HD I/O Board
Owner’s Guide

Document Number 007-3968-003

CONTRIBUTORS
Written by Carolyn Curtis and Alan Stein
Illustrated by Dan Young, Cheri Brown, Dany Galgani, and Carolyn Curtis
Production by Glen Traefald
Engineering contributions by Michael Poimboeuf, Bob Williams, Craig Sayers, Jeff Hane, Greg Sadowski, Ed Miszkiewicz, Kirk Knapp

Cover design by Sarah Bolles, Sarah Bolles Design, and Dany Galgani, SGI Technical Publications

REGULATORY INFORMATION

FCC Warning
This equipment has been tested and found compliant with the limits for a Class A digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Attention
This product requires the use of external shielded cables in order to maintain compliance pursuant to Part 15 of the FCC Rules.

European Union Statement
This device complies with the European Directives listed on the “Declaration of Conformity” which is included with each product. The CE mark insignia displayed on the device is an indication of conformity to the aforementioned European requirements.

International Special Committee on Radio Interference (CISPR)
This equipment has been tested to and is in compliance with the Class A limits per CISPR publication 22, Limits and Methods of Measurement of Radio Interference Characteristics of Information Technology Equipment; Germany's BZT Class A limits for Information Technology Equipment; and Japan's VCCI Class 1 limits.

Canadian Department of Communications Statement
This digital apparatus does not exceed the Class A limits for radio noise emissions from digital apparatus as set out in the Radio Interference Regulations of the Canadian Department of Communications.

Attention
Cet appareil numérique n'émet pas de perturbations radioélectriques dépassant les normes applicables aux appareils numériques de Classe A préscrites dans le Règlement sur les interferences radioélectriques etabli par le Ministère des Communications du Canada.

VCCI Class 1 Statement

この装置は、情報処理装置等電波障害自主規制協議会（VCCI）の基準に基づくクラス A 情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。

Class A Warning for Taiwan

警告使用者：

這是甲類的資訊產品，在居住的環境中使用時，可能會造成射頻干擾，在這種情況下，使用者會被要求採取某些適當的對策．

Record of Revision

Version	Description
003	August 2000 Covers release 1.2 of the board, which supports additional video formats

Contents

Figures

List of Tables

About This Guide

The SGI HD I/O Board allows the SGI 2000 and 3000 series workstations and servers and the Origin200 GIGAchannel server to generate and receive uncompressed high-definition television (HDTV) signals in real time. The board is also supported in certain other SGI workstations and servers.

Note: This board requires IRIX 6.5.4 or later; earlier versions of IRIX do not recognize the board.

You control board features using the Video Library (VL). For a description of VL device-independent calls and controls, see the *Digital Media Programming Guide* (007-1799-060 or later; online only).

This release, 1.2, adds support for additional video formats.

Audience

This guide is provided for the sophisticated video user in a professional or research environment. You should be familiar with video standards, the operation of the Silicon Graphics workstation or server, and the VL related information in the *Digital Media Programming Guide*.

Structure of This Guide

This guide includes the following chapters and appendices:

- Chapter 1, “Features and Capabilities,” outlines the main components of the HD I/O Board.
- Chapter 2, “Programming the HD I/O Board,” explains how to use the VL to perform typical tasks.
- Chapter 3, “Synchronizing Data Streams and Signals,” describes how to use unadjusted system time (UST) and media stream count (MSC).
- Appendix A, “HD I/O Board Specifications,” summarizes the board’s technical specifications.
- Appendix B, “Setting Up the HD I/O Board for Your Video Hardware,” explains how to connect video equipment to HD I/O Board connectors and how to use the control panel *vcp* to configure the board.
- Appendix C, “Pixel Packings and Color Spaces,” describes all the packing formats used by the HD I/O Board.
- Appendix D, “Programming Methods for Real-Time Digital Media Recording and Playback,” explains programming concepts, such as real-time disk I/O, and provides examples.
- Appendix E, “Installing HD I/O Software on a New Disk,” provides software installation instructions (if required for a new disk).

An index completes this guide.

Related Publications

Besides this guide, *Digital Media Connections* (007-3525-003 or later) is shipped with the HD I/O Board. The *Digital Media Programming Guide* is provided with the IRIX digital media development environment software (`dmedia_dev`); the online version of this manual is included with IRIX 6.5.4 and later.

It is also a good idea to have your system owner's guide available. If you do not have these guides handy, the information is available online in the following locations:

- In the IRIS InSight Library, choose Toolchest > Help > Online Books > SGI EndUser or SGI Admin, and select the applicable guide.
- In the Technical Publications Library on the Web. Enter the following URL: http://techpubs.sgi.com

  Once you are in the library, choose Catalogs > Hardware Catalog > and look under the Owner's Guides for the applicable documentation. For software guides, look on the bookshelf for the applicable IRIX version.

Conventions Used in This Guide

The following conventions are used throughout this document:

Convention	Meaning
`Command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
variable	Italic typeface denotes filenames, variable entries, and words or concepts being defined.
`user input`	Fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names.
Helvetica Bold	This font is used for labels on hardware, such as for ports and LEDs on the I/O panel.

In each chapter or appendix, when the *Digital Media Programming Guide* is referenced, it appears with its full title at the first occurrence and thereafter as the DMPG.

Product Support

SGI provides a comprehensive product support and maintenance program for its products. If you are in North America and would like support for your SGI-supported products, contact the Technical Assistance Center at 1-800-800-4SGI or your authorized service provider. If you are outside North America, contact the SGI subsidiary or authorized distributor in your country.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number can be found on the back cover.)

You can contact us in any of the following ways:

- Send e-mail to the following address:

  `techpubs@sgi.com`

- Use the Feedback option on the Technical Publications Library World Wide Web page:

  http://techpubs.sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  Technical Publications
  SGI
  1600 Amphitheatre Pkwy., M/S 535
  Mountain View, California 94043-1351

- Send a fax to the attention of “Technical Publications” at +1 650 932 0801.

  We value your comments and will respond to them promptly.

Features and Capabilities

The SGI HD I/O Board is an XIO card for the SGI 2000 and 3000 series workstations and servers and the Origin200 GIGAchannel server. This board interfaces with American Television Standards Committee high-definition television video formats. It supports real-time input and output of high-definition video at live frame rates, in SMPTE 274M, SMPTE 296M, and SMPTE 260M video formats, using component parallel digital interfaces and high-definition dual-link 4:4:4:4.

This chapter consists of the following sections:

HD I/O Features

The HD I/O Board includes the following features:

Supported Video Formats

The HD I/O Board supports video formats defined by the Advanced Television Standards Committee (ATSC), as well as several formats defined for high-definition digital motion pictures and post production. These formats can have pixel clocks of up to 74.25 MHz. These formats include support for:

- 16 x 9 aspect ratio with 1920, 1280 and 720 active pixels in line, depending on video format
- 24/1.001, 24, 25, 30/1.001, 50, and 60/1.001 Hz vertical rates

Examples of supported formats are SMPTE 274M (interlaced and progressive), SMPTE 296 (progressive), SMPTE 295M (interlaced), SMPTE 260M, and SMPTE 293M.

Table 1-1 summarizes HD I/O formats supported for this release.

Table 1-1 Supported Formats

Description	Frame Rate	Timing	Notation
1920x1080 interlaced	30/1.001 Hz	1920x1080_5994i	1920x1080i@59.94Hz 4:2:2
1280x720 progressive	60/1.001 Hz	1280x720_5994p	1280x720p@59.94Hz 4:2:2
1920x1080 interlaced	25 Hz	1920x1080_50i	1920x1080i@50Hz
1920x1080 progressive	24 Hz	1920x1080_24p	1920x1080p@24Hz
1920x1080 progressive	24/1.001 Hz	1920x1080_23.98p	1920x1080p@24/1.001Hz
1920x1080 progressive	25 Hz	1920x1080_25p	1920x1080p@25Hz
1920x1080 progressive, segmented frame	25 Hz	1920x1080_25PsF	1920x1080PsF@25Hz
1920x1080 progressive, segmented frame	24 Hz	1920x1080_24PsF	1920x1080PsF@24Hz
1920x1080 progressive, segmented frame	24/1.001 Hz	1920x1080_2398PsF	1920x1080PsF@24/1.001Hz
1920x1035 interlaced	60/1.001 Hz	920x1035_59.4i	1920x1035i@60/1.001Hz

In segmented progressive frame formats, the progressive frame is transmitted as two fields that are from the same progressive scan, whereas in interlaced formats the two fields are temporally displaced.

All formats are 8-bit or 10-bit.

These formats are used for content creation and telecine output, and support serial-parallel conversion.

Genlock and Timing Features

The board supports many genlock source/timing combinations, including analog reference (bilevel and trilevel sync) combinations and digital input genlock combinations. Genlock and timing features are all designed to SMPTE 274M timing specifications:

- Genlock to external analog and internal digital sync reference inputs:
  - Trilevel analog sync at high-definition (HD) rates
  - Bilevel analog sync at standard-definition (SD) rates (with and without output format timing conversion of SD to HD rates)
- Genlock to digital input: locking digital output to jitter-attenuated digital input clock
- Stand-alone free-run timing:

  Uncalibrated free-run mode provides 25 ppm-accurate free-run frequency. After calibration, free-run mode provides higher than 10 ppm-accurate free-run frequency, which can be recalibrated for local references and conditions to as high as 1 ppm accuracy. For information on calibration, see the HD I/O release notes.

For output timings for various inputs, see Table 2-3 in Chapter 2.

Other Features

Other features of the HD I/O Board are:

- Bit-parallel ECL differential interface (special connectors, cable, adapters)
- Automatic adjustment of input phase between Link A and Link B (up to +/- eight input-clock differential/dynamic phase tracking)
- YCrCb 10 or 8 bits per component (4:2:2 or 4:4:4 sampling rates)
- Alpha channel support
- Video interface support for RGB 10 or 8 bits
- Support for up to 48 bits/pixel RGB in memory

- Real-time transparent color-space conversion and key scaling
- User-programmable horizontal and vertical phase adjustment of the output video
- UST support on input and output
- Gamma correction support through user-downloadable 13-bit-wide lookup tables
- 3/2 pulldown mode on output
- Board internal loopback mode

Figure 1-1 shows the board.

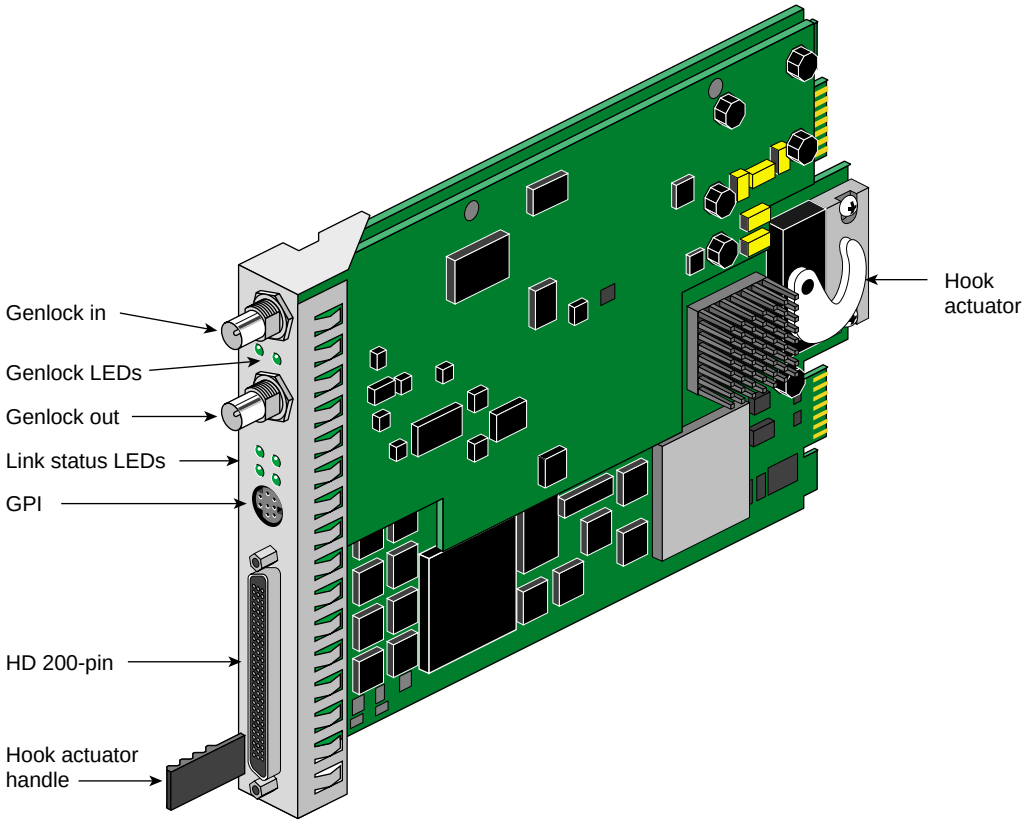


Figure 1-1 HD I/O Board

Figure 1-2 is a simplified top-level diagram of the HD I/O Board.

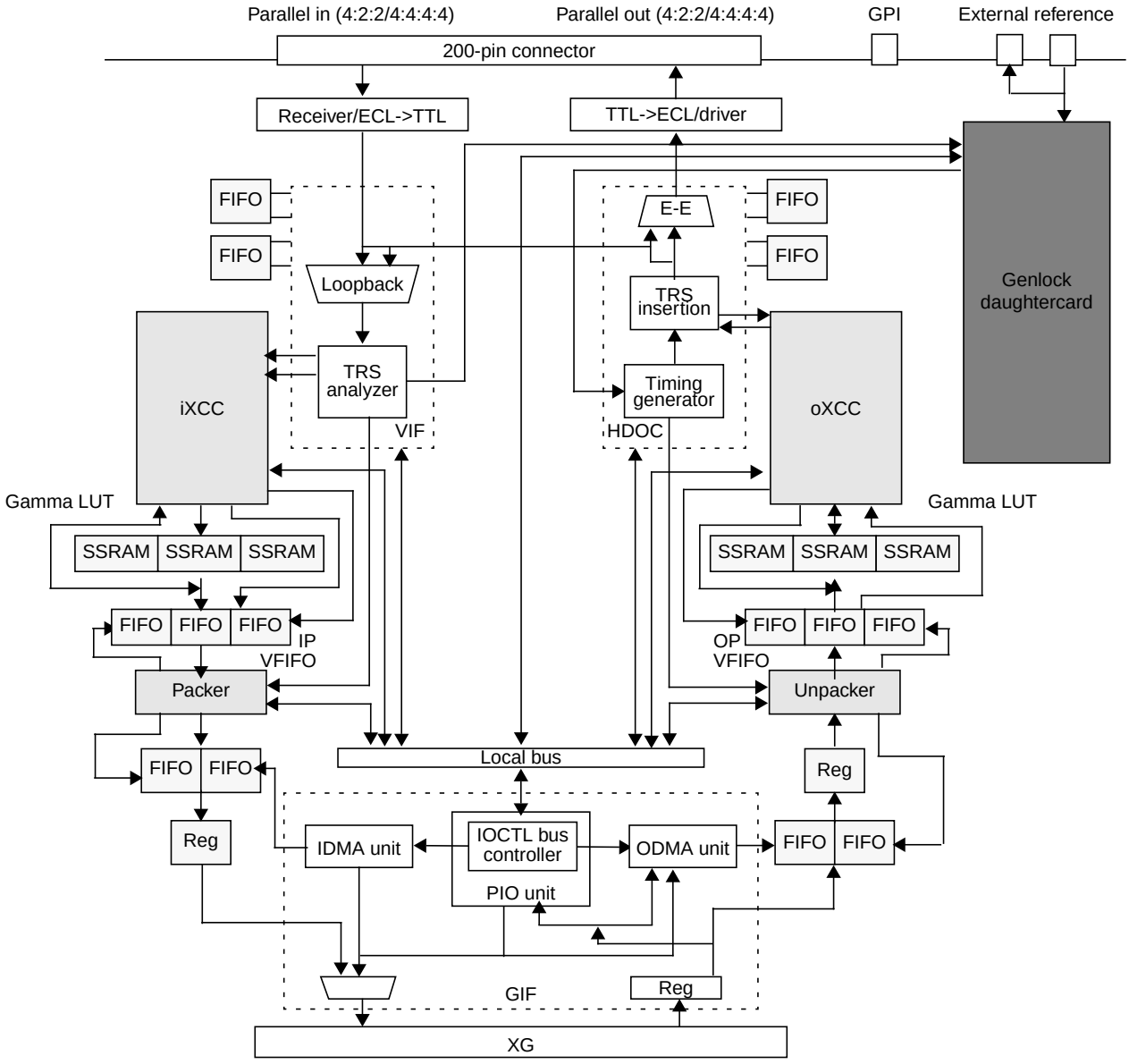


Figure 1-2 HD I/O Board Diagram, Simplified

HD I/O Panel and Cable

Figure 1-3 shows connectors on the HD I/O front panel.

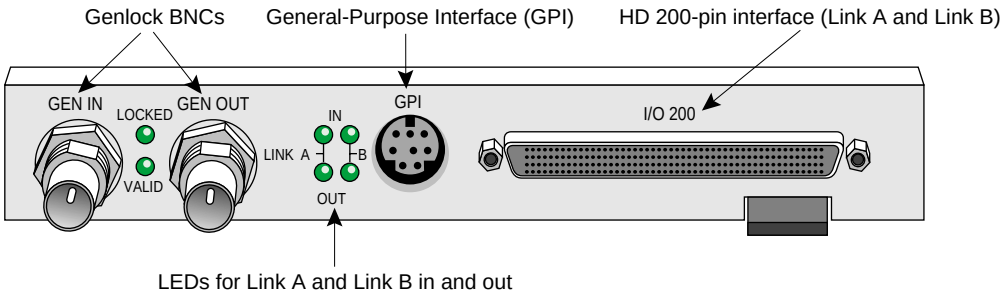


Figure 1-3 HD I/O Board Connectors

Figure 1-3 shows the two multiheaded cables included with the board; each has four 50-pin connectors for link A input, link B input, link A output, and link B output. The 50-pin connectors differ for each type of cable, following the Panasonic and Philips 50-pin video equipment interface standard.

