46  | MSB1-       |
| 7   | MSB6+    | 47  | MSB2+       |
| 8   | MSB6-    | 48  | MSB2-       |
| 9   | MSB7+    | 49  | MSB3+       |
| 10  | MSB7-    | 50  | MSB3-       |
| 11  | MSB12+   | 51  | MSB8+       |
| 12  | MSB12-   | 52  | MSB8-       |
| 13  | MSB13+   | 53  | MSB9+       |
| 14  | MSB13-   | 54  | MSB9-       |
| 15  | MSB14+   | 55  | MSB10+      |
| 16  | MSB14-   | 56  | MSB10-      |
| 17  | MSB15+   | 57  | MSB11+      |
| 18  | MSB15-   | 58  | MSB11-      |
| 19  | not used | 59  | not used    |
| 20  | not used | 60  | not used    |
| 21  | not used | 61  | not used    |
| 22  | not used | 62  | not used    |
| 23  | not used | 63  | not used    |
| 24  | SCNTLI+  | 64  | PSTRB+      |
| 25  | SCNTLI-  | 65  | PSTRB-      |
| 26  | not used | 66  | LINE+       |
| 27  | not used | 67  | LINE-       |
| 28  | not used | 68  | FRME+       |
| 29  | not used | 69  | FRME-       |
| 30  | not used | 70  | not used    |
| 31  | not used | 71  | not used    |
| 32  | SCNTLO+  | 72  | MC0         |
| 33  | SCNTLO-  | 73  | not used    |
| 34  | not used | 74  | MC1         |
| 35  | not used | 75  | not used    |
| 36  | not used | 76  | MC2         |
| 37  | not used | 77  | not used    |
| 38  | not used | 78  | FRMRST/EXP+ |
| 39  | not used | 79  | FRMRST/EXP- |
| 40  | Ground   | 80  | Ground      |

Table 1. Connector Pinout

## Registers

The Digital Video Remote Camera Interface is configured and controlled with the following registers. The addresses given are on the remote camera interface and map to the address space on the SCD-FOI starting at 0x0001.00C0.

### FOI\_Read\_Return Register

The FOI\_Read\_Return register is not implemented because nothing on the SDV-RCI can initiate a read of the SCD-FOI. Therefore, address 0x00 is not used.

### FOI\_Read\_Trigger Register

The FOI\_Read\_Trigger register is an 8-bit write-only register at address 0x01. When the SCD-FOI writes to this register with the address of an SDV-RCI register, the SDV-RCI responds by writing to the SCD-FOI Read\_Return register with the contents of the register specified by the address. The Xilinx firmware handles this register automatically, allowing the SCD-FOI to read registers on the SDV-RCI.

| Bit  | Description  |
|------|--|
| D0–7 | Address of the SDV-RCI register to be read by the SCD-FOI. |

Table 2. The FOI\_Read\_Trigger Register

### FOI\_Board\_ID Register

The FOI\_Board\_ID register is an 8-bit read-only register at address 0x05. Driver software on the host computer can read this register to determine the installed firmware version (*.rbt* file).

| Bit   | Description                      |
|-------|----------------------------------|
| B10–7 | Firmware version identification. |

Table 3. FOI\_Board\_ID Register

### FOI\_Board\_Select Register

The FOI\_Board\_Select register is an 8-bit write-only register at address 0x06. This register is used to select which of the four daughter boards will receive and transmit data. This register operates in the same manner before and after the Xilinx has been programmed.

| Bit   | Description  |
|-------|--|
| BS0–1 | Selects the specified daughter board (one of 0–3) to transmit data.<br>If bit 7 is not asserted, the selected board is also the only one to receive data.. |
| BS2–6 | not used   |
| B7    | A value of 1 indicates that all daughter boards receive data.  |

Table 4. FOI\_Board\_Select Register

## FOI\_Xilinx\_Prog Register

The FOI\_Xilinx\_Prog register is an 8-bit read-write register at address 0x07, used to program the Xilinx upon power-up and to return debug information for the SCD-FOI. This register is written to, and the Xilinx is loaded, automatically by the `sfoiload` utility provided with the SCD-FOI. After the Xilinx is loaded, assert no bits of this register except the XLNX\_LOAD bit if you need to reset the Xilinx.

