User's Guide

EDT Camera Link Simulator (CLS) Family

for PCI or PCI Express

Date: 2014 November 10
Rev#: 0004

Contact

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich
Germany
www.skyblue.de
info@skyblue.de
Tel. +49 (0) 89 - 780297 0

Terms of Use Agreement

Definitions. This agreement, between Engineering Design Team, Inc. (“Seller”) and the user or distributor (“Buyer”), covers the use and distribution of the following items provided by Seller: a) the binary and all provided source code for any and all device drivers, software libraries, utilities, and example applications (collectively, “Software”); b) the binary and all provided source code for any and all configurable or programmable devices (collectively, “Firmware”); and c) the computer boards and all other physical components (collectively, “Hardware”). Software, Firmware, and Hardware are collectively referred to as “Products.” This agreement also covers Seller’s published Limited Warranty (“Warranty”) and all other published manuals and product information in physical, electronic, or any other form (“Documentation”).

License. Seller grants Buyer the right to use or distribute Seller’s Software and Firmware Products solely to enable Seller’s Hardware Products. Seller’s Software and Firmware must be used on the same computer as Seller’s Hardware. Seller’s Products and Documentation are furnished under, and may be used only in accordance with, the terms of this agreement. By using or distributing Seller’s Products and Documentation, Buyer agrees to the terms of this agreement, as well as any additional agreements (such as a nondisclosure agreement) between Buyer and Seller.

Export Restrictions. Buyer will not permit Seller’s Software, Firmware, or Hardware to be sent to, or used in, any other country except in compliance with applicable U.S. laws and regulations. For clarification or advice on such laws and regulations, Buyer should contact: **U.S. Department of Commerce, Export Division, Washington, D.C., U.S.A.**

Limitation of Rights. Seller grants Buyer a royalty-free right to modify, reproduce, and distribute executable files using the Seller’s Software and Firmware, provided that: a) the source code and executable files will be used only with Seller’s Hardware; b) Buyer agrees to indemnify, hold harmless, and defend Seller from and against any claims or lawsuits, including attorneys’ fees, that arise or result from the use or distribution of Buyer’s products containing Seller’s Products. Seller’s Hardware may not be copied or recreated in any form or by any means without Seller’s express written consent.

No Liability for Consequential Damages. In no event will Seller, its directors, officers, employees, or agents be liable to Buyer for any consequential, incidental, or indirect damages (including damages for business interruptions, loss of business profits or information, and the like) arising out of the use or inability to use the Products, even if Seller has been advised of the possibility of such damages. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to Buyer. Seller’s liability to Buyer for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, product liability, tort including negligence, or otherwise), will be limited to fifty U.S. dollars ($50.00).

Limited Hardware Warranty. Seller warrants that the Hardware it manufactures and sells shall be free of defects in materials and workmanship for a period of 12 months from date of shipment to initial Buyer. This warranty does not apply to any product that is misused, abused, repaired, or otherwise modified by Buyer or others. Seller’s sole obligation for breach of this warranty shall be to repair or replace (F.O.B. Seller’s plant, Beaverton, Oregon, U.S.A.) any goods that are found to be non-conforming or defective as specified by Buyer within 30 days of discovery of any defect. Buyer shall bear all installation and transportation expenses, and all other incidental expenses and damages.

Limitation of Liability. *In no event shall Seller be liable for any type of special consequential, incidental, or penal damages, whether such damages arise from, or are a result of, breach of contract, warranty, tort (including negligence), strict liability, or otherwise.* All references to damages herein shall include, but not be limited to: loss of profit or revenue; loss of use of the goods or associated equipment; costs of substitute goods, equipment, or facilities; downtime costs; or claims for damages. Seller shall not be liable for any loss, claim, expense, or damage caused by, contributed to, or arising out of the acts or omissions of Buyer, whether negligent or otherwise.

No Other Warranties. Seller makes no other warranties, express or implied, including without limitation the implied warranties of merchantability and fitness for a particular purpose, regarding Seller’s Products or Documentation. Seller does not warrant, guarantee, or make any representations regarding the use or the results of the use of the Products or Documentation or their correctness, accuracy, reliability, currentness, or otherwise. All risk related to the results and performance of the Products and Documentation is assumed by Buyer. The exclusion of implied warranties is not permitted by some jurisdictions. The above exclusion may not apply to Buyer.