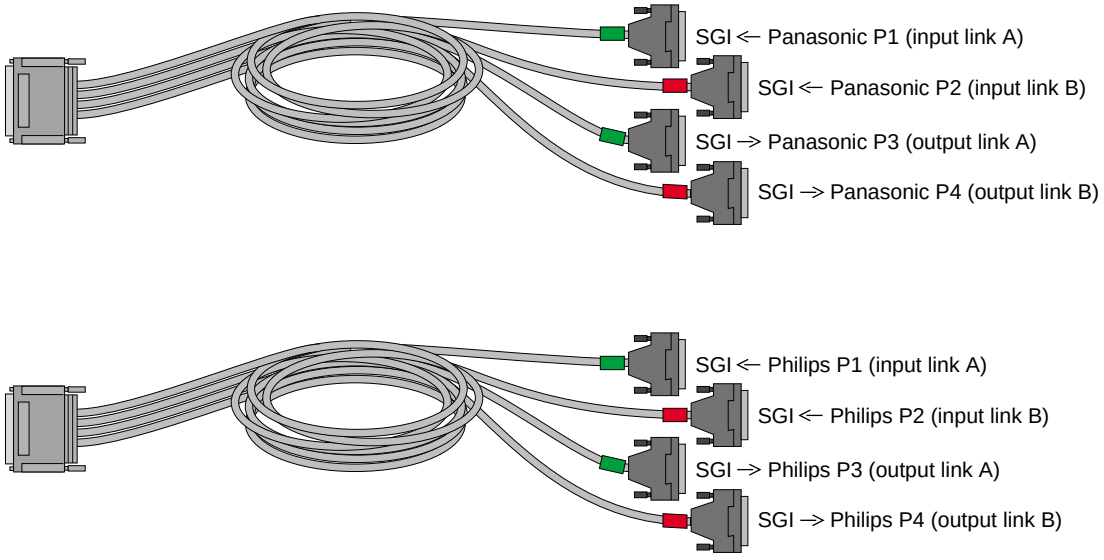


Figure 1-4 HD I/O Cables

Use the cable 50-pin connectors as indicated in Table 1-2.

Table 1-2 50-Pin Cable Connectors

Label Color	Label: Panasonic Cable	Label: Philips Cable	Use
Green	**SGI <--- PANASONIC P1**	**SGI <--- PHILIPS P1**	Input Link A
Red	**SGI <--- PANASONIC P2**	**SGI <--- PHILIPS P2**	Input Link B
Green	**SGI ---> PANASONIC P3**	**SGI ---> PHILIPS P3**	Output Link A
Red	**SGI ---> PANASONIC P4**	**SGI ---> PHILIPS P4**	Output Link B

For pinouts, see “Cable Connectors” on page 48 in Appendix A.

You can use the four 50-pin connectors for 4:4:4:4 in dual-link mode, or 4:2:2 in single-link mode where alpha is ignored:

- In YCrCb 4:4:4:4 mode, Link A carries Y:Cr0:Cb0 (even Cr/Cb samples), and Link B carries A:Cr1:Cb1 (odd CrCb samples).
- In 4:2:2 mode (single-link), Link A carries Y plus Cr and Cb; Link B is unused.
- In RGBA 4:4:4:4 mode, Link A carries G:R0:B0 (even RB samples), and Link B carries A:R1:B1 (odd RB samples).

The selected video format determines Link A and Link B usage. For more information, refer to the following standards, which contain provisions for video signals:

- SMPTE 240M

  SMPTE 240M corresponds to the early 1035i HDTV format. The HD I/O Board supports this model for compatibility with some equipment still using this standard. (This standard also defines color spaces, which must be set with a control, as explained in “VL_COLORSPACE” on page 26 in Chapter 2.)
- SMPTE 274M (subset, up to 74.25 MHz)
- SMPTE 296M (progressive)
- Recommendation 709 (ITU-R BT.709-2)

Programming the HD I/O Board

The HD I/O Board supports the Video Library (VL). This API is described in the *Digital Media Programming Guide* (007-1799-060 or later; hereafter referred to as the DMPG).

This chapter consists of the following sections:

VL Basics for the HD I/O Board

To build programs that run under VL, you must

- install the `dmedia_dev` and `dmedia_eoe` options
- link with `libvl`
- include `dmedia/vl.h` and `dmedia/vl_xthd.h` for device-dependent functionality

The client library for VL is `/usr/lib32/libvl.so`. The header files for the VL are in `/usr/include/dmedia`; the main file is `vl.h`. This file contains the main definition for the VL API and has controls that are shared by all hardware. You can find several useful digital media programming examples in `/usr/share/src/dmedia/video/XTHD.`

Note: When building a VL-based program, you must add `-lvl` to the linking command.

For more information on the Video Library and using the API, see the latest version of the DMPG.

The following topics are covered in this section:

VL Concepts

The Video Library defines a basic set of primitives and mechanisms that specify interconnections and controls for the desired setup. The following concepts are central to VL:

- *path*: an abstraction that describes the way data moves around
- *node:* an endpoint of a path

  The basic nodes are a *source* (such as a VTR) and a *drain* (such as memory). The example in Figure 2-1 shows one source node and one drain node, which illustrates the simplest VL path.

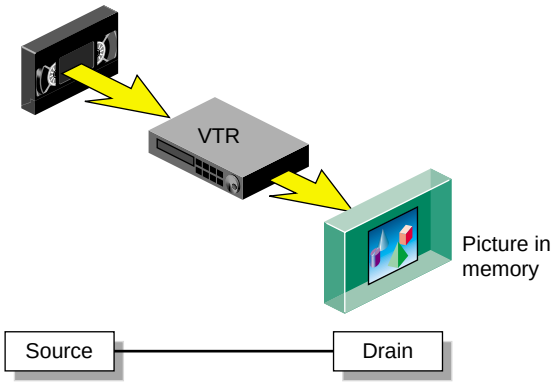


Figure 2-1 Simple VL Path

The HD I/O Board has a video source node (the video input), a video drain node (the video output), a memory source node (for output from application), and a memory drain node (for input to application). For transfers, each path must contain exactly one video node and one memory node. HD I/O nodes are further discussed in "VL Nodes for the HD I/O Board" on page 12.

VL Syntax Elements

VL syntax elements are as follows:

- VL types and constants begin with uppercase VL; for example, VLServer
- VL functions begin with lowercase vl; for example, **vlOpenVideo()**

VL Object Classes

The VL recognizes these classes of objects:

- *devices*, each including sets of nodes
- *nodes*, which are sources, drains, and internal nodes (as discussed in the preceding section)
- *paths*, connecting sources and drains (as discussed in the preceding section)
- *buffers*, for sending and receiving field/frame data to and from host memory

The HD I/O Board requires the use of DMbuffers (digital media buffers). DMbuffers, an abstraction of main memory, allow efficient and API-independent interchange of data between the different digital media libraries. For example, video fields can be captured into DMbuffers via VL and then displayed in graphics using OpenGL. They can also be passed between two processes without the data having to be copied explicitly. For details, refer to Chapter 5, "Digital Media Buffers," in the DMPG.

- *events*, for monitoring video I/O status
- *controls*, or parameters that modify how data flows through nodes; for example:
  - video device parameters, such as sync source
  - video data parameters such as packing, size, and color space

Following are the two types of VL controls:

- *device-global* or *device-independent* (prefix VL_), which can be used by several Silicon Graphics video products

  For details of the device-independent controls, refer to the DMPG.
- *device-dependent* (prefix VL_XTHD_), specific to a particular video device, in this case, the HD I/O Board

Using the HD I/O Board with both types of VL controls is explained later in this chapter.

VL Nodes for the HD I/O Board

Use **vlGetNode()** to specify nodes. This call returns the node's handle, which is used when setting controls or setting up paths. Its function prototype is:

```
VLNode vlGetNode(VLServer svr, int type, int kind, int number)
```

In this prototype, variables are as follows:

svr Names the server (as returned by **vlOpenVideo()**).

type Specifies the type of node:

- VL_SRC: source, such as a digital tapedeck connected to an input port

Note: The HD I/O Board has only one input.

- VL_DRN: drain, such as system memory
- VL_DEVICE: global control, such as a default source; Table 2-1 summarizes the values for this type

Note: If you are using VL_DEVICE, the *VLNode* should be set to 0.

kind Specifies the kind of node:

- VL_VIDEO: connection to a video device equipment; for example, a video tapedeck or camera
- VL_MEM: workstation memory

number Number of the node in cases of two or more identical nodes, such as two video source nodes. The default value for all *kind*s is 0.

VL_ANY can also be used as a value for *number* to reference the first available node of the specified *type* and *kind*.

In general, a path for the HD I/O Board has a memory node and a video node. The following fragment creates a digital video input source node and a memory drain node, and creates the path:

```
VLServer svr;
VLPath path;
VLNode src;
VLNode drn;
                    /*Set up video source node */
src = vlGetNode(svr, VL_SRC, VL_VIDEO, VL_ANY);
                    /*Set up memory drain node */
drn = vlGetNode(svr, VL_DRN, VL_MEM, VL_ANY);
                    /* Create source-to-drain path */
if((path = vlCreatePath(svr, VL_ANY, src, drn)) < 0){
     fprintf(stderr,"%s\n",vlStrError(vlGetErrno()));
     exit(1);
}
                    /* Set up path with shared src and drn node */
vlSetupPaths(svr, (VLPathList)&path, 1, VL_SHARE, VL_SHARE);
```

After you specify the nodes, use **vlSetControl()** to specify the parameters as follows:

- For memory nodes, you must set packing, color space, size, and capture type
- For video nodes, if desired, set video timing, format (for example digital component), and color space; otherwise, the default values displayed in the video control panel (*vcp*) are applied

Controls for each node are defined in "HD I/O Controls" on page 17, and are summarized in Table 2-2. You can set controls in any order.

VL Data Transfer Functions

This section summarizes VL data transfer categories, and provides basic instructions for creating an application. For the HD I/O Board, VL data transfers always involve memory (video to memory, memory to video), which requires a DMbuffer pool setup.

Follow these steps to create a VL application using the VL programming model:

1. Open a connection to the video daemon (**vlOpenVideo()**).
2. Specify nodes on the data path (**vlGetNode()**).
3. Create the path (**vlCreatePath()**).

   Optional step: add more nodes to a path (**vlAddNode()**).
4. Set up the hardware for the path (**vlSetupPaths()**).
5. Specify path-related events to be captured (**vlSelectEvents()**, **vlAddCallback()**).
6. Set input and output parameters (controls) for the nodes on the path (**vlSetControl()**).

   Video format and timing set in the *vcp* are persistent and default to reasonable values.
7. Create a dmBuffer pool to hold data for memory transfers (**vlDMGetParams()**, **dmBufferSetPoolDefaults()**, **dmBufferCreatePool()**, **vlGetTransferSize()**).
8. Register the buffer (**vlDMPoolRegister()**,**vlDMPoolDeregister()**).
9. Start the data transfer (**vlBeginTransfer()**).
10. Get the data (**vlDMBufferGetValid()**, **vlDMBufferPutValid()**, **dmBufferAllocate()**, **dmBufferAllocateSize()**, **dmBufferGetPoolState()**, **dmBufferGetPoolFD()**, **dmBufferSetPoolSelectSize()**, **dmBufferMapData()**, **dmBufferFree()**) to manipulate frame data.

11. Handle data stream events (**vlSelectEvents()**, **vlNextEvent(), vlPending()**).
12. Clean up (**vlEndTransfer()**, **vlDMPoolDeregister()**, **vlDestroyPath()**, **vlCloseVideo()).**

Note: Error handling (**vlPerror()**) is accomplished throughout.

HD I/O Data Flow

The diagram in Figure 2-2 shows the input data flow.

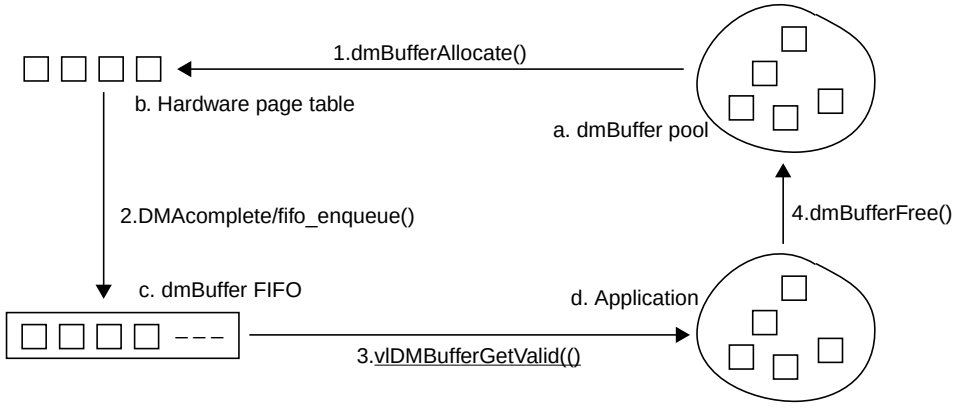


Figure 2-2 Input Data Flow

The diagram in Figure 2-2 shows the output data flow.

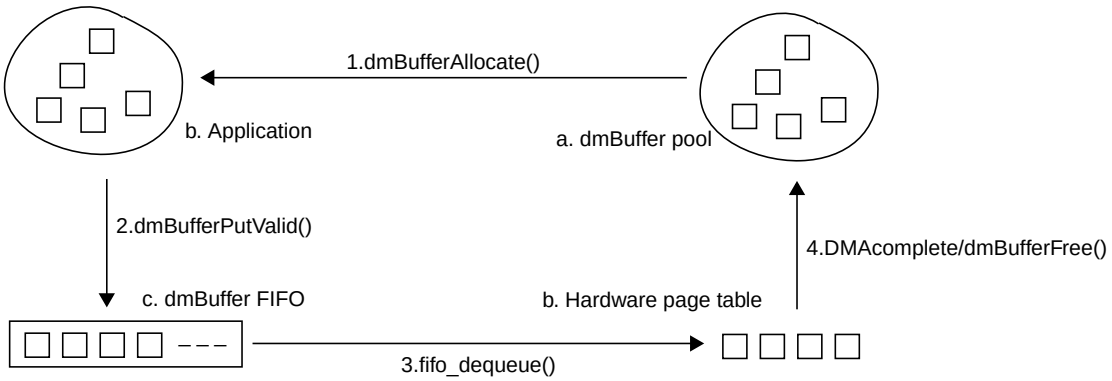


Figure 2-3 Output Data Flow

The diagram in Figure 2-4 shows the control flow.

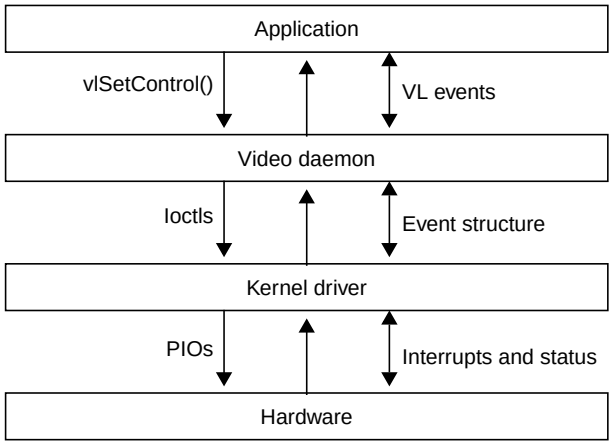


Figure 2-4 Control Flow

HD I/O Controls

This section covers the following topics:

Setting Controls

After you set up a path, and before you start a transfer, you must configure the memory and video nodes appropriately. Controls on the video node are persistent; that is, they retain their values between path destruction and creation. Memory node controls, however, are reset at each path creation. Because the video node controls are persistent, they need not be set by an application that can deal with arbitrary settings.

For a memory node, you must set the following controls because the default values are likely to be inappropriate:

- VL_PACKING
- VL_COLORSPACE
- VL_CAP_TYPE
- VL_SIZE

  The recommended way to set VL_SIZE is to query VL_SIZE on the video node (after setting VL_TIMING appropriately) and use the returned value to set VL_SIZE on the memory node.

On the video node, applications will probably need to set some controls as well. Most of them can also be set with the video control panel application (*vcp*).

To determine the available devices (that is, video options in the workstation, such as the HD I/O Board) and the nodes available on them, run *vlinfo*. To determine possible controls for each device, enter

```
vlinfo -l
```

Note: VL controls specified as true with **vlSetControl()** are executed immediately. However, they do not necessarily occur at a specific time.

To set controls for HD I/O nodes, use **vlSetControl()**. The following example sets video format and timing on a node:

```
timing.intVal = VL_TIMING_1125_1920x1080_5994i;
format.intVal = VL_FORMAT_DIGITAL_COMPONENT;

if (vlSetControl(svr, path, drn, VL_TIMING, &timing) <0)
{
     vlPerror("VlSetControl:TIMING");
     exit(1);
}
if (vlSetControl(svr, path, drn, VL_FORMAT, &format) <0)
{
     vlPerror("VlSetControl:FORMAT");
     exit(1);
 }
```

For details on **vlSetControl()** and **vlGetControl()**, see the latest version of the DMPG.

HD I/O Control Summary

Table 2-1 summarizes node controls for the HD I/O Board.

Table 2-1 HD I/O Node Controls

Control	Video Source	Memory Source	Video Drain	Memory Drain
VL_CAP_TYPE		X		X
VL_COLORSPACE	X (YCrCb and RGB_H only)	X	X YCrCb and RGB_H only)	X
VL_FIELD_DOMINANCE	X		X	
VL_FORMAT	X		X	
VL_H_PHASE			X	
VL_OFFSET	X (read only)	X	X (read only)	X
VL_PACKING		X		X
VL_SIZE	X (read only)	X	X (read only)	X
VL_SYNC			X	
VL_SYNC_SOURCE	X		X	
VL_TIMING	X		X	
VL_V_PHASE			X	
VL_XTHD_EE_MODE		X	X	
VL_XTHD_INTERFACE_PRECISION	X		X	
VL_XTHD_LOOPBACK	X			
VL_XTHD_OUTPUT_REPEAT	X			X
VL_ZOOM		X		X

Note: Except for VL_XTHD_VITC_LINE_OFFSET, VL_H_PHASE, and VL__VPHASE, none of the controls can be changed during a transfer.

Table 2-2 summarizes the values and uses of controls for the HD I/O Board.