| Bit  | Name      | Description  |
|------|-----------|--|
| R0   | XLNX_LOAD | Strobe this bit to reset the SDV-RCI to its state upon power-up, ready to load the Xilinx.   |
| R1   | ERR_CLEAR | Strobe this bit to stop the board from writing its hardware ID until the next time XLNX_LOAD is asserted. Strobing this bit also clears the NOCARRIER and RVSEERR status bits (see R3 below).  |
| R2   | PGM_OFF   | The Xilinx PROGRAM pin is asserted until this bit is strobed.  |
| R3   | GET_STAT  | Strobe this bit to receive a byte of status information in the SCD-FOI read return register:<br>0                   not used<br>1   RVSEERR       Violation symbol received<br>2   NOCARRIER    No carrier detected.<br>3   XINIT          State of the Xilinx XINIT pin.<br>4   XDONE         State of the Xilinx DONE pin.<br>5–7               not used<br><br><b>NOTE:</b> In order to avoid writing invalid data, do not strobe GET_STAT until after you have first strobed ERR_CLEAR to stop the board from writing its hardware ID. |
| R4   | XLNX_CCLK | Strobe this bit to send the value of the DIN bit.  |
| R5   | XLNX_DIN  | The bit used for the serial data stream containing the program. When the XDONE status bit is asserted (see R3), the IC has finished accepting the program data.  |
| R6–7 |           | not used   |

Table 5. FOI\_Xilinx\_Prog Register

## Powering Up and Resetting the SDV-RCI

If power to the SDV-RCI is lost and then regained, the remote camera interface must reinitialize itself. While doing so, it cannot accept data from the SCD-FOI board in the SBus host. Therefore, when the SDV-RCI first powers up, it repeatedly sends a hardware ID of 0x80 to the Remote Flag register on the SCD-FOI in order to report that it is reinitializing. You can stop this activity by strobing the ERR\_CLEAR bit of the FOI\_Xilinx\_Prog register.

If the SDV-RCI has detected errors in its incoming data—either a loss of carrier or a received violation symbol— then the NOCARRIER or RVSEERR bits will be true. If either or both are true, then the least significant bit of the hardware ID is set and the SDV-RCI will send a value of 0x81 instead of 0x80.

During initialization, the Xilinx is loaded with program data. After the Xilinx has been programmed, the Mach PAL in the SDV-RCI can no longer broadcast; instead, it can only respond to a strobe of the XLNX\_LOAD bit of the FOI\_Xilinx\_Prog register by resetting the Xilinx to its initial state, allowing it to be reprogrammed if necessary.

## Direction\_Control Register

The Direction\_Control register is a 16-bit read-write register at address 0x08.

This register determines whether the physical drivers or receivers on the interface are inputs or outputs. Setting the same pins to serve as both input and output is useful for testing. Setting the same pins to serve as neither input nor output is not useful.

For example, to implement the interface documented in the connector pinout shown on page 43, set this register as follows:

|           |                    |
|-----------|--------------------|
| Low bits  | 4–7, 9, 11, 12, 14 |
| High bits | 0–3, 8, 10, 13, 15 |

To do so, send the value 0x5A0F.

| Bit  | Description                                  |
|------|--|
| DC15 | Pins 32–39 are inputs when this bit is low.  |
| DC14 | Pins 24–31 are inputs when this bit is low.  |
| DC13 | Pins 72–79 are inputs when this bit is low.  |
| DC12 | Pins 64–71 are inputs when this bit is low.  |
| DC11 | Pins 72–79 are outputs when this bit is low. |
| DC10 | Pins 64–71 are outputs when this bit is low. |
| DC9  | Pins 32–39 are outputs when this bit is low. |
| DC8  | Pins 24–31 are outputs when this bit is low. |
| DC7  | Pins 11–18 are inputs when this bit is low.  |
| DC6  | Pins 3–10 are inputs when this bit is low.   |
| DC5  | Pins 51–57 are inputs when this bit is low.  |
| DC4  | Pins 43–50 are inputs when this bit is low.  |
| DC3  | Pins 51–57 are outputs when this bit is low. |
| DC2  | Pins 43–50 are outputs when this bit is low. |
| DC1  | Pins 11–18 are outputs when this bit is low. |
| DC0  | Pins 3–10 are outputs when this bit is low.  |