Disclaimer. Seller’s Products and Documentation, including this document, are subject to change without notice. Documentation does not represent a commitment from Seller.

Contents

EDT Camera Link Simulator (CLS) Family

Overview

This guide covers EDT Camera Link simulator (CLS) boards. A CLS board sends simulated Camera Link images to a frame grabber via Camera Link cabling. As an alternative, the PCIe simulator (PCIe8 DVa CLS) can be converted to a fully functional frame grabber (PCIe8 DVa C-Link) via a simple firmware utility, as outlined in the EDT user's guide for Camera Link PCIe frame grabbers (see Related Resources on page 7).

Table 1 lists each EDT CLS product, along with modes supported and other key information.

Table 1. Products in the EDT CLS Family

Product	Status	Bus type	Modes	Data rate	Notes / Unique Features
PCIe8 DVa CLS	Current ("DVa"*)	PCIe (8-lane)	Base, dual base, medium, or full mode	Up to 850 MB/s	Easily converts to a frame grabber. As a frame grabber, Power over Camera Link (PoCL) can be enabled; as a simulator, PoCL must be disabled.
PCIe8 DV CLS	Legacy ("DV"*)	PCIe (8-lane)	Base, dual base, medium, or full mode	Up to 300 MB/s	Functionally equivalent to "DVa" version, except it has lower data speed and does not provide frame grabber PoCL support.
PCI DV CLS	Legacy ("DV"*)	PCI (32 bits / 66 MHz)	Base (8-bit, 1- or 2-tap; or 24-bit RGB) or medium mode (8-bit, 4-tap)	Up to 200 MB/s	Does not convert to frame grabber.

* Note that current products have "DVa" in the name, while legacy products have "DV" (with no "a") in the name.

Figure 1 and Figure 2 show diagrams of the CLS products.

Figure 1. Current ("DVa") product: PCIe8 DVa CLS

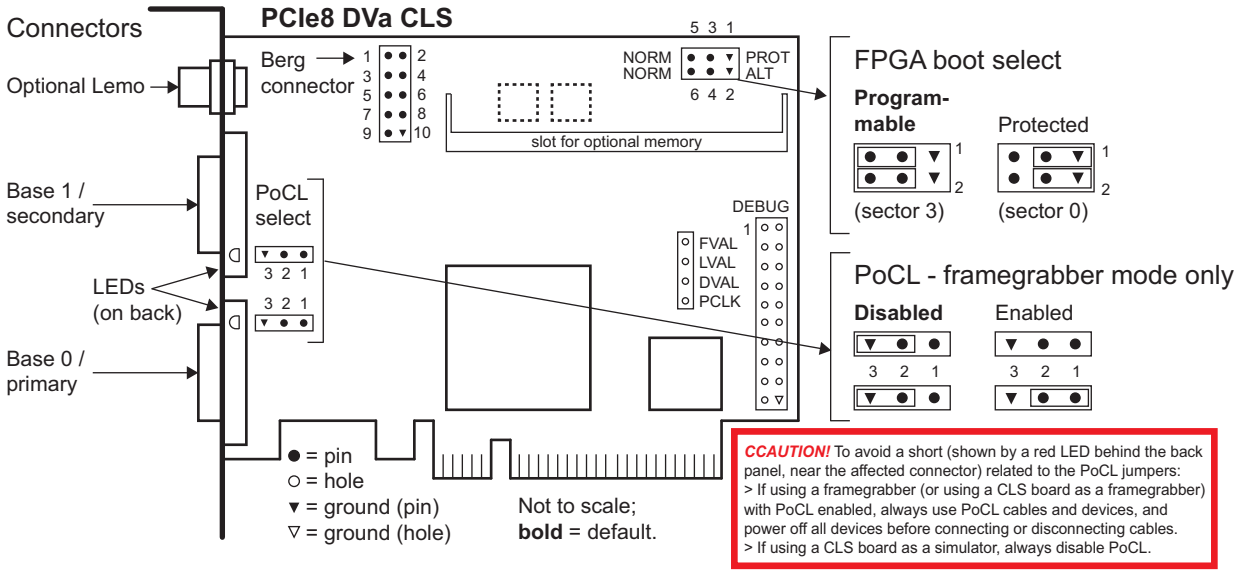
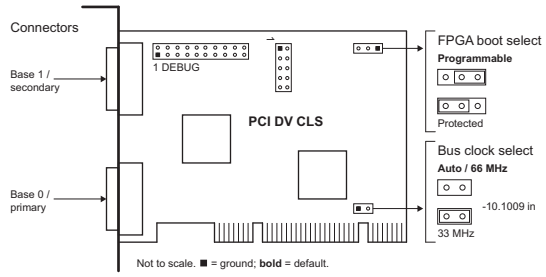
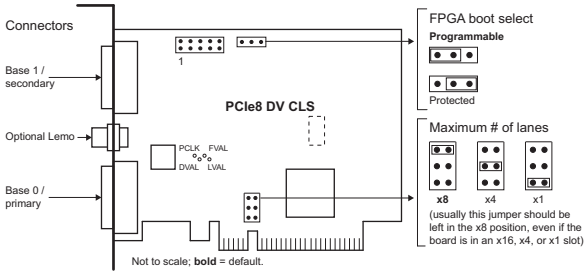