Table 2-2 Controls for the HD I/O Board

Control	Values or Range	Use
VL_CAP_TYPE	VL_CAPTURE_INTERLEAVED VL_CAPTURE_NONINTERLEAVED VL_CAPTURE_FIELDS	Selects type of frame(s) or field(s) to capture. See "VL_CAP_TYPE" on page 31 in this chapter.
VL_COLORSPACE	VL_COLORSPACE_REC601_YCRCB VL_COLORSPACE_REC601_YUV VL_COLORSPACE_REC601_RGB_H VL_COLORSPACE_REC601_RGB_F VL_COLORSPACE_240M_YCRCB VL_COLORSPACE_240M_YUV VL_COLORSPACE_240M_RGB_H VL_COLORSPACE_240M_RGB_F VL_COLORSPACE_REC709_YCRCB VL_COLORSPACE_REC709_YUV VL_COLORSPACE_REC709_RGB_H VL_COLORSPACE_REC709_RGB_F	Specifies color space of video data in memory or for input/output. See "VL_COLORSPACE" on page 26 in this chapter.
VL_FIELD_DOMINANCE	VL_F1_IS_DOMINANT VL_F2_IS_DOMINANT Note: Output frames are deinterlaced differently depending on the choice of output field dominance. Deinterlacing is specified in the application.	Identifies frame boundaries in a field sequence (interlaced formats); ignored by progressive timings. See "Field Dominance" on page 33 in this chapter.
VL_FORMAT	VL_FORMAT_DIGITAL_COMPONENT_DUAL VL_FORMAT_DIGITAL_COMPONENT	Specifies sampling format of data in or out: Standard YCrCb color space sampled at 4:4:4:4 or 4:4:4, depending on packing. 4:2:2:4 or 4:2:2, depending on packing. See "VL_FORMAT" on page 25 in this chapter.
VL_H_PHASE	[-1000,1000] pixels (increment of 1)	Sets horizontal phase.
VL_OFFSET	Fixed at (0,0).	Sets the position within the video raster to (0,0).
VL_PACKING	See Appendix C, "" for values.	Sets packing format for memory source or drain node. See "VL_PACKING" on page 25 and Appendix C.

Table 2-2 (continued) Controls for the HD I/O Board

Control	Values or Range	Use
VL_SIZE	Raster size.	Memory: Sets size of the desired region of the video raster, which must be the same as the default active region. Video: Returns size of video raster (frame); dependent on timing.
VL_SYNC	VL_SYNC_INTERNAL VL_SYNC_GENLOCK	Sets sync mode for video output; see "VL_SYNC" on page 24.
VL_SYNC_SOURCE	VL_SYNC_HOUSE VL_SYNC_DIGITAL_INPUT_LINK_A VL_SYNC_DIGITAL_INPUT_LINK_B VL_XTHD_SYNC_DIGITAL_INPUT_LINK_A VL_XTHD_SYNC_DIGITAL_INPUT_LINK_B	Selects the genlock source if VL_SYNC_GENLOCK is used. VL_SYNC_HOUSE is analog reference. VL_XTHD_SYNC_DIGITAL_INPUT _LINK_[A | B] on video source node only.
VL_TIMING	VL_TIMING_1125_1920x1080_5994p VL_TIMING_1125_1920x1080_50p VL_TIMING_1125_1920x1080_5994i VL_TIMING_1125_1920x1080_50i VL_TIMING_1125_1920x1080_2997p VL_TIMING_1125_1920x1080_25p VL_TIMING_1125_1920x1080_24p VL_TIMING_1125_1920x1080_2398p VL_TIMING_1250_1920x1080_50p VL_TIMING_1250_1920x1080_50i VL_TIMING_1125_1920x1035_5994i VL_TIMING_750_1280x720_5994p VL_TIMING_525_720x483_5994p VL_TIMING_1125_1920x1080_24PsF VL_TIMING_1125_1920x1080_2398PsF VL_TIMING_1125_1920x1080_2997PsF VL_TIMING_1125_1920x1080_25PsF	Sets or gets timing; see "VL_TIMING" on page 22.
VL_V_PHASE	[-600,600] lines (increment of 1)	Sets vertical phase.
VL_XTHD_EE_MODE	VL_XTHD_EE_MODE_OFF VL_XTHD_EE_MODE_ON	Causes digital output to transmit a copy of digital input data; output must be genlocked.
VL_XTHD_INTERFACE _PRECISION	VL_XTHD_INTERFACE_PRECISION_8 VL_XTHD_INTERFACE_PRECISION_10	Specifies whether external video interface is 8 bits or 10 bits wide.

Table 2-2 (continued) Controls for the HD I/O Board

Control	Values or Range	Use
VL_XTHD_LOOPBACK	TRUE FALSE	For video source only, sets video input to the output pipe rather than the input jack (TRUE).
VL_XTHD_OUTPUT_REPEAT	VL_XTHD_OUTPUT_REPEAT_DISABLED VL_XTHD_OUTPUT_REPEAT_LAST_FIELD VL_XTHD_OUTPUT_REPEAT_LAST_FRAME	Controls whether system repeats DMbuffers output is underflowing; see "Automatically Correcting for Output Underflow" on page 35.
VL_ZOOM	1, -1	Sets position of top video line in the buffer; see "VL_ZOOM" on page 33.

VL_TIMING

The VL_TIMING control sets timing type, which expresses the timing of video presented to a source or drain.

Note: For the HD I/O Board, VL_TIMING is specified on the video node only, and not on the memory mode, as with other Silicon Graphics video options.

Table 2-3 lists output timings for various inputs.

Table 2-3 Supported Output Timings for Inputs or Reference Sources

Input/Reference Source (Sync Source)	Output Timing	VL_TIMING
525_NTSC	1920x1080i@59.94Hz	1920x1080_5994i
	1280x720p@59.94Hz	1280x720_5994p
	1920x1080p@23.98Hz	1920x1080_2398p
	1920x1080psF@23.98Hz	1920x1080_2398PsF
	1920x1035i@59.94Hz	1920x1035_5994i
625_PAL	1920x1080i@50Hz	1920x1080_50i
	1920x1080p25Hz	1920x1080_25p
	1920x1080PsF25Hz	1920x1080_25PsF

Table 2-3 (continued) Supported Output Timings for Inputs or Reference Sources

Input/Reference Source (Sync Source)	Output Timing	VL_TIMING
1920x1080i@59.94Hz (1920x1080_5994i)	1920x1080i@59.94Hz 1280x720p@59.94Hz	1920x1080_5994i 1280x720_5994p
1920x1080p@23.98Hz (1920x1080_2398p)	1920x1080p@23.98Hz 1080PsF@23.98Hz	1920x1080_2398p 1920x1080_2398PsF
1080PsF@23.98Hz (1920x1080_2398PsF)	1920x1080p@23.98Hz 1920x1080PsF@23.98Hz	1920x1080_2398p 1920x1080_2398PsF
1920x1080p24Hz (1920x1080_24p)	1920x1080p24Hz 1920x1080PsF24	1920x1080_24p 1920x1080_24PsF
1920x1080PsF24Hz (1920x1080_24PsF)	1920x1080p24Hz 1920x1080PsF24	1920x1080_24p 1920x1080_24PsF
1920x1080i@50Hz (1920x1080_50i)	1920x1080i@50Hz	1920x1080_50i
1920x1035i@59.94Hz 1920x1035_5994i)	1920x1035i@59.94Hz	1920x1035_5994i
1920x1080i@59.94Hz 1920x1080_5994i	1920x1035i@59.94Hz	1920x1080_5994i
1920x1080p25Hz (1920x1080_25p)	1920x1080p25Hz 1920x1080PsF25Hz	1920x1080_25p 1920x1080_25PsF
1920x1080PsF25Hz (1920x1080_25PsF)	1920x1080p25Hz 1920x1080PsF25Hz	1920x1080_25p 1920x1080_25PsF

Each value for VL_TIMING indicates the raster configuration of a particular SMPTE specification, such as SMPTE 274M-1995 system 3. The values are named according to the raster format:

- The first field is the number of total lines, such as 1125, 750, 525, or 625.
- The second field is the size of the active region, in pixels by lines.
- The third field is the vertical refresh rate and the scanning format; the scanning format is
  - i: interlaced
  - p: progressive (noninterlaced

– PsF: progressive, segmented frame

In segmented frame formats, the frame is transmitted as two fields that are of the same time instant, whereas in interlaced formats the two fields are temporally displaced.

For example, VL_TIMING_1125_1920x1080_5994i specifies 1125 total lines, an active region of 1920 pixels by 1080 lines, 59.94 fields per second, and 2:1interlacing.

Note: Although the VL defines timings with sampling rates up through 148.5 MHz, the HD I/O Board supports only those through 74.25 MHz.

VL_SYNC

The HD I/O Board has flexible sync and genlock controls. You can select sync for the outputs only; inputs sync only to the sync and clock information embedded in the digital input stream.

You can configure the video output to lock to an external sync source (genlock) or to free run (stand-alone). The VL_SYNC control allows you to select stand-alone mode or one of the genlock modes. If the setting is genlock, the following sync sources are available:

- Bilevel sync sources: external 525 NTSC, 625 PAL

  These choices select the analog composite video signal present on the connector labeled **GEN IN**. Be sure to terminate the loop-through connector (**GEN OUT**). For more information, see “Setting Up External Sync” on page 65 in Appendix B.

- Trilevel sync sources: external 1920x1080_5994i, 1920x1080_50i, 1920x1080_25p, 1920x1080_24p, 1920x1080_2398p, 1920x1035_5994i, 1920x1080_24PsF, 1920x1080_2398PsF, 1920x1080_25PsF

  These choices select the analog composite or component (Y or G) video signal present on the connector labeled **GEN IN**. Be sure to terminate the loop-through connector (**GEN OUT**). For more information, see “Setting Up External Sync” on page 65 in Appendix B.

For valid sync source/timing genlock combinations, see Table 2-3. Only these combinations are supported.

VL_FORMAT

The VL_FORMAT control is used on video nodes only. It specifies the sampling format of data on the wire, specifically, dual-link (4:4:4 or 4:4:4:4) or single-link (4:2:2).

- VL_FORMAT_DIGITAL_COMPONENT_DUAL: either 4:4:4 or 4:4:4:4, depending on the specified packing
- VL_FORMAT_DIGITAL_COMPONENT: Standard YCrCb color space sampled at 4:2:2.

The memory packing mode VL_PACKING determines how components are actually sampled.

VL_FORMAT does not imply color space, nor does it imply whether the second link is used. The second link is used whenever alpha/key channel is used, which depends on the VL_PACKING setting, or when color is sampled at 4:4:4.

VL_PACKING

A video packing describes how a video signal is stored in memory, in contrast to a video format, which describes the characteristics of the video signal. For example, the memory source node accepts packed video from a DMbuffer and outputs video in a given format. Packings are specified through the VL_PACKING control on the memory nodes.

Note: Because of HDTV's multiple color spaces, "old style" packings, such as VL_PACKING_Y_8_P, are ambiguous and therefore no longer supported for the HD I/O Board. You must specify the packing and color space explicitly.

The HD I/O Board supports common packings up through 32 bits per pixel, including 4:4:4, 4:2:2, and 4:4:4:4, at 8, 10, and 12 bits per component. Specifically, it supports packings compatible with OpenGL and IRIS GL, and those in common use from other video products. Appendix C shows the layout of each packing for the HD I/O Board. It also provides the corresponding names for these packings that are used by other libraries.

An application must set both VL_PACKING and VL_COLORSPACE. Note that changes in one parameter may change the values of other parameters set earlier; for example, clipped size may change if VL_PACKING is set after VL_SIZE. For example:

```
VLControlValue val;

val.intVal = VL_PACKING_444_8;
vlSetControl(vlSvr, path, memdrn, VL_PACKING, &val);
```

Note: At the beginning of a data transfer, it takes several seconds to activate a change in this control.

VL_COLORSPACE

The VL_COLORSPACE control specifies the color space of video data in memory or for input and output. A color space is a color component encoding format, for example, RGB and YUV. Because video equipment uses more than one color space, the HD I/O video nodes, in addition to the memory nodes, support the VL_COLORSPACE control.

Each component of an image has

- a color that it represents
- a canonical minimum value
- a canonical maximum value

Normally, a component stays within the minimum and maximum values. For example, for a luma signal such as Y, you can think of these limits as the black level and the peak white level, respectively. For an unsigned component with *n* bits, there are two possibilities for [minimum value, maximum value]:

- full range: [0, (2^{nbits})-1], which provides the maximum resolution for each component
- compressed (headroom) range, which provides numerical headroom, which is often useful when processing video images:
  - Cr and Cb: [(2^n)/16, 15*(2^n)/16]
  - Y, A, R, G, and B: [(2^n)/16, 235*(2^n)/256]

Color Spaces and Color Models

Various HDTV specifications define color models differently from those defined in Recommendation 601 (ITU-R BT.601), which is used by most standard-definition digital video equipment. For HDTV, the VL defines three color models:

- SMPTE 240M
- SMPTE 274M Recommendation 709 (ITU-R BT.709-2)
- Recommendation 601 (ITU-R BT.601)

Within each color model, four different color spaces exist:

- YCrCb: headroom range

  Headroom range means that black is at, for example, code 64 rather than 0, and white is at, for example, code 940 rather than 1023. Headroom-range color spaces can accommodate overshoot (superwhite) and undershoot (superblack) colors. Full-range color spaces clamp these out-of-range colors to black and white.
- YUV: full range
- RGB_H: headroom range
- RGB_F: full range

For memory nodes, these four color spaces are defined for each of three color models, resulting in 12 color spaces. Note that all 12 are supported on memory nodes, but only YCrCb and RGB_H color spaces are supported on video nodes.

Table 2-4 summarizes currently supported combinations of VL_FORMAT and VL_COLORSPACE; future releases will support more combinations. For detailed information on supported combinations, refer to the XTHD MAN pages and release notes regarding "Color Representations."

Table 2-4 VL_FORMAT and VL_COLORSPACE Combinations Supported

Video Node	Memory Node
4:2:2 YCrCb	4:4:4 RGB_F
4:2:2 YCrCb	4:4:4 RGB_H
4:2:2 YCrCb	4:2:2 YCrCb
4:4:4YCrCb	4:4:4 YCrCb
4:4:4 RGB_H	4:4:4 RGB_H

Color-space conversion is performed within a color model if the color spaces are different on the memory and video nodes. Conversion between the color models is not supported.

Note: Changing this control at the beginning of data transfer takes several seconds to go into effect.

Typically, two sets of colors are used together, RGB (RGBA) and YCrCb/YUV (VYUA). YCrCb (YUV), the most common representation of color from the video world, represents each color by a luma component called Y and two components of chroma, called Cr (or V), and Cb (or U). The luma component is loosely related to brightness or luminance, and the chroma components make up a quantity loosely related to hue. These components are defined in ITU-R BT.601 (also known as Rec. 601 and CCIR 601), ITU-R BT.709-2, and SMPTE 240M.

The alpha channel is not a real color. For that channel, the minimum value means completely transparent, and the maximum value means completely opaque.

For OpenGL, IRIS GL, and DM:

- the library constant indicates whether the data is RGBA or VYUA
- RGBA data is full-range by default
- VYUA data in DM can be full-range or compressed-range; you must determine this from context

For more information about color spaces, see *A Technical Introduction to Digital Video*, by Charles A. Poynton (New York: Wiley, 1996).

VL_COLORSPACE Control of Blanking

Along with memory node color space, VL_COLORSPACE determines the color-conversion matrix values. In addition, this control affects the type of blanking output by the board during horizontal and vertical blanking, and during active video area when not transferring data. On a video drain node, VL_COLORSPACE affects the type of blanking that the board outputs, in accordance with SMPTE 274M:

- YCrCb: blanking is Y = 64, Cr/Cb = 512, A = 64
- RGB_H: blanking is R = 64, G = 64, B = 64, A = 64

VL_COLORSPACE and Lookup Tables

The HD I/O Board supports lookup tables (LUTs) on input and output for gamma correction or decorrection. To successfully run an application with linear components, you can use LUTs to convert between linear and nonlinear spaces.

The HD I/O hardware includes a separate LUT for each RGB color component. Each of the three LUTs is a table of 8192 entries; each entry stores 13 bits. The application programs the entries in each table. The LUTs produce offsets, if they are required by the memory storage format.

The LUTs perform rounding as follows:

- If the LUT is not explicitly programmed by the application, the output LUT is in pass-through mode, all rounding is performed in the color-space converter and the input LUT performs both rounding and offset.
- If the LUT is programmed explicitly by the application, the application can control rounding as part of the lookup table function. The packer (hardware that reads the LUT and formats data for the host memory; see Figure 1-2 on page 5) performs a final conversion from 13-bit LUT format to host memory format.

It is also possible for an application to use the LUT to convert between video node RGB_H and memory node RGB_F. Because each component is independent of the others for this conversion, a matrix multiplication is not needed (pass through mode). The required component scaling and rounding can be placed into each LUT.

VL_COLORSPACE Examples

Figure 2-5 and Figure 2-6 show examples of color-space conversions.

In the examples, RGB are values in linear space and R'G'B' are values in nonlinear space after the optoelectric transfer function is applied as specified in ITU-R BT.709. You can use the LUTs to apply this function or its inverse to convert between RGB and R'G'B'.

The example in Figure 2-5 shows a typical video capture path. In this example, the input jack is YCrCb 4:2:2 and the desired result in system memory is RGB. First, an appropriate filter interpolates YCrCb 4:2:2 to YCrCb 4:4:4 to fill in the missing CrCb samples. Then a 3x3 matrix multiplier with appropriate offsets and coefficients obtains RGB values for each pixel. At this point, you can use the LUT option to convert gamma pre-corrected RGB values to linear RGB values. Finally, the packer swizzles the bits into the desired memory packing format and DMA places the result in system memory.

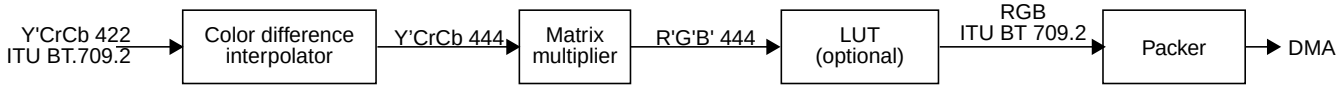


Figure 2-5 Color-Space Conversion Example 1

Figure 2-6 summarizes a complicated conversion between color spaces for 4:2:2 sampling rates and different primary colors. The example shows the conversion of material produced using SMPTE 240M colorimetry to material in SMPTE 274M format (ITU-R BT.709.2 colors).