**Table 6. The Direction\_Control Register**

## Specifications

The Digital Video Remote Camera Interface conforms to the following specifications.

### Device Data Transfer

Format            Programmable

### Software

Drivers for Solaris 1.x (Sun OS 4.1.x) and Solaris 2.x (Sun OS 5.x.)

### Power

5 V at 2 A

### Environmental

Temperature    Operating: 10 to 40 C  
                  Nonoperating: -20 to 60 C

Humidity        Operating: 20 to 80% noncondensing at 40 C  
                  Nonoperating: 95% noncondensing at 40 C

### Physical

Dimensions    16" x 10.5" x 1.75"

Weight with one daughter board  
                  8.75 lbs.

Each additional daughter board  
                  4.4 oz.

## Contacting EDT

A variety of services, from sales information to updated manuals to technical support, is available through EDT's World Wide Web site, at

`http://www.edt.com`

If you have had the board for a period of time before installing it, we recommend you get the latest software over the internet to ensure you have all the latest enhancements. You can do this by accessing our World Wide Web page, and selecting **Technical Support**, then **Software**, then specifying which operating system version and driver to download. Alternately, you can ftp the software directly using the following procedure:

1. FTP to the edt ftp server by typing

```
ftp ftp.edt.com
```

2. Login as anonymous. Password is your e-mail address.
3. Change to the appropriate directory for the version of the operating system you are running. If you are running SunOS 4.1.x (Solaris 1.x), type

```
cd /pub/sdv/4x
```

If you are running SunOS 5.x (Solaris 2.x), type

```
cd /pub/sdv/5x
```

4. Set binary transfer mode by typing

```
bin
```

5. Download the software and README file:

```
get EDTsdv.tar.Z
```

```
get README
```

6. Exit ftp:

```
quit
```

7. Follow the instructions in the README file to extract and install the software from the .z file.

The latest release of this manual is also available, via the World Wide Web site under **Technical Support, Manuals**, or through ftp to ftp.edt.com in `/pub/manuals/sdvrci.ps.Z` (postscript format, compressed).

## Appendix A Input and Output

Ordinarily, you will access the SDV-RCI device by means of **The Digital Video Library** described in the previous section, as it provides a complete, simple, high level interface to the device. This section is provided primarily for information, and we recommend you avoid including direct I/O calls or any of the *ioctl*s described in this section in your programs unless absolutely necessary. If you must do so, study the library source code carefully before implementing any direct accesses to the device driver through *read()*, *write()* or *ioctl()*.

To perform DMA, *read()* and *write()* system calls are used. Parameters to these calls must supply the appropriate file descriptor, the address of the data, and the number of bytes to transfer.

The SDV-RCI is configured and controlled with arguments to the *ioctl(2)* system call. Parameters to the *ioctl()* system call must supply the appropriate file descriptor, the specific parameter to use (see below), and the address of the data. The file *sdv.h* defines these parameters to *ioctl()* for the SDV-RCI driver.

Each call to *ioctl()* with a *sdvrci* parameter performs one of three functions:

- It reads or writes a specific register on the device, or
- it reads or writes a software variable in the device driver.

### *ioctl()* Parameters

The following *ioctl* parameters are used in the SDV software library, diagnostics, and some older example programs. Others may be defined, but are used for Engineering Design Team's internal purposes only.

**EDTx\_DEBUG** Set (S) or get (G) the debug interrupt level. A level of 1 reports driver status when *modinfo* (Solaris) or *modstat* (SunOS 4.1.x) is run. Other values are used for purposes internal to EDT. Provide an unsigned short as an argument.