Figure 2. Legacy (“DV”) products: PCI DV CLS and PCIe8 DV CLS

PCI DV CLS



PCIe8 DV CLS



Requirements

EDT CLS products are high-speed DMA devices that require adequate bus bandwidth for reliable operation, especially depending upon such elements as the motherboard and chipset. In addition, different cameras run at different speeds, so bandwidth requirements will vary depending on which camera is being simulated. Therefore, you should choose and configure your system with bus bandwidth requirements in mind.

PCI Express (PCIe)

For PCIe, performance rates depend upon your system and its setup. Systems using EDT PCIe 8-lane vision products have tested at the maximum Camera Link specification of 850 MBytes.

EDT’s PCIe boards will not work in slots dedicated for graphics display, such as slots typically labeled “graphics” in some Dell system specifications. Consult your motherboard manufacturer to find out specifically which slots are available for general purpose I/O.

EDT’s PCIe 8-lane boards will fit in 1, 8, or 16-lane slots. In 8- or 16-lane slots the boards will work at their rated bandwidth. In 1-lane slots the bandwidth will be reduced to roughly one-eighth the maximum.

PCI

For PCI, the interface runs at 66 MHz and can sustain a maximum DMA bandwidth of approximately 200 MBytes per second. Use the PCI DV CLS in a slot that is 66 MHz or faster (you can use a 33 MHz slot, but the maximum sustainable DMA bandwidth will then drop to approximately 90–95 MBytes per second). If your application exceeds these limits, then data loss (broken images or timeouts) will occur.

The PCI DV CLS has a 32-bit PCI data bus, though you can use it in a 64-bit bus slot.

If your CLS board and your frame grabber reside in the same host and use the same PCI bus, then they share the bus bandwidth. If the total bandwidth requirements of both boards exceed the PCI bandwidth limits specified above, move either the CLS or the frame grabber so that they use different PCI buses.

NOTE If your image source is an internal counter, not actual images from the host computer (see Sending Simulated Data on page 18), the CLS is not using the PCI / PCIe bus for DMA and thus is not consuming bus bandwidth.

Related Resources

The resources below may be helpful or necessary for your applications.

EDT Resources

Description	*Detail*	*Web link*
• Product specifications (datasheets)	PCI DV CLS, PCIe8 DV CLS, PCIe8 DVa CLS	www.edt.com (search by product)
• Current (“DVa”) frame grabbers	For PCIe	www.edt.com/digital_video.html
• Legacy (“DV”) frame grabbers	For PCIe and PCI	www.edt.com/digital_video.html
• Extenders for frame grabbers	For fiber and coax	www.edt.com/digital_video.html
• Addendum: Firmware Guide	For EDT firmware	www.edt.com/manuals/PDV/dv-clink.pdf
• Addendum: Camera Configuration Guide	For cameras tested by EDT	www.edt.com/manuals/PDV/camconfig.pdf
• Application programming interface (API)	HTML & PDF	www.edt.com/manuals.html
• Installation packages: Windows, Linux, Mac OS X	Software / firmware	www.edt.com/software.html

Standards / Specifications

Description		*Web link*
• Camera Link	Camera Link standard	www.visiononline.org

Parts

Description	*Part number*	*Manufacturer*	*Web link*
Cabling	[multiple options]	[multiple options]	www.edt.com/cables

Installation

EDT provides installation packages for all supported operating systems (Windows, Linux, Mac OS X). These packages are provided on the EDT installation disk that ships with your EDT product.