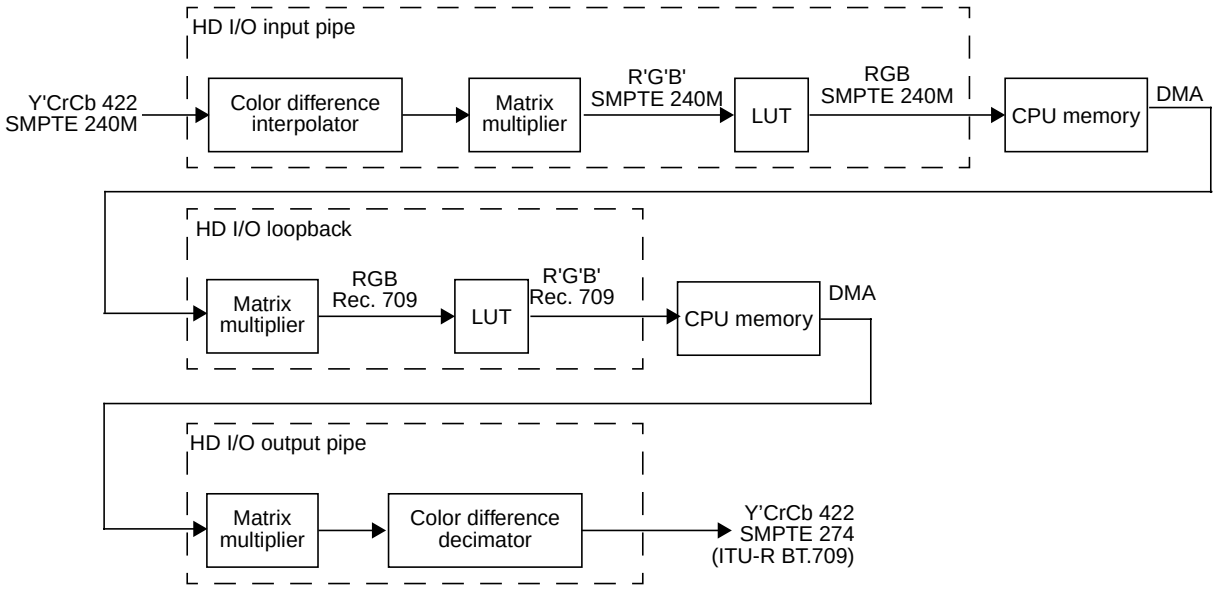


Figure 2-6 Color-Space Conversion Example 2

VL_CAP_TYPE

An application can request that the HD I/O Board capture or play back a video stream in a number of ways. For example, the application can request that each field be placed in its own buffer, or that each buffer contain an interleaved frame. This section describes the capture types supported by the HD I/O Board.

Supported capture types are as follows:

- VL_CAPTURE_NONINTERLEAVED
- VL_CAPTURE_INTERLEAVED

VL_CAPTURE_FIELDS is equivalent to VL_CAPTURE_NONINTERLEAVED. The HD I/O Board does not support VL_CAPTURE_EVEN_FIELDS and VL_CAPTURE_ODD_FIELDS.

Note: VL_SIZE refers to the size of each frame, rather than the size of the buffer in memory, so it is independent of VL_CAP_TYPE; the VL_CAP_TYPE setting does not change VL_SIZE.

VL_CAPTURE_NONINTERLEAVED

The VL_CAPTURE_NONINTERLEAVED capture type specifies that frame-size units are captured noninterleaved. Each field is placed in its own buffer, with the dominant field in the first buffer. If one of the fields of a frame is dropped, all fields are dropped. Consequently, an application is guaranteed that the field order is maintained; no special synchronization is necessary to ensure that fields from different frames are mixed.

If you specify VL_CAPTURE_NONINTERLEAVED for playback, similar guarantees apply as for capture. If one field is lost during playback, it is not possible to "take back" the field. The HD I/O Board resynchronizes on the next frame boundary, although black or "garbage" video might be present between the erring field and the frame boundary.

VL_CAPTURE_INTERLEAVED

Interleaved capture interleaves the two fields of a frame and places them in a single buffer; the order of the fields depends on the value set for VL_DOMINANCE_FIELD. The HD I/O Board guarantees that the interleaved fields are from the same frame: if one field of a frame is dropped, then both are dropped.

During playback, a frame is deinterleaved and output as two consecutive fields, with the dominant field output first. If one of the fields is lost, the HD I/O Board resynchronizes to a frame boundary before playing the next frame. During the resynchronization period, black or "garbage" data may be displayed.

VL_SIZE and VL_OFFSET

VL_SIZE refers to the size of each frame, rather than the size of the buffer in memory, so it is independent of VL_CAP_TYPE; the VL_CAP_TYPE setting does not change VL_SIZE.

For memory nodes, VL_SIZE specifies the desired region of the video raster, which must be the same as the default active region. There is no default; this control must be set. On video nodes, this control is read-only and cannot be set.

For memory nodes, VL_OFFSET is related to VL_SIZE; it specifies the origin of the video region to be captured at (0,0), the standard beginning of active video. The recommended way to set VL_SIZE for the memory node is to query it on the video node (after setting VL_TIMING appropriately) and use the returned value to set VL_SIZE on the memory node.

VL_ZOOM

The VL_ZOOM control specifies the site of the top video line in memory; it does not scale (zoom and decimate) a video image as for some other Silicon Graphics video options. The VL_ZOOM value can be 1 or -1:

- 1: Normal line order; the top video line is at the lowest address in the buffer.
- -1: Inverted line order; as in OpenGL, the top video line is at the highest address in the buffer.

Field Dominance

Field dominance identifies the frame boundaries in a field sequence; that is, it specifies which pair of fields in a field sequence constitute a frame. The control VL_FIELD_DOMINANCE allows you to specify whether an edit occurs on the nominal video field boundary (Field 1, or F1) or on the intervening field boundary (Field 2, or F2).

- F1 dominant: The edit occurs on the nominal video field boundary.
- F2 dominant: The edit occurs on the intervening field boundary.

Whether a field is Field 1 or Field 2 is determined by the setting of bit 9, the F bit, in the XYZ word of the EAV and SAV sequences. This setting is defined as follows:

- For Field 1 (also called the odd field), the F bit is 0.
- For Field 2 (also called the even field), the F bit is 1.

Note: Field dominance has no effect on progressive timings.

Figure 2-7 shows fields and frames as defined for digital 1080-line formats for the HD I/O Board.

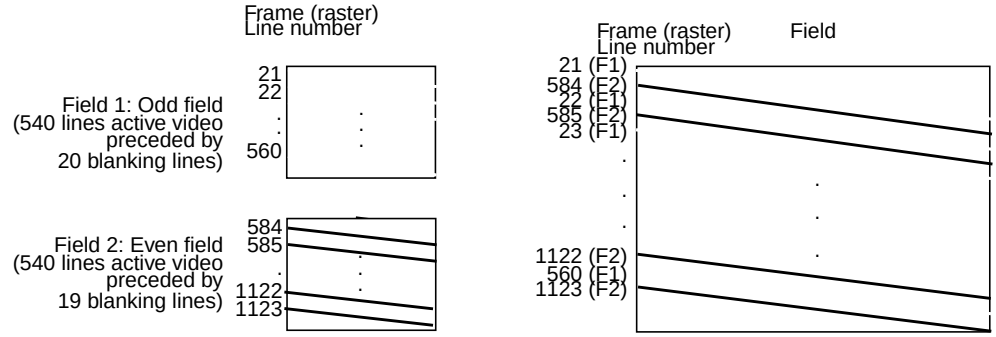


Figure 2-7 Fields and Frames for SMPTE 274M

Editing is usually on Field 1 boundaries, where Field 1 is defined as the first field in the video standard's two-field output sequence. However, some users may want to edit on F2 boundaries, which fall on the field in between the video standard's frame boundary. To do so, use this control, then program your deck to select F2 edits.

A set of frames to be output must be deinterlaced into fields differently, depending on the choice of output field dominance: for SMPTE 274M, the top line is in F1, as shown in Figure 2-7; for SMPTE 240M, the top line is in F2. For example, when F1 dominance is selected, the field with the topmost line must be the first field to be transferred; when F2 dominance is selected, the field with the topmost line must be the second field to be transferred.

Automatically Correcting for Output Underflow

If the application is not sending buffers fast enough for the receiving equipment's video frame rate, you can set VL_XTHD_OUTPUT_REPEAT to repeat DMbuffers automatically. The values for this control vary, depending on whether the transfer is progressive or interlaced.

For interlaced noninterleaved transfers, choices are as follows:

- VL_XTHD_OUTPUT_REPEAT_LAST_FIELD

  This setting repeats the last buffer twice. This setting is spacially imperfect, but does not cause flicker.

- VL_XTHD_OUTPUT_REPEAT_LAST_FRAME (the default)

  This setting repeats the last pair of buffers. This setting is spacially better than VL_XTHD_OUTPUT_REPEAT_LAST_FIELD, but causes flicker.

- VL_XTHD_OUTPUT_REPEAT_DISABLED

  This setting, which does not resend any data, is the most useful for debugging, since underflow is then quite visible on output.

For progressive or interleaved transfers, choices are as follows:

- VL_XTHD_OUTPUT_REPEAT_LAST_FRAME (the default)

  This setting repeats the last buffer.

- VL_XTHD_OUTPUT_REPEAT_DISABLED

Capturing Graphics to Video

To capture graphics to video, you can use OpenGL to read pixels into memory. However, the coordinate system differs between video and Open GL; under OpenGL, the origin is at the lower left corner and in video, origin is in the upper left corner. To adjust for this difference, set VL_ZOOM to -1; see "VL_ZOOM" on page 33.

HD I/O Events

The VL provides several ways of handling data stream events, such as completion or failure of data transfer, vertical retrace event, loss of the path to another client, lack of detectable sync, or dropped fields or frames. The method you use depends on the type of application you are writing:

- For a strictly VL application, use
  - **vlSelectEvents()** to choose the events to which you want the application to respond
  - **vlCallback()** to specify the function called when the event occurs
  - your own event loop or a main loop (**vlMainLoop()**) to dispatch the events
- For an application that also accesses another program or device driver, or if you are adding video capability to an existing X or OpenGL application, set up an event loop in the main part of the application and use the IRIX file descriptor (FD) of the event(s) you want to add.

For more information on these functions, see Chapter 4 in the DMPG.

Table 2-5 summarizes events for the HD I/O Board. For these options, this table supersedes the table of events in Chapter 14, "VL Event Handling," in the DMPG; the HD I/O Board supports only the events listed in Table 2-5.

Table 2-5 HD I/O Events

Event	Use
VLSyncLost	Sync is not detected
VLStreamStarted	Stream started delivery
VLStreamStopped	Stream stopped delivery
VLSequenceLost	A field/frame was dropped
VLControlChanged	A control on the path has changed
VLTransferComplete	A field/frame transfer has completed
VLTransferFailed	A transfer has failed and DMA is aborted
VLTransferError	A transfer error was discovered; field may be invalid

Reporting

The DMediaInfo structure reports the Unadjusted System Time (UST).

The HD I/O Board makes use of the error events noted in Chapter 4 of the DMPG, as well as VLTransferErrorEvent, which reports nonfatal video transfer errors. The VLTransferComplete and VLSequenceLost events also report the Media Stream Count (MSC) of the field or frame transferred or failed.

Examples

This section contains three examples:

Capture to Memory for Disk Recording

This example shows an application that simply captures video data and records it to disk, for later playout. The video is left in its original 4:2:2 sampling and color space. Because 10 bits are required, the R242_10 packing mode is chosen. Frames start on F1 boundaries. After creating the VL path, the application sets up the video node, mostly as a result of user input, since it depends on external equipment.

```
videoSource.VL_TIMING = VL_TIMING_1125_1920x1080_5994i
videoSource.VL_COLORSPACE = VL_COLORSPACE_REC709_YCRCB
videoSource.VL_FIELD_DOMINANCE = VL_F1_IS_DOMINANT
videoSource.VL_FORMAT = VL_FORMAT_DIGITAL_COMPONENT
videoSource.VL_XTHD_LOOPBACK = VL_XTHD_LOOPBACK_OFF
videoSource.VL_XTHD_INTERFACE_PRECISION = VL_XTHD_INTERFACE_PRECISION_10
```

Next, the application configures memory node:

```
memoryDrain.VL_SIZE = videoSource.VL_SIZE
memoryDrain.VL_COLORSPACE = videoSource.VL_COLORSPACE
memoryDrain.VL_CAP_TYPE = VL_CAPTURE_NONINTERLEAVED
memoryDrain.VL_PACKING = VL_PACKING_R242_1
```

Next, the application calls **vlGetTransferSize()** to get the required buffer size to hold each field. Using that size, it creates and registers a DMS buffer pool and starts the transfer.

Playback From Memory for Disk Playback

This example shows an application that plays back data captured in the previous example.

The application in the previous example stored various attributes along with the data, including color space, frame rate, dominance, and size. These attributes, along with some user-specified options, are used to derive the control values. After creating the VL path, the application sets up the video node:

```
videoDrain.VL_TIMING = VL_TIMING_1125_1920x1080_5994i
videoDrain.VL_COLORSPACE = VL_COLORSPACE_REC709_YCRCB
videoDrain.VL_FIELD_DOMINANCE = VL_F1_IS_DOMINANT
videoDrain.VL_FORMAT = VL_FORMAT_DIGITAL_COMPONENT
videoDrain.VL_XTHD_INTERFACE_PRECISION = VL_XTHD_INTERFACE_PRECISION_10
videoDrain.VL_SYNC = VL_SYNC_GENLOCK
videoDrain.VL_SYNC_SOURCE = VL_XTHD_SYNC_HOUSE
```

Next, the application configures the memory node:

```
memorySource.VL_SIZE = videoDrain.VL_SIZE
memorySource.VL_COLORSPACE = videoDrain.VL_COLORSPACE
memorySource.VL_CAP_TYPE = VL_CAPTURE_NONINTERLEAVED
memorySource.VL_PACKING = VL_PACKING_R242_10
```

Next, the application calls **vlGetTransferSize()** to get the required buffer size to hold each field. Using that size, it creates a DMS buffer pool, allocates, fills, and prequeues some buffers, and starts the transfer.

Capture to Memory for Graphics

In this example, the application captures video and draws it on the graphics screen using OpenGL. This example resembles the *videoin* application.

The incoming video is converted to 10-bit 4:4:4 full-range RGB in an OpenGL-compatible packing format. The 4444_10_10_10_2 packing is chosen; it is compatible with OpenGL using packed-pixel extension GL_UNSIGNED_INT_10_10_10_2_EXT pixel format. Video is interleaved in memory into frames, and is written upside down to be compatible with OpenGL's default coordinate system (glPixelZoom of (1.0, 1.0)).

First, the video node is configured:

```
videoSource.VL_TIMING = VL_TIMING_1125_1920x1080_5994i
videoSource.VL_COLORSPACE = VL_COLORSPACE_REC709_YCRCB
videoSource.VL_FIELD_DOMINANCE = VL_F1_IS_DOMINANT
videoSource.VL_FORMAT = VL_FORMAT_DIGITAL_COMPONENT
videoSource.VL_XTHD_LOOPBACK = VL_XTHD_LOOPBACK_OFF
videoSource.VL_XTHD_INTERFACE_PRECISION = VL_XTHD_INTERFACE_PRECISION_10
```

Next, the application configures memory node:

```
memoryDrain.VL_SIZE = videoSource.VL_SIZE
memoryDrain.VL_COLORSPACE = VL_COLORSPACE_REC709_RGB_F
memoryDrain.VL_CAP_TYPE = VL_CAPTURE_INTERLEAVED
memoryDrain.VL_PACKING = VL_PACKING_4444_10_10_10_2 memoryDrain.VL_ZOOM = -1/1
```

Next, the application calls **vlGetTransferSize()** to get the required buffer size to hold each field. Using that size, it creates and registers a DMS buffer pool and starts the transfer.

Synchronizing Data Streams and Signals

You can use special signals recognized or generated by the HD I/O Board—UST (unadjusted system time), MSC (media stream count)—to synchronize data streams. This chapter explains

Using UST, MSC, and Buffered Media Streams for Synchronization

Whenever a VL path is open in continuous mode, the HD I/O Board and certain other Silicon Graphics video devices continuously try to dequeue media stream samples from the path's buffer for input, or to enqueue media stream samples onto the path's buffer for output. If the buffer between the application and each device never underflows or overflows, then the application can measure and schedule the timing of input and output signals to 100% of the accuracy of the underlying device.

Occasionally, the application is held off and audio, video, or both come out late. Buffer underflow on output and overflow on input can result if the application does not keep the buffer adequately filled for the following reasons:

- The application is busy with other tasks and allows too much time to elapse between the placement of fields into the buffer.
- Processes are subject to various interruptions (10 to 80 ms for some processes) under IRIX because
  - the process for filling the buffer is running at too low a priority
  - the process cannot get a resource from IRIX that it needs, such as memory pages

To get around this problem, a patented[1] mechanism built into the VL helps keep track of data flow into and out of buffers by providing accurate timing information for each frame of video that enters or leaves the system. This mechanism, called UST/MSC, produces matched pairs of the following two numbers:

- unadjusted system time (UST), a time value that is used to state timing measurements to applications
- media stream count (MSC), a count value that identifies a particular media stream sample (a video field or frame)

The device keeps a counter called the device media stream count (device MSC), which increments by one every time the device attempts to enqueue or dequeue a media stream sample, whether or not the enqueue or dequeue attempt is successful. UST/MSC was designed to return timing information in a form that is valid whenever the buffer is not underflowing or overflowing.

The UST/MSC capability and the buffering that goes with it are appropriate for applications and devices such as movie players and digital video editing devices.

UST/MSC provides maximally accurate synchronization when scheduling cannot be guaranteed and some buffering is acceptable. Also, if scheduling becomes reliable at some later point, UST/MSC continues to function the same way with no code changes required; the buffers can be made smaller, and the result is a low-latency application with the same accurate synchronization.