**SDVG\_INTFC\_STAT** Returns device-specific status information. This information varies depending upon the device being driven and the programming of the Xilinx registers. Provide an unsigned character as an argument.

**SDVG\_QUEUE** Returns the number of input/output requests that have been queued since the device was opened. Provide an unsigned integer as an argument.

**SDVG\_IO\_COUNT** Returns the number of input/output transfers that have been started since the device was opened. Provide an unsigned integer as an argument.

**SDVG\_DIR\_CTRL** Get the value of the direction control register. Provide an unsigned short as an argument.

**SDV\_EXPOSE** Open and close the shutter for the appropriate length of time as set by the shutter speed. No argument is required.

**SDVx\_SPEED** Set (S) or get (G) the shutter speed—the exposure time— in ten millisecond increments. Provide an unsigned character as an argument.

**SDVG\_SHUTTER\_TIME** Get the amount of time that the shutter has yet to remain open, in ten millisecond increments. Provide an unsigned character as an argument.

- SDVS\_MODE** Set the acquisition mode—either single acquisition or continuous acquisition. Provide an unsigned character as an argument.
- SDVS\_LOCK\_SHUTTER**  
Lock the shutter open with an argument of 1, or unlock and close it with an argument of 0. Provide an unsigned character as an argument.
- SDVG\_COUNT** Get the number of bytes remaining in the current DMA. Provide an unsigned integer as an argument.
- SDVx\_WIDTH** Set (S) or get (G) the width of the video image. Provide an unsigned short as an argument.
- SDVx\_HEIGHT** Set (S) or get (G) the height of the video image. Provide an unsigned short as an argument.
- SDVx\_DEPTH** Set (S) or get (G) the depth (number of bits per pixel) of the video image. Provide an unsigned short as an argument.
- SDVx\_GAIN** Set (S) or get (G) the gain. Provide a signed integer as an argument.
- SDVx\_BLACKLVL**  
Set (S) or get (G) the black level, which controls image brightness. Provide an unsigned short as an argument. Provide a signed integer as an argument.
- SDV\_MAP\_FX** Remaps the uppermost or lowest 16 pixel values in order to leave 16 free slots in the color map for the window manager to use. The constants `FILTER_F0_ON` and `FILTER_00_ON` are defined in *camera.h*. An argument of `FILTER_F0_ON` remaps the top 16 pixel values; an argument of `FILTER_00_ON` remaps the bottom 16 pixel values. Provide an unsigned short as an argument.
- EDTS\_EODMA\_SIG**  
Register an end-of-DMA signal when the next DMA operation has been completed. The third argument to the *ioctl* is the address of an unsigned integer specifying the number of the signal to send to the calling process. (SIGIO is recommended, as its purpose is to signal an I/O event.) This registration produces one occurrence of a signal; the signal handler must reregister each time a signal is required. Provide an unsigned short as an argument.
- EDTS\_NBUFIO** Initiates continuous ring-buffer mode. The third argument to the *ioctl* is the address of an unsigned integer specifying the number of buffers in the ring: legal values are between 1 and 40. The driver waits until it has received the specified number of requests for DMA operation, then allocates operating system resources to each buffer, and finally performs continuous DMA to each buffer on a round-robin basis, starting with the first request and wrapping to the beginning again after the last.
- EDTG\_NBUFIO** Returns the number of the buffer that has most recently completed DMA and is available for processing. Provide an unsigned integer as an argument.
- EDT\_FREE\_BUF**  
Turns off continuous ring-buffer mode. No argument is required.
- EDTG\_DONECOUNT**  
Use with continuous ring-buffer mode (**EDTS\_NBUFIO**). Get the count of the last buffer completed by the driver. The count starts at 0 and increments each time DMA on a buffer completes. Provide an unsigned integer as an argument.

**EDTS\_WAKEUP\_DONECOUNT**

Use with continuous ring-buffer mode (**EDTS\_NBUFIO**) and **EDTG\_DONECOUNT**. This *ioctl* does not return until the count of buffers completed reaches the count supplied in the third argument to the *ioctl*. This causes the application to wait until the driver has performed the specified number of DMA operations. If the specified count has already been reached it returns immediately.