However, to prevent installation package version issues, EDT recommends that you go to www.edt.com and do one of the following:

- For a new application, download the latest package.
- For an existing application, use the same package that was used to build it (from your own or EDT's archives), or recompile / relink the application with the latest installation package download.

NOTE Remove previous software releases before installing updates.

In either case, to find the installation package you need (either the latest one or an archived version), you can use the link under Related Resources on page 7.

To install the CLS:

1. Install the Pdv driver software as specified above.
2. Install the board into the host computer according to the computer manufacturer's instructions.
3. Connect the board to your frame grabber or other input device with a standard Camera Link cable. For part information, see the link under Related Resources on page 7. For connector pinouts, see Appendix A: Pin Assignments on page 26.

NOTE If you are using a single base-mode input device with a legacy "DV" CLS board (PCI DV CLS or PCIe8 DV CLS), the input device must connect to the CLS board's primary MDR26 connector (i.e., the one nearest the PCI / PCIe bus). With a current "DVa" CLS board, (PCIe8 DVa CLS), such a device can connect to either MDR26 connector. For details, see Units, Connectors, and Channels on page 13.

Included Files

The EDT PDV installation package includes files for all EDT vision products. Some files, like the device driver, are common to all products, while other files apply only to certain products. Files specific to the CLS include a CLS setup application, as well as C source and executables for several command-line example, diagnostic, and utility programs. These CLS-specific files are listed in the table below.

NOTE For Windows executables, the extension `.exe` is implied.

Table 2. CLS-specific files, part 1 of 2

File	Description
`clstest256.cfg`	Sample configuration file for demonstration, testing, or verification (via `simple_clsend` or `send_tiffs`). Located in `camera_config`.
`clsim.bit`	FPGA configuration file for the PCI version of the CLS. (On the PCIe version, the UI FPGA is embedded.)
`clsim_lib.c, clsim_lib.h`	C source and header file for `clsim_lib` – the portion of the PCI DV library (`pdvlib`) that covers CLS boards.
`clsiminit, clsiminit.c`	A command-line application (binary and source code) to set up the CLS for a specific output. For the list of arguments and options, enter... `clsiminit --help`
`clsimvar, clsimvar.c`	Utility to test line scan operation; includes option to vary the output line length. For usage, enter... `clsimvar --help`
`dvcl4*.bit`	FPGA configuration files for PCI CLS boards (see Firmware on page 22). Located in `flash/xc2s200`.

Table 2. CLS-specific files, part 2 of 2

`generic*cl.cfg`	Generic files (in `camera_config` directory) that you can copy and edit to configure the CLS for the camera you are simulating. Provide at least values for image height, width, and depth. For details, see Initializing the Board on page 15.
`imagelist.txt`	Sample image list for demonstration, testing, or verification (via `simple_clsend` or `send_tiffs`).
`pe8dvaclsim*.bit, pe8dvclsim*.bit`	FPGA configuration files for PCIe CLS boards (see Firmware on page 22). Located in `flash/xc5vlx30t`.
`send_tiffs, send_tiffs.c`	A command-line application (binary and source code) to send one or more TIF images to the frame grabber as simulated camera image data. For details on using this application, see Sending Simulated Data on page 18.
`simple_clsend, simple_clsend.c`	A command-line application (binary and source code) to send one or more image files to the frame grabber as simulated camera image data – similar to `send_tiffs` but with much of the rarely-used functionality stripped out. Currently it works for TIF image files only, but has stubs for other image formats to make it easier for users to add their own image formats (some of which will be filled in by EDT in future versions). Though not all that simple, `simple_clsend` is simpler as example code than `send_tiffs`. For details on using this application, see Sending Simulated Data on page 18.
`tiffs256/*.tif`	Sample images for demonstration, testing, or verification (via `simple_clsend` or `send_tiffs`).
`pdvlib.*`	Library binaries for the EDT digital imaging library, including `clsim_lib.c` functions; see `clsim_lib.c` (above) and consult the EDT API (see Related Resources on page 7]. Files installed on Windows have the extensions `.lib` and `.dll`; files installed on Linux have the extensions `.so` and `.a`; and files installed on Mac OS X have the extensions `.so` and `.dylib`.