Note that UST/MSC itself

- does not add any latency to an application

  The buffer adds latency: it increases the time the application would take to respond to some output event by changing its input (and vice versa). This solution to the synchronization problem is useful for applications in which a small latency can be sacrificed for more accuracy.
- does not require that an application trade off latency for accuracy
- does not require that an application use any particular size buffer
- delivers the full accuracy of the underlying hardware's timing support regardless of the scheduling characteristics of the application

---

[1] U.S. Patent 5,764,965, "Synchronization Infrastructure For Use In A Computer System"

- could be useful for graphics and texture even for low-latency applications

For the HD I/O Board, UST/MSC pairs are maintained in software and are valid only during a transfer. Make calls to **vlGetUSTMSCPair()** and **vlGetFrontierMSC()** only during a transfer; these calls block until at least one buffer is successfully transferred. Note the following:

- For interlaced timings, MSC always increments by 1 per field.
- For progressive timings, MSC always increments by 1 per frame.

The code below is a high-level algorithm to maintain synchronization of two buffered media streams that send data from memory to hardware outputs; a corresponding one is necessary for the other direction:

```
create video buffer between me and the audio output;
create audio buffer between me and the video output;
while (1)
{
   sleep until one of the buffers is getting empty;
   for (video buffer)
      {
         use UST/MSC to determine:
            “at what time (what UST) will the next video data I enqueue
            on the buffer actually go out the jack of the machine?”;
      }

   for (audio buffer)
      {
         (exact same thing as above, except for audio)
      }

   From the predicted video and audio USTs, determine
      “what is the synchronization error between the audio and video
      streams?”

   Enqueue more frames to fill up the audio and video buffer queues.
   If there is synchronization error, enqueue new frames to either skip
   frames on the stream that is behind or repeat frames on the stream
   that is ahead.
     }
}
```

The answers to the questions in the pseudocode above are obtained with three VL calls that manipulate UST and MSC and are explained in the next section.

Media Library Interfaces for UST and MSC

UST/MSC calls allow you to associate a UST with a particular piece of data that just left a buffer or is about to enter a buffer. The VL calls for determining the MSC and UST—vlGetUSTMSCPair(3dm), vlGetFrontierMSC(3dm), and vlGetUSTPerMSC(3dm)—help synchronize input and output of different data streams in cases where the application is getting data from or putting data into each device via a buffer. The application is at the "frontier" end of this buffer and the devices are at the "device" end of the buffer.

- **vlGetUSTMSCPair()** gets the timing information for each frame or field as it enters or leaves the physical jack of a device.

  This call returns an atomic UST/MSC pair for the jack (specified with the VL_NODE) for a given path that contains a VL_MEM node. The returned MSC is not guaranteed to be the one currently at the jack, nor is it even guaranteed to be the number of any media stream sample currently in the application's buffer. To relate the returned MSC to a particular item in the application's buffer, you must use **vlGetFrontierMSC()**.

- **vlGetFrontierMSC()** gets the frontier MSC associated with a particular VL_MEM node.

  The frontier MSC, at the application end of the media stream, is the MSC of the next item that the application removes from or puts into the buffer.

- **vlGetUSTPerMSC()** gets the time spacing of fields or frames in a path (the nominal average UST time elapsed between media stream samples in a given VLPath that includes a VL_MEM node).

These calls are used for extrapolating a UST/MSC pair as shown in **vlGetFrontierMSC()**. For other types of media streams, a similar mechanism extrapolates the UST/MSC pair; for example, for audio, use equivalent AL calls.

After you calculate the extrapolated UST/MSC pairs for both media streams, you can determine the synchronization error. The difference in the audio and video USTs for matching frame numbers is the amount they are out of sync. To resynchronize them, you must enqueue new frames to either skip frames on the stream that is behind or repeat frames on the stream that is ahead. The number of frames to be skipped or repeated is the difference in USTs divided by the frame rate.

To use UST/MSC, the application must have separate handles for each separate piece of data coming in or going out of some kind of buffer. The application can use these handles to specify, for example, a particular frame to output or pixels of a particular field to get.

Note: For complete details, including syntax, code examples, and caveats, see the man pages for these calls.

HD I/O Board Specifications

This appendix summarizes hardware specifications for the HD I/O Board and its cable, in these sections:

Cable Connectors

Figure A-1 shows the two multiheaded cables included with the board; each has four 50-pin connectors for link A input, link B input, link A output, and link B output. The 50-pin connectors differ for each type of cable, following the Panasonic and Philips 50-pin video equipment interface standard.

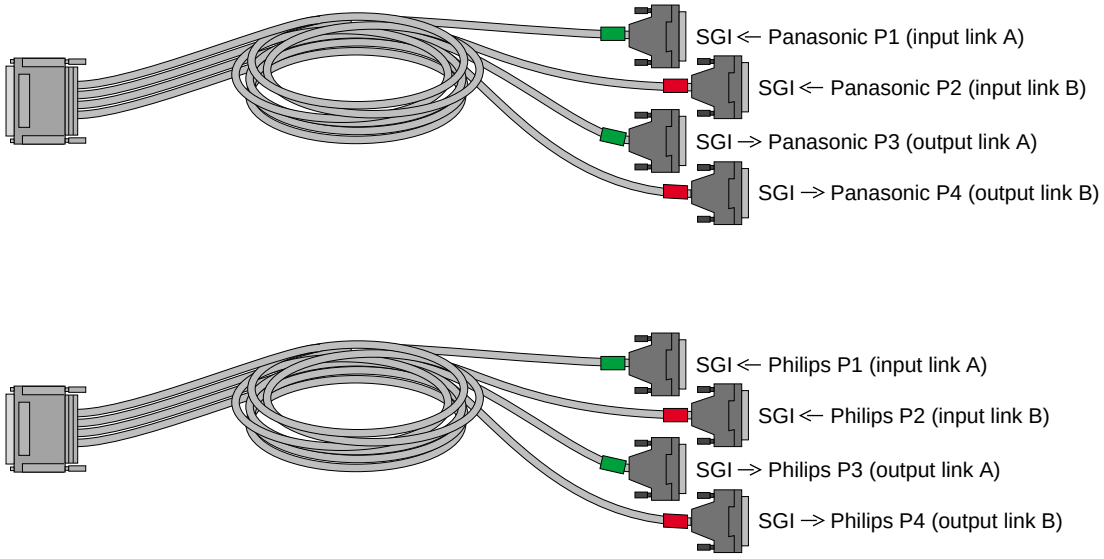


Figure A-1 HD I/O Cables

Table A-1 summarizes the Panasonic (HD-D5) pinout for the cable's 50-pin connector.

Table A-1 Panasonic 50-Pin Connector Pinout (HD-D5)

Pin	Signal	Pin	Signal	Pin	Signal
1	CLK+			34	CLK-
2	Data9+	18	Data1+	35	Data9-
3	Data8+	19	Data1-	36	Data8-
4	Data7+	20	Data0+	37	Data7-
5	Data6+	21	Data0-	38	Data6-
6	Data5+	22	GND	39	Data5-
7	Data4+	23	GND	40	Data4-
8	Data3+	24	GND	41	Data3-
9	Data2+	25	GND	42	Data2-
10	Data19+	26	Data11+	43	Data19-
11	Data18+	27	Data11-	44	Data18-
12	Data17+	28	Data10+	45	Data17-
13	Data16+	29	Data10-	46	Data16-
14	Data15+	30	GND	47	Data15-
15	Data14+	31	GND	48	Data14-
16	Data13+	32	GND	49	Data13-
17	Data12+	33	GND	50	Data12-

Table A-2 summarizes the Philips (Spirit DataCine) pinout for the cable's 50-pin connector.

Note: An early version of the Phillips 50-pin connector (SGI PN 018-0802-001) has a known wiring error. If you have this connector, contact your sales or service representative for a replacement connector (SGI PN 018-0802-002).

Table A-2 Philips 50-Pin Connector Pinout (Spirit DataCine)

Pin	Signal	Pin	Signal	Pin	Signal
1	CLK+			34	CLK-
2	Data9+	18	GND	35	Data9-
3	Data8+	19	GND	36	Data8-
4	Data7+	20	Data1+	37	Data7-
5	Data6+	21	Data1-	38	Data6-
6	Data5+	22	Data0+	39	Data5-
7	Data4+	23	Data0-	40	Data4-
8	Data3+	24	GND	41	Data3-
9	Data2+	25	GND	42	Data2-
10	Data19+	26	GND	43	Data19-
11	Data18+	27	GND	44	Data18-
12	Data17+	28	Data11+	45	Data17-
13	Data16+	29	Data11-	46	Data16-
14	Data15+	30	Data10+	47	Data15-
15	Data14+	31	Data10-	48	Data14-
16	Data13+	32	GND	49	Data13-
17	Data12+	33	GND	50	Data12-

Table A-3 summarizes the use of the **LINK A** connector for 4:2:2 mode. The **LINK A** connector carries 10-bit wide UVY information.

Table A-3 LINK A Usage in 4:2:2 Mode

Sample	LINK A
0	Cb0 Y0
1	Cr1 Y1
2	Cb2 Y2
3	Cr3 Y3

Table A-4 summarizes usage for 10-bit RGBA.

Table A-4 LINK A and LINK B Usage in RGBA Mode

Sample	LINK A	LINK B
0	B0 G0	B1 A0
1	R0 G1	R1 A1
2	B2 G2	B3 A2
3	R2 G3	R3 A3

Table A-5 summarizes the use of **LINK A** and **LINK B** connectors for 4:4:4:4 mode. The **LINK A** connector carries a 4:2:2 sampled portion of 10-bit wide UVY; the **LINK B** connector carries the remaining 10-bit UV samples and 10-bit alpha. Usage is similar for 10-bit RGBA.

Table A-5 LINK A and LINK B Usage in 4:4:4:4 Mode

Sample	LINK A	LINK B
0	C_r0 Y0	C_r1 A0
1	C_b0 Y1	C_b1 A1
2	C_r2 Y2	C_r3 A2
3	C_b2 Y3	C_b3 A3

GPI Interface

The General Purpose Interface (GPI) port provides two channels of input and output trigger signal pairs on one connector. This section consists of the following subsections:

GPI Connector

Figure A-2 shows the GPI connector on the HD I/O panel.

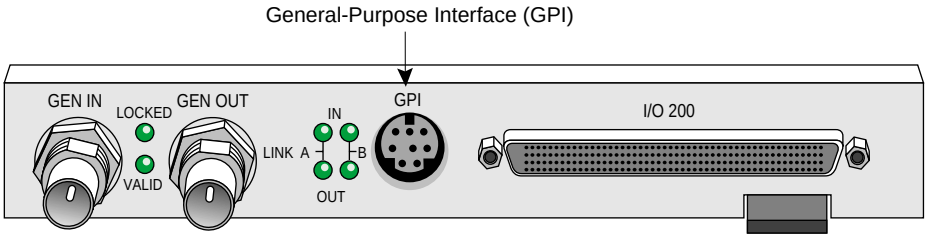


Figure A-2 GPI Connector

Figure A-3 shows the pinouts for the GPI connector.

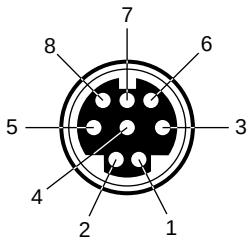


Figure A-3 GPI Pinouts

Table A-6 defines each of the pins in Figure A-3.

Table A-6 GPI Pinouts

Pin	Use	For Video Pipe	CCT/CCR
8	In transmit +	In	CCT +
4	In transmit -	In	CCT -
5	Out transmit +	Out	CCT +
2	Out transmit -	Out	CCT -
6	In receive +	In	CCR +
7	In receive -	In	CCR -
3	Out receive +	Out	CCR +
1	Out receive -	Out	CCR -

Figure A-4 diagrams the relationship between the HD I/O Board's video pipes and the GPI pins.

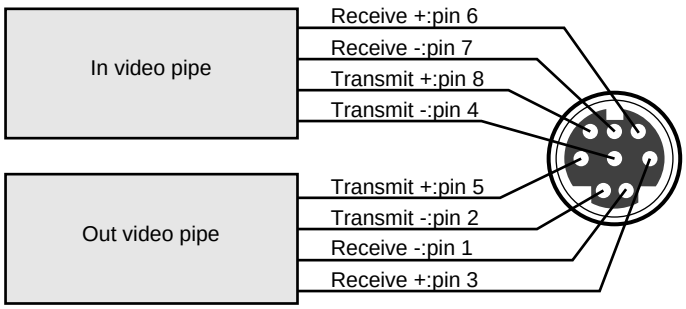


Figure A-4 GPI Pins and HD I/O Video Pipes

GPI Transmitter

GPI contact closure transmit (CCT) outputs use an optically coupled solid-state relay (SSR) to provide a means of electrical isolation for destination equipment.

Table A-7 and Figure A-5 show electrical specifications for the GPI transmitter.

Table A-7 GPI Transmitter Electrical Specifications

Parameter	Value
On resistance	10 ohms typical, 15 ohms maximum
Off resistance	5000 G ohms
Current limit	360 mA typical, 460 mA maximum
Output capacitance	60 pF
Continuous DC load current	180 mA
Output power dissipation	600 mW
Isolation voltage	3750 V rms

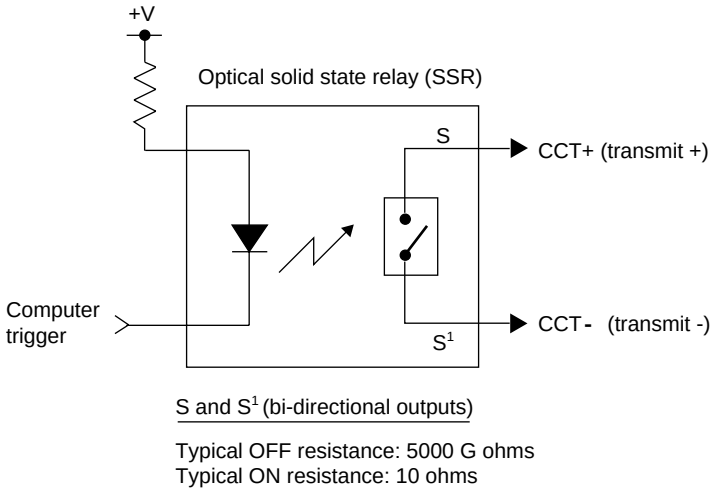


Figure A-5 GPI Transmitter Electrical Specifications

The GPI transmitter can be interfaced to the destination equipment by tying the CCT- terminal to GND and using the CCT+ terminal as a current sink. The input device of the destination equipment can consist of a logic device with active pullup, an optoisolator LED with series-limiting resistor, or relay primary with series-limiting resistor.

The GPI transmitter's logic sense can be swapped (inverted) by tying the CCT+ terminal to the logic power supply (VCC) of the destination equipment and using the CCT- terminal to drive the input of the receiving device.

GPI Receiver

GPI receive (CCR) inputs use an optical isolator device to provide a means of electrical isolation from source equipment. The device consists of a bidirectional input LED optically coupled to a bipolar transistor. A voltage pulse applied across the CCR+/- pins causes the LED to become momentarily forward-biased, which produces a GPI trigger to the computer.

Figure A-6 shows electrical specifications for the GPI receiver.

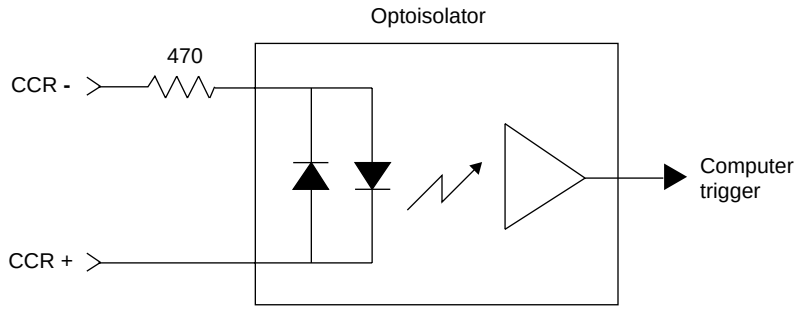


Figure A-6 GPI Receiver Electrical Specifications

Table A-8 summarizes electrical specifications for the GPI receiver optoisolator.

Table A-8 GPI Receiver Input Optoisolator Electrical Specifications

Parameter	Value
Forward voltage (V_F)	1.55 V, 1.2 V typical (IF = 10 mA)
Continuous forward current (I_F)	30 mA
Peak forward current	1000 mA (10 us duration, 1% DC)
Reverse current (I_R)	0.1 uA, 100 uA maximum (V_R = 6 V)
Isolation surge voltage (V_{10})	2500 VAC_{RMS} (t = 1 min)

The GPI receiver can be interfaced to the source equipment by tying either the CCR+ terminal or the CCR- terminal across the output terminals of an optoisolator, solid-state relay, or any device that acts like a single-pole contact switch. The other terminal (CCR- or CCR+) must then be appropriately tied to power or ground. Whenever the logic device is sourcing current (driving a logic high), a GPI trigger is generated.

Note: Proper biasing and current limitations must be observed for the input LED. See Table A-8.

Genlock

The **GEN IN** and **GEN OUT** BNC connectors on the HD I/O front panel provide a passive genlock loopthrough connection. If you attach a cable to one **GEN** connector, you must attach to the other **GEN** connector either a 75-ohm BNC terminator or a cable to other equipment accepting analog sync. If another cable is connected, it must be terminated at the end of the loopthrough chain. Figure A-7 shows the genlock BNCs.

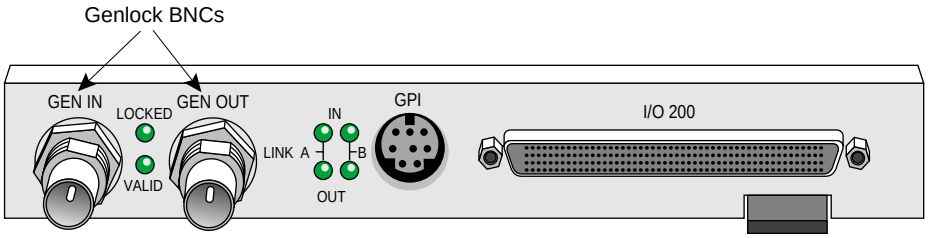


Figure A-7 Genlock BNCs

Setting Up the HD I/O Board for Your Video Hardware

This appendix shows you how to attach video equipment to the HD I/O Board front panel connectors and how to use the video control panel *vcp* to set the board to match your installation.