The driver counter wraps at  $2^{32}$ , and the driver is implemented to handle this behavior correctly. Therefore let the count in your application wrap as well.

Provide an unsigned integer as an argument.

**EDTS\_LOCKSTEP**

Use with continuous ring-buffer mode (**EDTS\_NBUFIO**). Initiates lockstep mode, in which the driver waits for notification from the application before proceeding with DMA. The number of buffers processed before waiting are specified in the value of the address pointed to by the third argument to this *ioctl*. The driver then waits until it receives notification by means of the following *ioctl*. **EDT\_FREERUN** turns off lockstep mode. Provide an unsigned integer as an argument.

**SDV\_START\_CONTINUOUS**

Enables hardware continuous acquisition mode, if your camera model supports this feature. Declare a *framecount* variable as an unsigned integer. Assign *framecount* the number of frames you wish to acquire; the driver automatically disables hardware continuous acquisition mode when it has acquired the specified number of frames. Pass *framecount*'s address as the third parameter to the *ioctl*. A value of 0 assigned to *framecount* enables hardware continuous acquisition mode until **SDV\_STOP\_CONTINUOUS** is called. Provide an unsigned integer as an argument.

Contact Engineering Design Team for help with the interface specific to your camera model.

**SDV\_STOP\_CONTINUOUS**

Disables hardware continuous acquisition mode, described above. No argument is required.

**EDTS\_APPBUFS\_COMPLETED**

Use with continuous ring-buffer mode (**EDTS\_NBUFIO**) and lockstep mode (**EDTS\_LOCKSTEP**). Instructs the driver to process the number of buffers as specified in the third argument to this *ioctl*, then wait until this *ioctl* is called again. Provide an unsigned integer as an argument.

## Asynchronous Input and Output

If the SDV-RCI is performing constant sequential read or write operations, it cannot achieve the fastest possible throughput because of application scheduling latencies and operating system overhead. However, you can queue buffers asynchronously to improve your application performance. To help you do so, the *sdvr* driver supports a continuous ring-buffering mode.

A ring buffer is an ordered collection of buffers upon which DMA can be performed in round-robin fashion. DMA starts with the first buffer; when the last buffer is completed, DMA starts over again with the first buffer. The *sdvr* can use from 1 to 40 buffers in this fashion.

**NOTE: This number can be increased if necessary. Contact Engineering Design Team, Inc. for more information.**

You can also control the rate at which the driver processes buffers, ensuring that the driver does not overwrite data before your application has had the chance to process it. Or you can instruct the driver to notify the application after a certain number of buffers have been processed, or after each buffer has been processed.

**NOTE: The following description of ring-buffering is provided primarily for informational purposes for those who want to know the details of the ring buffering implementation. The SDV library routines `sdv_image()`, `sdv_start_image()` and `sdv_wait_image()` handle all of the complex bookkeeping tasks and should be used to acquire images in your application.**

The following steps are necessary in order for an application to use ring-buffering:

1. Allocate memory for the buffers.
2. Tell the driver how many buffers the application will use, and turn on continuous ring-buffering mode, using the `ioctl EDTS_NBUFIO`. The third argument to this `ioctl` is an address whose value is the number of buffers for which you have allocated memory.
3. Pass the buffers to the driver using `aioread` or `aiowrite`, if you are using SunOS Version 4.1.x, or `thr_create` and `read` or `write`, if you are using Solaris 2.x. (See the Sun man pages for further information on these calls.)
4. Choose the approach you wish to use to ensure that your application and the `sdvrci` work together as expected. If you wish to control the rate at which the driver processes buffers, use the `ioctl EDTS_LOCKSTEP` to initiate lockstep mode, in which the driver waits for notification from the application before proceeding with DMA. The number of buffers are processed as specified in the third argument to this `ioctl`. The driver then waits until it receives notification by means of the `ioctl EDTS_APPBUFS_COMPLETED`, which instructs the driver to process the number of buffers specified in the third argument, and then wait until this `ioctl` is called again. (If the application processes data fast enough to stay ahead of the driver, the driver will not wait.)