Application Programming Interface (API)

EDT provides a common application programming interface (API) for all supported operating systems, so an application written for one EDT vision product will work with the others with minimal modification; any exceptions – such as differences between Windows, Linux, and Mac OS X – are noted in this guide.

The API includes three subset libraries:

- `clsim_lib` (`pdv_cls_` subroutines) – this library includes subroutines for EDT simulators only.
- `edtlib` (`edt_` subroutines) – this library includes subroutines (e.g., `pdv_configure_ring_buffers`, `edt_start_buffers`) that are used for configuring DMA and sending out data.
- `pdvlib` (`pdv_` subroutines) – this library includes subroutines that work with both simulators and frame grabbers (e.g., `pdv_open`, `pdv_close`), as well as subroutines that are for frame grabbers only.

To learn how these subroutines are used in practice, see the example applications. All of these resources are provided in your EDT installation package (see Related Resources on page 7).

Building or Rebuilding an Application

By default, EDT's installation package is saved in `C:\EDT\pdv` (Windows) or `/opt/EDTpdv` (Linux / Mac OS X). The package includes C source and executables for all EDT examples, utilities, and diagnostics.

NOTE Applications which access EDT boards must be compiled and linked to match the platform in use (32- or 64-bit). Applications linked with 32-bit EDT libraries will not run correctly on 64-bit systems, or vice versa. The EDT driver / software installation script detects whether the system is running 32- or 64-bit, and installs the appropriate files.

To build or rebuild programs, use an appropriate compiler and follow the steps below.

Windows (Visual Studio 6.0 or later)

1. From the installation directory `C:\EDT\pdv`, run *Pdv Utilities*.
2. Do any of the following:
   – To build or rebuild a specific application, first make sure your build environment variables are set properly. For example, enter...

     ```
     C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcvarsall.bat amd64
     ```

     Then enter...

     `nmake` *`file`*

     ...replacing *`file`* with the name of the example program you wish to build.
   – To build or rebuild and install all of the EDT examples, utilities, and diagnostics, navigate to the installation directory and enter...

     `nmake`
   – Alternatively, you can use the included Visual Studio 2008 project, or set up your own project in Visual C++.

Linux or Mac OS X

1. In a terminal window, navigate to the installation directory `/opt/EDTpdv`.
2. Do any of the following:
   – To build or rebuild a specific application, enter...

     `make` *`file`*

     ...replacing *`file`* with the name of the example program you wish to build.
   – To build or rebuild and install all of the EDT examples, utilities, and diagnostics, navigate to the installation directory and enter...

     `make`

Getting Started

Your EDT installation package includes various resources to help you get started and verify that your simulator is working properly. It covers how to initialize your board, transfer image file data, and test serial communication.

For details on the below applications and examples, see the C source code and the documentation for each: this guide covers `clsiminit`, `simple_send`, and `pdvterm`, while the EDT user's guide for frame grabbers (see Related Resources on page 7) covers `take` and `initcam`.

For details on usage, you can also run...

appname `--help`

Simple Image Data Verification

The following example sequence is for an EDT frame grabber with an EDT simulator (enumerated by the system as 0 and 1 respectively in this example). If you are using a third-party frame grabber, initialize it as instructed in the manufacturer's documentation to capture 256 x 256 x 8 bits and view the results).

To begin, connect the frame grabber to channel 0 (the MDR26 connector closest to the motherboard).

For your setup, run `pciload` with no arguments to see how the system has enumerated your devices, and then adjust the `-u` and `-pdvN` arguments accordingly. .