This appendix covers the following topics:

Figure B-1 shows the HD I/O front panel connectors.

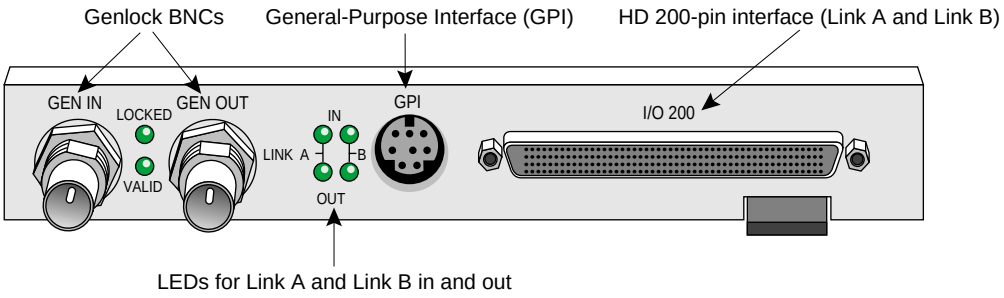


Figure B-1 HD I/O Ports

Setting Up Digital Source Video

The cables supplied with the HD I/O Board each have four 50-in connectors for attaching to equipment that complies with the SMPTE 274M standard. With the HD I/O Board, you can use these cables for 4:4:4:4 dual-link mode or 4:2:2 single-link mode. For 4:2:2 single-link mode, ignore the alpha:

- In 4:4:4:4 mode, Link A carries Y plus the U and V from even-numbered sample points; Link B carries alpha plus the U and V from odd-numbered sample points.

To set up the board for a digital video source, follow these steps:

1. Using the Panasonic-standard or Philips-standard cable included with the HD I/O Board, attach the connector with the green label **SGI <--- PANASONIC P1** or **SGI <--- PHILIPS P1** to a device outputting data.

   Depending on the equipment to be connected to, use the Panasonic or Philips cable. If necessary, see "Cable Connectors" on page 48 in Appendix A for an explanation of the labeling and color coding.

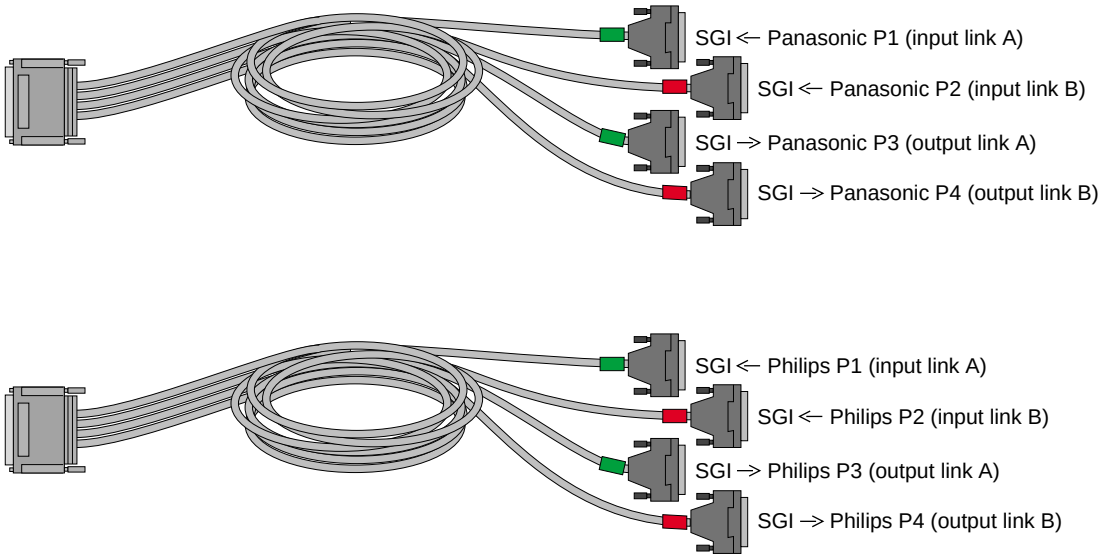


Figure B-2 HD I/O Cables

2. If you are using a device for alpha key data, also attach the connector with the red label **SGI <--- PANASONIC P2** or **SGI <--- PHILIPS P2** to a device outputting alpha.

3. Call up the panel:

   `/usr/sbin/vcp`

4. In the Inputs(s): HD I/O Digital Video Source section of the control panel *vcp* (pointed out in Figure B-3) for the channel(s) you are using, select the format that matches your equipment: Digital Component 4:2:2 (the default) or Digital Component 4:4:4:4.

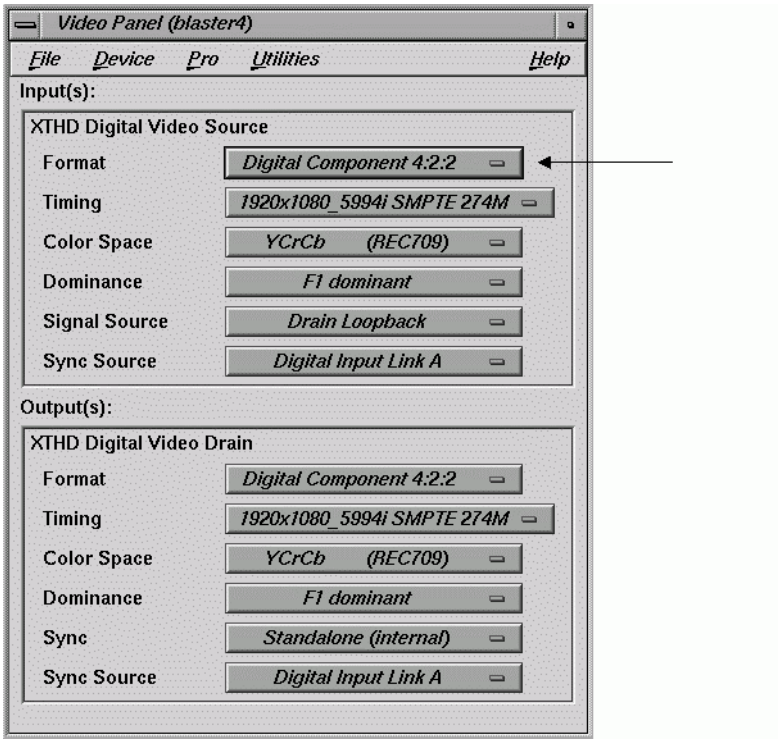


Figure B-3 Selecting Digital Input Video Format in vcp

5. In the Digital Video Source portion of the panel for the channel(s) you are using, select the timing that matches your equipment.

Setting Up the Output (Drain)

To set up the digital video output, follow these steps:

1. Using the Panasonic-standard or Philips-standard cable included with the HD I/O Board, attach the connector with the green label **SGI ---> PANASONIC P3** or **SGI ---> PHILIPS P3** to a device to which you want to send data.

   If you use only one output, you must use this connector.

   Depending on the equipment to be connected to, use the Panasonic or Philips cable. If necessary, see "Cable Connectors" on page 48 in Appendix A for an explanation of the labeling and color coding.

2. If you are using a device for alpha key data, also attach the connector with the red label **SGI ---> PANASONIC P4** or **SGI ---> PHILIPS P4** to a device receiving alpha key data.

3. If necessary, call up the panel (`/usr/sbin/vcp`).

4. In the Output(s): Video Drain section of the control panel (pointed out in Figure B-4), select the format that matches your equipment: Digital Component 4:2:2 (the default) or Digital Component 4:4:4:4.

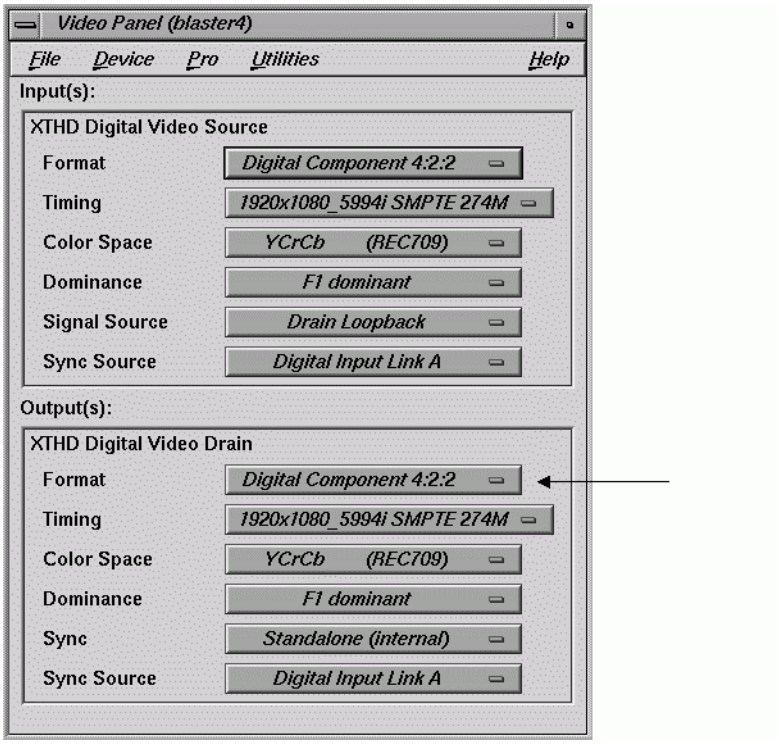


Figure B-4 Selecting Video Drain Format

5. Select the timing that matches your equipment.
6. To set field dominance, in the "Input Timing" menu, select "F1 dominant" for the edit to occur on the nominal video field boundary, or "F2 dominant" for the edit to occur on the intervening field boundary.

   For more information on field dominance, see "Field Dominance" on page 33 in Chapter 2.

Setting Up Sync

This section explains

Setting Up Internal Sync

In the Output(s): Digital Video Drain section of the control panel, select the Sync format that matches your equipment:

- stand-alone (not synced to another device): select Stand-alone (internal)
- output sync to an external source connected to the genlock in: select Genlock

These two choices toggle; Figure B-5 shows the default.

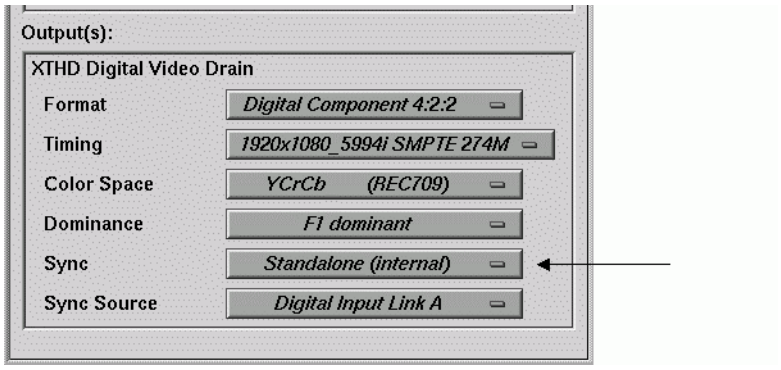


Figure B-5 Setting Stand-alone or Genlock Sync

Setting Up External Sync

To set up the HD I/O Board for an external sync source, follow these steps:

1. If necessary, call up the panel (`/usr/sbin/vcp`).
2. Select the appropriate external sync source in Output(s): Sync Source:
   - For a device connected to **GEN IN**, select External 1920x1080_5994_i or External 525 NTSC depending on your sync source.
   - If you are syncing to the device attached to **SGI <--- PANASONIC/PHILIPS P1**, select Digital Input Link A.
   - If you are syncing to the device attached to **SGI <--- PANASONIC/PHILIPS P2**, select Digital Input Link B.

   Note: See Table 2-3 in Chapter 2 for details on the interrelationships of timing and sync source.

   If a genlock source has not yet been cabled to the HD I/O panel, any application in progress might generate an error message.
3. Connect the sync source equipment to one of the following connectors:
   - the **GEN IN** BNC on the I/O panel (see Figure B-1 if necessary)

   or

   - the cable connector **SGI <--- PANASONIC/PHILIPS P1** or **SGI <--- PANASONIC/PHILIPS P2** (if this cable connector is not already attached to the device supplying sync)
4. If you are using the same signal for other equipment, attach a BNC cable to the **GEN OUT** BNC to loop the signal through the board.

   Make sure the final element in the chain is terminated.

   If the HD I/O Board is the last element in the sync chain, attach a terminator to the **GEN OUT** BNC.
5. Check the genlock LEDs:
   - If the yellow **VALID** LED is lit, an acceptable sync source is attached to the genlock input.
   - If the green **LOCKED** LED is lit, genlock is enabled (via *vcp* or an application), the reference is adequate, and the board is ready to use.

Saving Settings

After you set the values in *vcp* to match your installation, save them; they are written to */usr/etc/video/videod.defaults*. Select "Restore Settings" in the video control panel File menu to load the values in this file to *vcp*.

The last settings saved are automatically loaded every time the system is reinitialized. If the panel is running, current settings are in effect.

Note: You do not need to open the panel to activate its settings.

You can also use File menu choices to restore the factory defaults and close the panel.

Pixel Packings and Color Spaces

This appendix explains

HD I/O Pixel Packings

This section presents each packing used by the HD I/O hardware, providing a diagram and its tokens in the pertinent libraries.

The following topics are covered:

Packings and Color Spaces

A packing

- Determines which of the four components are sampled, either RGBA or CrYCb (sometimes called VYUA)
- Determines the sampling pattern (for example, 4:4:4:4 or 4:2:2:4), which specifies where and how often each component of the image is sampled.
- Allocates a certain number of bits to represent the component samples, and positions those samples along with possible padding in memory; each sample is an unsigned number, unless specified otherwise in the description of the packing

A color space

- Determines the color in each component by specifying the color set
- Specifies a canonical minimum and maximum value for each component, either full range or compressed range (headroom range); see "VL_COLORSPACE" on page 26 in Chapter 2 for an explanation

In most Silicon Graphics libraries, a single token encodes both color space and packing. For example, VL_PACKING_RGBA_8 is a 32-bit packing in the RGBA color space. For the VL of SGI advanced video products, the two parameters are specified separately with different controls: VL_PACKING and VL_COLORSPACE. The color space must be defined with the VL_COLORSPACE control.

Packing Diagram Conventions

In all illustrations, as you move from left to right:

- Each byte goes from the most significant bit to the least significant bit
- The bytes increase in memory address by 1
- Component samples go from most significant bit to least significant bit

Each illustration shows the smallest repeating spatial pattern of component samples that is a multiple of 8 bits wide. No additional padding or alignment is to be inferred. For example, a 24-bit-per-pixel diagram, such as that for VL_PACKING_444_8 (Figure C-1), indicates 3-byte quantities packed together in memory; the values are not padded out to 32-bit boundaries.

Pixel 1		
Byte 1	Byte 2	Byte 3
r r r r r r r r	g g g g g g g g	b b b b b b b b
v v v v v v v v	y y y y y y y y	u u u u u u u u

Figure C-1 VL_PACKING_444_8

The packing defines a bit layout, but for convenience, as shown in Figure C-1, the component slots are filled with the RGBA or CrYCb color set as appropriate. (See "VL_COLORSPACE" on page 26 in Chapter 2 for more information.) For chroma components, Cr and Cb are more accurate terms than V and U; however, this chapter uses the letters V and U in the illustrations of packings for typographical convenience.

Packings that use 4:2:2 sampling also show the location of each component sample: left and right for 4:2:2. The diagrams assume row-major, left-to-right ordering of pixels in memory.

An x ("don't care") in a bit means the following:

- Readers may get any garbage in this bit.
- Writers can leave this bit as garbage.

A 0 means the following:

- Readers may assume this bit is zero.
- Writers can leave this bit as garbage.

An s indicates a padding bit that is a sign extension bit. For the HD I/O Board, this convention applies only to the more significant bits in 12-bit and 13-bit packings with rightward orientation; that is, VL_PACKING_4444_12_in_16_R and VL_PACKING_4444_13_in_16_R.

A p indicates a padding bit in the least significant bits of a left-justified 10-, 12-, or 13-bit word, such as VL_PACKING_R242_10_in_16_L or VL_PACKING_4444_13_in_16_L:

- Readers can assume that the bits are replicated from the component found in the same word: With bits numbered starting with 0 for the least significant, there are n contiguous p bits to the right of the component. The p bits contain a copy of bits [9,9-n+1] of the component.
- Writers can leave the p bits as garbage.

The HD I/O device natively transfers data of all packings in this appendix in real time.

Packings and Library Tokens

Following each packing diagram are comments and library tokens for that packing, listing, where applicable, the color set (RGBA or CrYCb) and the library (VL, OpenGL, and DM) for each library token.

- DM refers to the tokens in */usr/lib/dmedia/dm_image.h*, which are used by several libraries (*libdmedia* (dmParams, dmIC, dmColor), *libmoviefile*, and others). See “VL_COLORSPACE” on page 26 in Chapter 2 for more information.
- The VL includes the packing control value and a color-space control value; for example, VL_PACKING_4_8 + VL_COLORSPACE_{CCIR,YUV}. For the HD I/O Board, you set packing and color space separately for memory nodes. These definitions provide a more flexible way to specify memory layout of pixels and their color spaces.

Note: Because of HDTV’s multiple color spaces, “old style” packings, such as VL_PACKING_Y_8_P, are ambiguous and therefore not supported for the HD I/O Board. You must specify the packing and color space explicitly.

All HD I/O packings are also supported by the DIVO and DIVO-DVC boards.

Packing Naming Conventions

In packing tokens, the following applies:

- _L and _R appended to the end of tokens with padding (0 bits) indicate that the 0 bits are at the left end or the right end of the pattern, respectively; for example, VL_PACKING_4444_10_in_16_L and VL_PACKING_4444_10_in_16_R).
- X before the numerical part of the token at the end of a token indicates a component order other than the standard (RGBA or ABGR, CrYCbA or ACbYCr); for example, DM_IMAGE_PACKING_XBGR.
- R before the numerical part of the token indicates reverse order of the components; for example, VL_PACKING_242_8 and VL_PACKING_R242_8 have the same pattern of component bits, but their order is reversed in VL_PACKING_R242_8.
- Z at the end of the token name means that the packing is padded to the word boundary; for example, the packing in VL_PACKING_2424_10_10_10_2Z is 30 bits per pixel, but it is padded to 32 bits per pixel.