If you wish the driver to notify the application after a certain number of buffers have been processed, use the `ioctl EDTS_WAKEUP_DONECOUNT`, which does not return until as many buffers have completed as you specified in the third argument to the `ioctl`. This causes the application to block until the driver has reached the specified number of DMA operations. (If the specified number of buffers have already been processed it returns immediately.) The `ioctl EDTG_DONECOUNT` returns the count of the last buffer completed by the driver. The count starts at 0 and increments each time a read request is satisfied. (The driver counter wraps at  $2^{32}$ , and the driver is implemented to handle this behavior correctly. Therefore let the count in your application wrap as well.)

You can also instruct the driver to notify the application after each buffer has been processed, using the `ioctl EDTS_EODMA_SIG`.

Examples of the first two approaches can be found in the example program `semtest.c`. The source code is shipped with the SDV-RCI; feel free to examine it. Then invoke the program and examine its output; it is explained below.

We also recommend that you examine the example program `speedtest.c`, a simpler program that applies ring-buffering to the specific task of measuring SDV-RCI throughput.

## **semtest**

Creates a ring buffer and reads data from an external data source into the buffers.

### **Usage**

```
semtest [-u device] [-b bufs] [-n notify] [-l ] [-w] size
```

**Arguments**

|                        |   |
|------------------------|---|
| <code>-u device</code> | Specifies the SDV-RCI device number (assigned during installation). The default is 0.   |
| <code>-b bufs</code>   | Sets the number of buffers in the ring. Legal values are 1–40. The default is 1.  |
| <code>-n notify</code> | Sets the number of buffers after which to notify the application. The application is notified after the specified number of buffers have been processed. Legal values are 1–4 billion. The default is 1.  |
| <code>-l</code>        | Specifies that the driver is to be locked to the speed of the application. The default is off.  |
| <code>-w</code>        | Specifies that the SDV-RCI is to wait for the user to press <Return> after each notification. The default is off.   |
| <code>size</code>      | Sets the amount of data contained in each buffer. Unmodified numbers are interpreted as bytes. Legal modifiers are W, w, K, k, M, or m. W (upper- or lowercase) indicates that the preceding number specifies the number of two-byte words. K (upper- or lowercase) indicates that the preceding number specifies the number of kilobytes. M (upper- or lowercase) indicates that the preceding number specifies the number of megabytes. |

**Notes**

Invoking `semtest` without any arguments produces a statement explaining its usage. To run this program, you must supply at least the size of the buffer.

**Output**

Executing `semtest` produces two columns of numbers. The left column shows the number of DMA requests from the application. The right column shows the number of DMA operations that the driver has performed.

**Examples**

Execute `semtest` with the following options to view a representative sample of behavior:

`semtest 1m` Creates a ring buffer consisting of a single buffer holding 1 MB and notifies the application after each buffer has been read, using the default driver rate (as fast as possible).

`semtest -l 1m` Creates a ring buffer consisting of a single buffer holding 1 MB and notifies the application after each buffer has been read, locking the driver rate to the speed of the application.

`semtest -b 34 -n 17 128K` Creates a ring buffer consisting of 34 buffers holding 128 KB each and notifies the application after 17 have been read, using the default driver rate (as fast as possible). This has the effect of allowing the driver to perform DMA to half the buffers while the application processes the other half; the driver and the application can then swap halves.

`semtest -u 1 -b 34 -n 17 -w 10K` Creates a ring buffer consisting of 34 buffers holding 10 kilobytes each and notifies the application after 17 have been read, after which it waits for you to press <Return> before resuming execution. The driver will process buffers faster than the application can keep up with it, which will be reflected in the output.

```
semtest -l -w 256W
```

Creates a ring buffer consisting of a single buffer that holds 256 two-byte words (512 bytes), locking the driver rate to the speed of the application, and waits for you to press <Return> before resuming execution.