Table C-1 lists the HD I/O packings by number of bits in the pattern of component samples—the order in which they are described in the rest of this section.

Table C-1 HD I/O Packings

Packing	Bits	Color Space
VL_PACKING_ 242_8	16	CrYCb
VL_PACKING_R242_8	16	CrYCb
VL_PACKING_242_10	20	CrYCb
VL_PACKING_R242_10	20	CrYCb
VL_PACKING_444_8	24	RGB/CrYCb
VL_PACKING_R444_8	24	RGB/CrYCb
VL_PACKING_4444_8	32	RGBA/CrYCb
VL_PACKING_R4444_8	32	RGBA/CrYCb

Table C-1 (continued) HD I/O Packings

Packing	Bits	Color Space
VL_PACKING_R0444_8	32	RGBA/CrYCb
VL_PACKING_0444_8	32	RGBA/CrYCb
VL_PACKING_4444_10_10_10_2	32	RGBA/CrYCb
VL_PACKING_R4444_10_10_10_2	32	RGBA/CrYCb
VL_PACKING_2424_10_10_10_2Z	32	CrYCb
VL_PACKING_R2424_10_10_10_2Z	32	CrYCb
VL_PACKING_242_10_in_16_L	32	CrYCb
VL_PACKING_242_10_in_16_R	32	CrYCb
VL_PACKING_R242_10_in_16_L	32	CrYCb
VL_PACKING_R242_10_in_16_R	32	CrYCb
VL_PACKING_444_12_in_16_L	48	RGB
VL_PACKING_444_12_in_16_R	48	RGB

The packings are explained in these categories:

16-Bit Pixel Packings

Figure C-2 shows VL_PACKING_242_8, a 16-bit CrYCb packing.

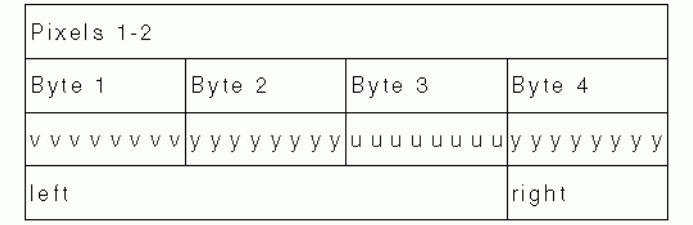


Figure C-2 VL_PACKING_242_8

Note: Cr and Cb are more accurate terms than V and U; however, this appendix uses the letters V and U in the illustrations of packings for typographical convenience.

This rarely used packing is VL_PACKING_242_8 + VL_COLORSPACE_{CCIR,YUV} in the VL. It samples chroma and luma in a 4:2:2 pattern. See “Sampling Patterns” on page 86.

Figure C-3 shows VL_PACKING_R242_8, a 16-bit 4:2:2 CrYCb packing. The most commonly used 4:2:2 packing, it is used by other Silicon Graphics video hardware.

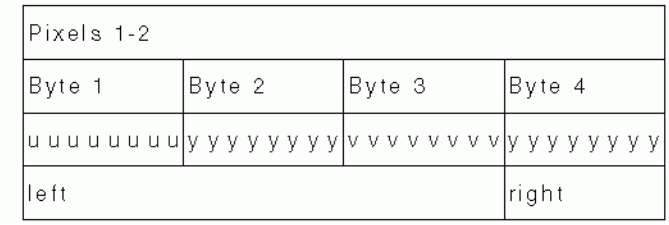


Figure C-3 VL_PACKING_R242_8

This packing is

- VL_PACKING_R242_8 + VL_COLORSPACE_{CCIR,YUV}
- GL_YCRCB_422_SGIX GL_UNSIGNED_BYTE in OpenGL
- DM_IMAGE_PACKING_CbYCrY in DM

20-Bit Pixel Packings

Figure C-4 shows VL_PACKING_242_10, a 20-bit CrYCb packing.

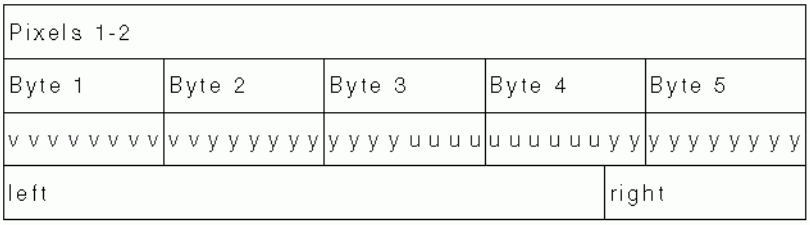


Figure C-4 VL_PACKING_242_10

This packing is VL_PACKING_242_10 + VL_COLORSPACE {CCIR,YUV}. It uses 4:2:2 sampling.

Figure C-5 shows VL_PACKING_R242_10, a 20-bit CrYCb packing.

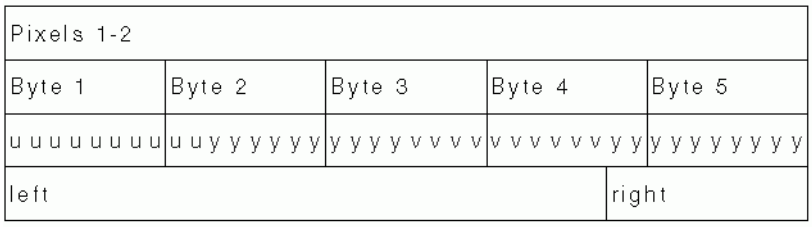


Figure C-5 VL_PACKING_R242_10

This packing is VL_PACKING_R242_10 + VL_COLORSPACE {CCIR,YUV}. It uses 4:2:2 sampling.

Note: Cr and Cb are more accurate terms than V and U; however, this appendix uses the letters V and U in the illustrations of packings for typographical convenience.

24-Bit Pixel Packings

Figure C-6 shows VL_PACKING_444_8, a 24-bit RGB/CrYCb packing.

Pixel 1		
Byte 1	Byte 2	Byte 3
r r r r r r r r	g g g g g g g g	b b b b b b b b
v v v v v v v v	y y y y y y y y	u u u u u u u u

Figure C-6 VL_PACKING_444_8

This packing is

- RGB:
  - GL_RGB GL_UNSIGNED_BYTE in OpenGL
  - VL_PACKING_444_8 + VL_COLORSPACE_{RGB,RP175}
  - PM_RGB GL_UNSIGNED_BYTE on RealityEngine (IRIS GL)
  - DM_IMAGE_PACKING_RGB in DM
- CrYCb: VL_PACKING_444_8 + VL_COLORSPACE_{YUV/YCRCB}

Figure C-7 shows VL_PACKING_R444_8, a 24-bit RGB/CrYCb packing.

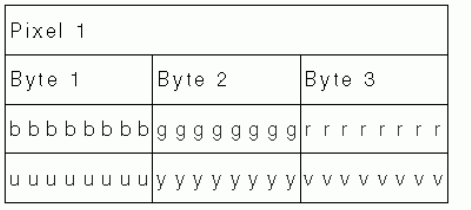


Figure C-7 VL_PACKING_R444_8

This packing is

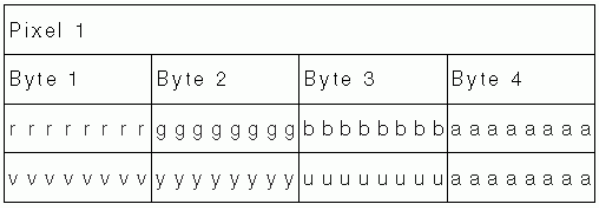
- RGB:
  - VL_PACKING_R444_8 + VL_COLORSPACE_{RGB,RP175}
  - DM_IMAGE_PACKING_BGR in DM
  - PM_BGR PM_UNSIGNED_BYTE on RealityEngine (IRIS GL)
- CrYCb:
  - VL_PACKING_R444_8 + VL_COLORSPACE_{CCIR,YUV}
  - DM_IMAGE_PACKING_CbYCr in DM

32-Bit Pixel Packings

This section explains

OpenGL-Like 32-Bit Pixel Packing

Figure C-8 shows VL_PACKING_4444_8, an OpenGL-like 32-bit packing. This packing, supported by many Silicon Graphics video products, is the most commonly used OpenGL packing.



Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
r r r r r r r r	g g g g g g g g	b b b b b b b b	a a a a a a a a
v v v v v v v v	y y y y y y y y	u u u u u u u u	a a a a a a a a

Figure C-8 VL_PACKING_4444_8

This packing is

- RGBA:
  - VL_PACKING_4444_8 + VL_COLORSPACE_{RGB,RP175}
  - GL_RGBA GL_UNSIGNED_BYTE in OpenGL (the default)
  - DM_IMAGE_PACKING_RGBA in DM
  - PM_RGBA PM_UNSIGNED_BYTE on RealityEngine (IRIS GL)
- CrYCbA: VL_PACKING_4444_8 + VL_COLORSPACE_{CCIR,YUV}

Note: Cr and Cb are more accurate terms than V and U; however, this appendix uses the letters V and U in the illustrations of packings for typographical convenience.

IRIS GL-Like 32-Bit Pixel Packings

Figure C-9 shows VL_PACKING_R4444_8, an IRIS GL-like 32-bit packing. This packing, supported by many Silicon Graphics video products, is the default IRIS GL packing.

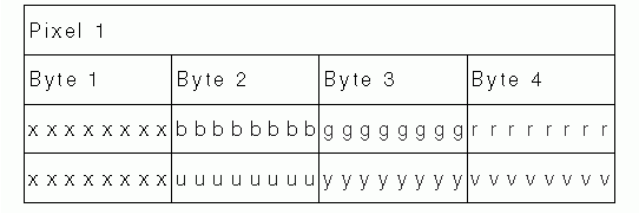
Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
a a a a a a a a	b b b b b b b b	g g g g g g g g	r r r r r r r r
a a a a a a a a	u u u u u u u u	y y y y y y y y	v v v v v v v v

Figure C-9 VL_PACKING_R4444_8

This packing is

- RGBA:
  - VL_PACKING_R4444_8 + VL_COLORSPACE_{RGB,RP175}
  - GL_ABGR_EXT GL_UNSIGNED_BYTE in OpenGL
  - DM_IMAGE_PACKING_ABGR in DM
  - PM_ABGR PM_UNSIGNED_BYTE (default IRIS GL packing)
- CrYCbA: VL_PACKING_R4444_8 + VL_COLORSPACE_{CCIR,YUV}

Figure C-10 shows VL_PACKING_R0444_8, an IRIS GL-like 32-bit packing. This packing is supported by many Silicon Graphics video products.



Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
x x x x x x x x	b b b b b b b b	g g g g g g g g	r r r r r r r r
x x x x x x x x	u u u u u u u u	y y y y y y y y	v v v v v v v v

Figure C-10 VL_PACKING_R0444_8

- RGB:
  - VL_PACKING_R0444_8 + VL_COLORSPACE_{RGB,RP175}
  - DM_IMAGE_PACKING_XBGR

    Use DM_IMAGE_PACKING_ABGR instead of this packing unless you specifically want to inform a piece of software (such as dmColor) not to spend processing time on the alpha channel.
- CrYCb: VL_PACKING_R0444_8 + VL_COLORSPACE_{CCIR,YUV}

Note: Cr and Cb are more accurate terms than V and U; however, this appendix uses the letters V and U in the illustrations of packings for typographical convenience.

32-Bit Pixel Packing for QuickTime

Figure C-11 shows VL_PACKING_0444_8, a 32-bit packing used for QuickTime files (uncompressed format without alpha).

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
x x x x x x x x	r r r r r r r r	g g g g g g g g	b b b b b b b b
x x x x x x x x	v v v v v v v v	y y y y y y y y	u u u u u u u u

Figure C-11 VL_PACKING_0444_8

This packing is

- RGB:
  - VL_PACKING_0444_8 + VL_COLORSPACE_{RGB,RP175}
  - DM_IMAGE_PACKING_XRGB in DM
- CrYCb: VL_PACKING_0444_8 + VL_COLORSPACE_{CCIR,YUV}

4:4:4:4 10_10_10_2 32-Bit Pixel Packings

Figure C-12 shows VL_PACKING_4444_10_10_10_2, a 32-bit 4:4:4:4 10_10_10_2 packing.

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
r r r r r r r r	r r g g g g g g	g g g g b b b b	b b b b b b a a
v v v v v v v v	v v y y y y y y	y y y y u u u u	u u u u u u a a

Figure C-12 VL_PACKING_4444_10_10_10_2

This packing is

- RGBA:
  - VL_PACKING_4444_10_10_10_2 + VL_COLORSPACE_{RGB,RP175}
  - GL_RGBA GL_UNSIGNED_INT_10_10_10_2_EXT in OpenGL
- CrYCbA: VL_PACKING_4444_10_10_10_2 + VL_COLORSPACE_{CCIR,YUV}

Figure C-13 shows VL_PACKING_R4444_10_10_10_2, an alternate 32-bit 4:4:4:4 10_10_10_2 packing.

Pixel 1			
Byte 1	Byte 2	Byte 3	Byte 4
a a b b b b b b	b b b b g g g g	g g g g g g r r	r r r r r r r r
a a u u u u u u	u u u u y y y y	y y y y y y v v	v v v v v v v v

Figure C-13 VL_PACKING_R4444_10_10_10_2

This packing is

- RGBA:
  - VL_PACKING_R4444_10_10_10_2 + VL_COLORSPACE_{RGB,RP175}
  - GL_ABGR GL_UNSIGNED_INT_10_10_10_2_EXT in OpenGL

- CrYCbA: VL_PACKING_R4444_10_10_10_2 + VL_COLORSPACE_{CCIR,YUV}

4:2:2:4 10_10_10_2 32-Bit Pixel Packings

Figure C-14 shows VL_PACKING_2424_10_10_10_2Z, the 4:2:2:4 10_10_10_2 32-bit CrYCbA packing.

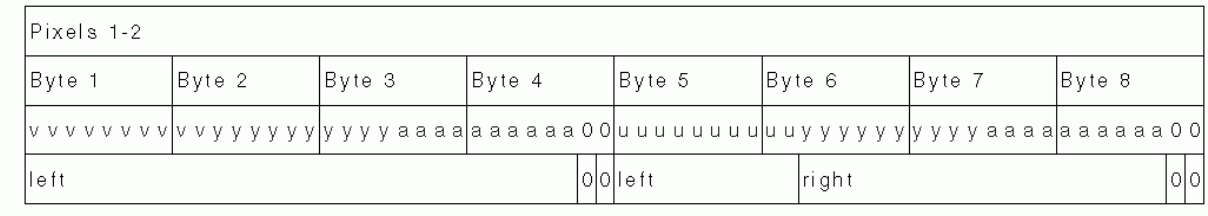


Figure C-14 VL_PACKING_2424_10_10_10_2Z

This packing is VL_PACKING_2424_10_10_10_2Z + VL_COLORSPACE_{CCIR,YUV} in the VL.

Figure C-15 shows VL_PACKING_R2424_10_10_10_2Z, an alternate 4:2:2:4 10_10_10_2 32-bit packing.

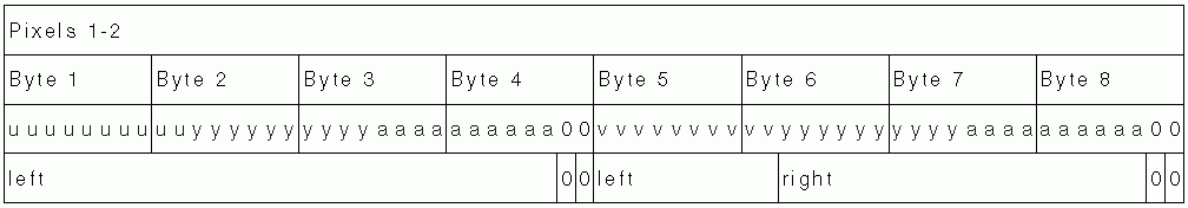


Figure C-15 VL_PACKING_R2424_10_10_10_2Z

This packing is VL_PACKING_R2424_10_10_10_2Z + VL_COLORSPACE_{CCIR,YUV} in the VL.

Note: Cr and Cb are more accurate terms than V and U; however, this appendix uses the letters V and U in the illustrations of packings for typographical convenience.

4:2:2 10_in_16 32-Bit Pixel Packings

The diagrams of packings that use 4:2:2 sampling show the location (left and right) of each component sample. Only DIVO and DIVO-DVC use this packing.

Figure C-16 shows VL_PACKING_242_10_in_16_L, a 4:2:2 10_in_16 32-bit CrYCb packing. This packing is supported by several Silicon Graphics video products. For an explanation of the p bit, see “Packing Diagram Conventions” on page 69.

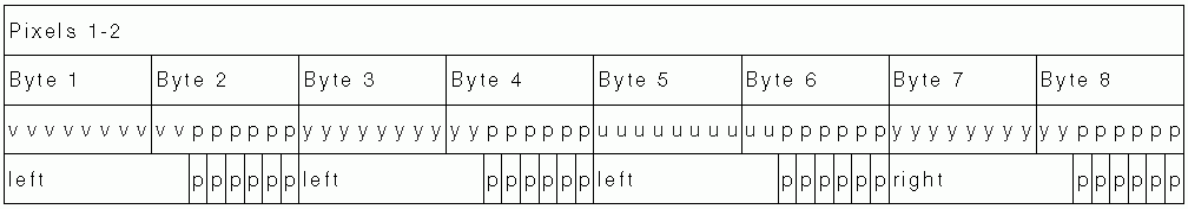


Figure C-16 VL_PACKING_242_10_in_16_L

This packing is VL_PACKING_242_10_in_16_L + VL_COLORSPACE_{CCIR,YUV} in the VL.

Figure C-17 shows VL_PACKING_242_10_in_16_R, a 4:2:2 10_in_16 32-bit CrYCb packing.

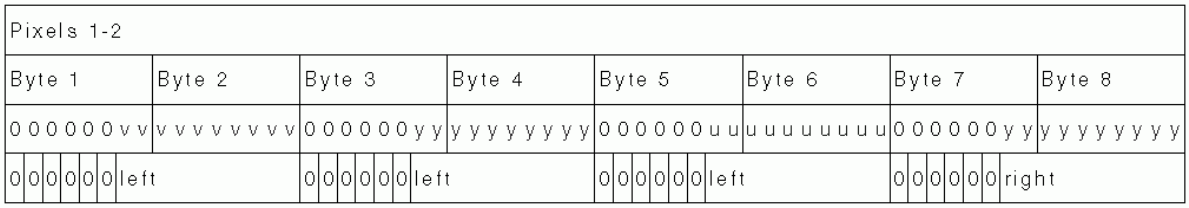


Figure C-17 VL_PACKING_242_10_in_16_R

This packing is CrYCb VL_PACKING_242_10_in_16_R + VL_COLORSPACE_{CCIR,YUV} in the VL.

Figure C-18 shows VL_PACKING_R242_10_in_16_L, a 4:2:2 10_in_16 32-bit CrYCb packing. This packing is supported by several Silicon Graphics video products. For an explanation of the p bit, see “Packing Diagram Conventions” on page 69.

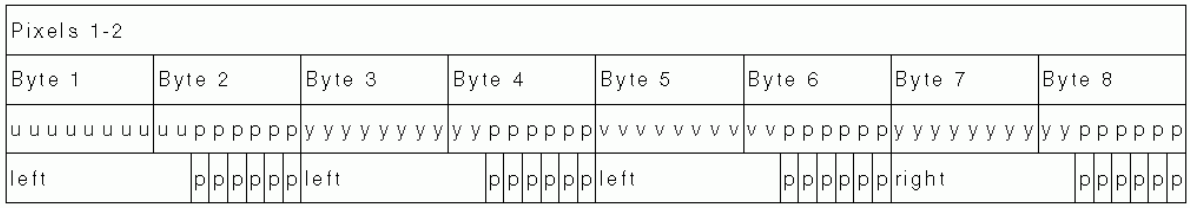


Figure C-18 VL_PACKING_R242_10_in_16_L

This packing is VL_PACKING_R242_10_in_16_L + VL_COLORSPACE_{CCIR,YUV} in the VL.

Figure C-19 shows VL_PACKING_R242_10_in_16_R, a 4:2:2 10_in_16 32-bit CrYCb packing.

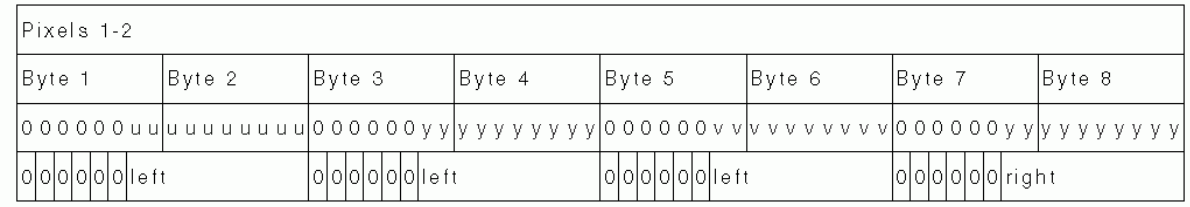


Figure C-19 VL_PACKING_R242_10_in_16_R

This packing is CrYCb VL_PACKING_R242_10_in_16_R + VL_COLORSPACE_{CCIR,YUV} in the VL.

Note: Cr and Cb are more accurate terms than V and U; however, this appendix uses the letters V and U in the illustrations of packings for typographical convenience.

48-Bit Pixel Packings

Figure C-20 shows VL_PACKING_444_12_in_16_L, a 48-bit RGB packing. For an explanation of the p bit, see “Packing Diagram Conventions” on page 69.

Pixel 1					
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
r r r r r r r r	r r r r p p p p	g g g g g g g g	g g g g p p p p	b b b b b b b b	b b b b p p p p

Figure C-20 VL_PACKING_444_12_in_16_L

This packing is VL_PACKING_444_12_in_16_L + VL_COLORSPACE_{RGB,RP175} in the VL, new style.

Figure C-21 shows VL_PACKING_444_12_in_16_R, a 64-bit RGBA packing for use with extended RGB components.

Pixel 1					
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
s s s s r r r r	r r r r r r r r	s s s s g g g g	g g g g g g g g	s s s s b b b b	b b b b b b b b

Figure C-21 VL_PACKING_444_12_in_16_R

This packing is VL_PACKING_444_12_in_16_R + VL_COLORSPACE_{RGB,RP175} in the VL, new style.

Note: The components in this packing are signed, with positive and negative values varying by color space. The s in the more significant bits in Figure C-21 indicates a sign extension padding bit.

Sampling Patterns

Sampling patterns are

4:4:4 and 4:4:4:4 Sampling

Some of the diagrams in the "HD I/O Pixel Packings" section indicate 4:4:4 or 4:4:4:4 sampling. This video industry terminology means that each of the three or four components is sampled at every pixel. Figure C-22 diagrams this sampling pattern.

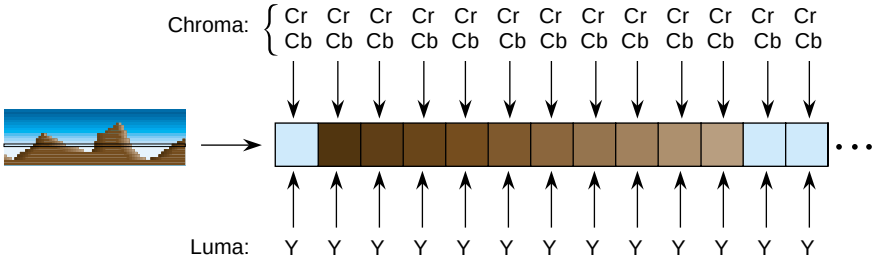


Figure C-22 4:4:4 Sampling

4:2:2 and 4:2:2:4 Sampling

The packings shown in diagrams that indicate 4:2:2 sampling make sense only in the CrYCbA color spaces. For every two pixels, there are two luma samples (two Ys) but only one chroma sample (one sample of Cr and Cb, which together determine the chroma), as shown in Figure C-23.

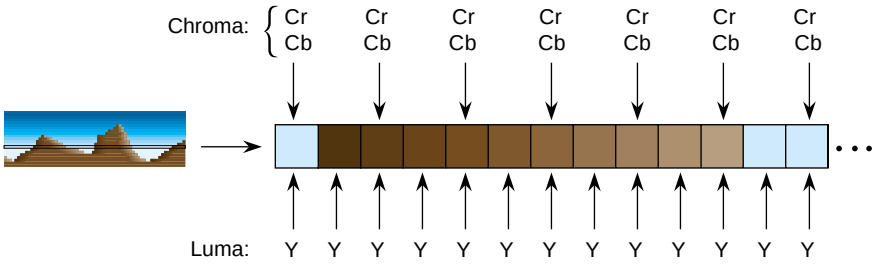


Figure C-23 4:2:2 Sampling

The chroma samples belong at the same instant in space as the left Y sample (the chrominance samples and the left Y are co-sited). The diagrams for 4:2:2 packings in the "HD I/O Pixel Packings" section of this appendix show the location of each Y, Cr, or Cb component as left or right. The first pixel of each line is a left pixel.

Converting 4:4:4 video to 4:2:2 video is like converting 44.1 kHz audio into 22.05 kHz audio: just dropping every other Cr,Cb sample yields extremely poor results. Video devices that need to convert between 4:4:4 and 4:2:2 use carefully designed filters. The characteristics of the required filter are specified in ITU-R BT.601 (Rec. 601).

4:2:2 sampled packings that also include alpha are called 4:2:2:4. This method has one alpha value per pixel, like the Y value.

Programming Methods for Real-Time Digital Media Recording and Playback

This appendix explains the following real-time disk I/O concepts:

The methods described in this appendix use digital media buffers (DMbuffers), a real-time data transport facility. See the *Digital Media Programming Guide* (document number 007-1799-060 or later, hereafter referred to as the DMPG) for more details. The emphasis here is not on how data is acquired from or transported to the video device, but rather on how data is moved to disk in real time.

The DMPG covers basic digital media programming concepts. The directory */usr/share/src/dmedia/video/XTHD* contains two simple programming examples:

- `sdhd_memtovid.c` illustrates how video data is copied out of the DMbuffers for the simpler non-real-time case
- `sdhd_vidtogfx.c` illustrates how video data is transferred into memory and from there into graphics

At an abstract level, high-bandwidth throughput is simple; the work is in the details, as explained in this appendix.

Note: For source code examples for developer I/O, contact the SGI Developer program, for example, through the Web site http://www.sgi.com.

Digital Media Buffers

In IRIX 6.5.4 and later, you can tune how memory is allocated for digital media buffer pools. This capability can offer a performance advantage for some HD I/O applications.

For example, the parameter DM_POOL_NODE_MASK allows you to request memory allocation on particular physical node cards. Each bit in the node mask enables memory allocation on that node. For example, to allocate buffers distributed across nodes 0 and 1, set the node mask to 3:

```
long long mask = 0x3;

if(dmParamsSetLongLong(plist,DM_POOL_NODE_MASK,mask) ) {
  fprintf(stderr, “Error setting dmparam for pool node mask\n”);
  exit(-1);
}

if (dmBufferCreatePool(plist, &pool) != DM_SUCCESS) {
  fprintf(stderr,”Error creating dmbuffer pool\n”);
  exit(-1);
}
```

Note that the node mask is a long long parameter. To see the effect of setting this parameter, use *osview -a* and examine the free memory on each node. For more information on dmBuffers, see the dmbuffercreatepool(3dm) man page.

Direct I/O

The most efficient way to move data on and off a disk device is to use the XFS filesystem with direct I/O mode and large data transfer sizes. If large transfer sizes cannot be achieved, you can combine memory pages from noncontiguous locations using writev(2) or readv(2). Finally, you can use asynchronous I/O to queue multiple I/O requests to the kernel without waiting for blocked calls to return. Other real-time software features and products, such as REACT, can be used to assure low-latency interrupts and high-priority scheduling, but are not absolutely necessary for digital media applications.

Normally, when a disk file is opened with no status flags specified, a call to write(2) for that file returns as soon as the data is copied to a buffer managed by the device driver (see open(2)). The actual disk write may not occur until considerable time elapses. A common pool of disk buffers is used for all disk files.

Disk buffering is integrated with the virtual memory paging mechanism. A daemon executes periodically and initiates output of buffered blocks according to the age of the data and the needs of the system. You can force the writing of all pending output for a file by calling fsync(2) or by opening the file and specifying the O_SYNC flag. However, the process blocks until the data is written to disk, and all output data must still be copied from the buffer in the user address space to a buffer in the kernel address space. See Chapter 8, "Optimizing Disk I/O for a Real-Time Program," in the *REACT Real Time Programmer's Guide* for details.

If you use the O_DIRECT flag, writes to the file are executed directly from your program's buffer, and the data is not copied to a buffer in the kernel first. Because the filesystem cache is bypassed, your application must manage buffer alignment and block size specification. To use O_DIRECT, you must transfer data in quantities that are multiples of the filesystem block size. The following code shows how to query the filesystem block size and system DMA transfer size limit.

```
struct dioattr da;
struct stat fileStat;
char *ioFileName = "videodata";
int ioBlockSize, ioMaxXferSize;

ioFileFD = open(ioFileName,O_DIRECT | O_RDWR | O_CREAT | O_TRUNC,0644);
if (ioFileFD < 0)
    return(DM_FAILURE);
if (fcntl(ioFileFD, F_DIOINFO, &da) < 0)
    return(DM_FAILURE);
ioBlockSize = da.d_miniosz;

ioMaxXferSize = da.d_maxiosz;
```

The two important constraints of direct I/O with XFS are memory address alignment and buffer length. Direct I/O requires all memory addresses to be page-aligned. XFS requires buffers to be allocated as a multiple of the filesystem block size, *ioBlockSize*. DMbuffers are guaranteed to be page-aligned, but to ensure that the buffers are properly padded, you must set the buffer size, *bytesPerXfer,* to the size of the image data you will transfer rounded up to the nearest multiple of *ioBlockSize*.

```
VLServer vlServer;
VLPath vlPath;
DMparams * paramsList;
int dmBufferPoolSize = 30;  /* 1 second of video */
int vlBytesPerImage = vlGetTransferSize(vlServer, vlPath);
int ioBlocksPerImage = (vlBytesPerImage+ioBlockSize - 1) / ioBlockSize;
int bytesPerXfer = ioBlocksPerImage * ioBlockSize;
```

```
if (dmBufferSetPoolDefaults(paramsList,dmBufferPoolSize,bytesPerXfer,
    DM_TRUE, DM_TRUE) == DM_FAILURE) {
    fprintf(stderr, “error setting pool defaults\n”);
    return(DM_FAILURE);
}
```

All Silicon Graphics systems have a configurable maximum DMA transfer size (see systune(1M)). This value should be compared with the user’s I/O request size.

```
if (bytesPerXfer > ioMaxXferSize) {
    fprintf(“DMA request size is too small. Reconfigure with
             systune()\n”);
    return(DM_FAILURE);
}
```

Multiprocessing

Some aspects of digital media programming lend themselves to a multiprocessing programming model. On a multiprocessor system, the various tasks of moving multiple streams of video and audio data on and off disk, serial I/O control of external video equipment and input devices, processing of video data, or the transport of video data in and out of the graphics framebuffer can be assigned to different processors. New processes must be created with all virtual space attributes (shared memory, mapped files, data space) shared. The following fragment illustrates how to create a process to perform video recording.

```
if ((video_recorder_pid = sproc(video_recorder, PR_SADDR|PR_SFDS))<0){
    perror(“video_recorder”);
    exit(DM_FAILURE);
}
```

If you use multiprocessing, note the following caveats:

- When VL calls are made, VL objects such as VLServer, VLPath, VLNode, and so on, are passed through the kernel to the video driver. However, you cannot create any VL objects without first creating a VLServer, from which everything else is instanced.
- In a process share group, only one VL call whose arguments derive from a VLServer can execute at a time. This requirement applies even to VL calls that do not explicitly take a VLServer as an argument (for example, vlBufferAdvise(3dm)).

- You can use objects derived from a given VLServer in any number of threads as long as you use a locking scheme, such as usnewsema(3P) or pthread_mutex_init(3P), to make the use in each thread mutually exclusive of a use in any of the other threads.

The VL error state, returned by vlGetErrno(3dm), is global to a share group, not per VLServer. If a VL call using one VLServer in one thread executes simultaneously with a VL call using another VLServer in another thread, both calls try to set the error state returned by **vlGetErrno()**. This call should be global only to the thread, not to the entire process share group.

Asynchronous I/O

Asynchronous I/O allows an application to process multiple read or write requests simultaneously. On Silicon Graphics platforms, asynchronous I/O is available through the *aio* facility. This facility, based on sproc(2)'ed processes, provides all of the benefits of multiprocessing for free. Because multiple I/O requests might be outstanding, when you use asynchronous I/O, the round-trip delay between making a request, having it serviced, and issuing another request is removed. Any process-scheduling delay between these steps is also eliminated.

Because asynchronous I/O operations complete out of sequence, the application must keep track of the order in which data appears in the DMbuffers. DMbuffers are contained in a DMbufferPool; the pool itself is unordered and buffers can be obtained and returned to the pool in any order. Ordering is achieved by a first-in-first-out queue and maintained only while the buffers reside in the queue. Once the buffers are dequeued, the application is free to impose any processing order.

Installing HD I/O Software on a New Disk

Use *inst* to install the HD I/O software. For more detailed information on booting the miniroot from a remote CD-ROM, see the *IRIX Admin: Software Installation and Licensing* manual (007-1364-*xxx*).

To perform the software installation, follow these steps:

1. Make sure a CD-ROM drive is attached to your system (or is available on the network).
2. If necessary, upgrade your system software to IRIX 6.5.4 or later, from the CD set included with the HD I/O software.
3. Install the HD I/O software.
4. Reboot the system.
5. Enter **hinv** at the IRIX level to determine whether your system recognizes all installed boards. A typical output might appear as follows:

```
# hinv
2 180 MHZ IP27 Processors
CPU: MIPS R10000 Processor Chip Revision: 2.6
FPU: MIPS R10010 Floating Point Chip Revision: 0.0
Main memory size: 256 Mbytes
Instruction cache size: 32 Kbytes
Data cache size: 32 Kbytes
Secondary unified instruction/data cache size: 1 Mbyte
Integral SCSI controller 0: Version QL1040B, single ended
  Disk drive: unit 1 on SCSI controller 0
Integral SCSI controller 1: Version QL1040B, single ended
IOC3 serial port: tty1
IOC3 serial port: tty2
Integral Fast Ethernet: ef0, version 1, module 1, slot io1, pci 2
Origin BASEIO board, module 1 slot 1: Revision 3
XT-HDIO Video: controller 2, unit 0, version 0x0
IOC3 external interrupts: 1
```

You can use *hinv -m* to obtain the board serial number, revision level, and other statistics. A typical output might appear as follows:

```
# hinv -m
        MODULEID Board: barcode K0002031   part              rev
       8P12-MPLN Board: barcode CEM652     part 013-1547-003 rev  A
            IP27 Board: barcode DAW914     part 030-1266-001 rev  C
         BASEIO Board: barcode DCH915     part 030-1124-002 rev  A
          HDTV1 Board: barcode DPX101     part 030-1282-001 rev  A
    HDTV_GENLOCK Board: barcode DPD453     part 030-1382-003 rev  A
2 180 MHZ IP27 Processors
CPU: MIPS R10000 Processor Chip Revision: 2.6
FPU: MIPS R10010 Floating Point Chip Revision: 0.0
Main memory size: 256 Mbytes
Instruction cache size: 32 Kbytes
Data cache size: 32 Kbytes
Secondary unified instruction/data cache size: 1 Mbyte
Integral SCSI controller 0: Version QL1040B, single ended
  Disk drive: unit 1 on SCSI controller 0
Integral SCSI controller 1: Version QL1040B, single ended
IOC3 serial port: tty1
IOC3 serial port: tty2
Integral Fast Ethernet: ef0, version 1, module 1, slot io1, pci 2
Origin BASEIO board, module 1 slot 1: Revision 3
XT-HDIO Video: controller 2, unit 0, version 0x0
IOC3 external interrupts: 1
```

Index

D

E

F

G

H

R

S

T

U

V

X

Y

Z