Example sequence:

`initcam -u 0 -f camera_config/clstest256.cfg #`(to initialize the EDT frame grabber)

`clsiminit -u 1 -f camera_config/clstest256.cfg #`(to initialize the EDT simulator)

`simple_clsend -u 1 -m -l 0 -i imagelist.txt #`(to continuously send all images in the directory `clstest256`; note that the character after `-u` is the digit "1" and the character after `-m` is the letter "l")

`send_tiffs -u 1 -i imagelist.txt`

In another window:

`take -u 0 -N 4 -l 100 #`(capture 100 images with the EDT frame grabber and make sure no timeouts)

...or...

`pdvshow -pdv0 #`(use `pdvshow` to capture and view images from the EDT frame grabber)

Simple Serial Verification

To test serial communication, use the loopback option (`-l`) to `clsiminit,` which enables UART looping to echo back every character sent. Then use the `pdvterm` terminal emulator application. For example:

`clsiminit -u 1 -l -f camera_config/clstest256.cfg`

`pdvterm`

`> aabbccdd` (output should look like this if you type "abcd")

Operational Details

EDT's CLS boards provide the following key features:

- Field-programmable gate arrays (FPGAs), to implement DMA:
- A choice between two sources of image data – either internal counters, or DMA from the host computer;
- Serial data; and
- Triggering.

This section explains how these features are implemented.

Image Data Source

With the CLS board, your source for simulated image data can be internal counters on the board itself, or actual image data sent to the DMA buffers by the `send_tiffs` application. Alternatively, you can write your own application to send data to the DMA buffers.

If your source is internal counters, the CLS does not use the DMA engine to transfer data.

If your source is DMA, the DMA engine transfers the data as described below.

NOTE The transfer process below applies to both sources, except that if your source is internal counters, the references to DMA do not apply.

When each frame starts, the simulator does not begin sending the image until after it has collected 16 KB of DMA data into its FIFO from the host. (This number can be lowered to accommodate images of less than 16 KB.) Once started, the image data from the host streams through the simulator continuously until the end of the image; if the DMA cannot keep up, there is a FIFO underflow, and the image data transferred to the frame grabber is corrupted. In this case, if the DMA is not reinitialized between frames, subsequent frames may show skewed data. However, the timing of the pixel clock, line-valid, and frame-valid signals from the simulator is not affected by underflows.

At the end of the image, the simulator pauses for a period of time specified by the 0x4C–4E Vertical Count Maximum (the value in those registers, minus the active video time specified in the 0x4A–4B Vertical Active) before readying itself for the next image. Once ready, it waits for DMA data to arrive before starting again. To stop the simulator, the host need only stop the DMA transfer.

The PCIe version of the CLS can support full mode (up to 8 taps, 8 bits per pixel) as well as all of the medium modes (4 taps at 8 to 16 bits per pixel).

In some applications, the number of rasters in each image can vary. Before the frame is started, the raster count is written into 0x4A–4B Vertical Active by the host. When the frame is started, this information is transferred to a holding register. The rising edge of frame-valid can trigger an interrupt to the host, and the host then has an entire frame transfer time to respond to this interrupt and modify the registers for the next frame. Values in certain other registers (0x44 FillA, 0x46 FillB, 0x4C–4E Vertical Count Maximum, 0x58–59 Horizontal Read Valid Start, and 0x5A–5B Horizontal Read Valid End) are held in holding registers in the same way, allowing them also to be modifed for each frame.

Serial Data

In addition to the frame-valid interrupt, the CLS also implements interrupts associated with the universal asynchronous receiver and transmitter (UART). You can send and receive serial data to and from the UART using the `pdv_serial` library functions.

Triggering

The CLS can be configured to trigger each frame-valid signal from the frame grabber using the rising edge of the CC1 camera control line. The trigger is ignored unless all other conditions for start-of-frame are already met – such as the completion of any vertical blanking interval specified for the previous frame, and the collection of sufficient DMA data in the FIFO.

To emulate the EXSYNC input to Dalsa linescan cameras, you can configure the CLS to trigger each line-valid signal on either the rising or falling edge of the CC1 line.

For more details, consult the EDT user's guide for your frame grabber (see Related Resources on page 7).

Units, Connectors, and Channels

This section covers how to work with multiple units, connectors, and channels, which are defined as follows:

unit	EDT product (board)
connector	physical connector (for example, an MDR26 connector)
channel	DMA channel

A typical C-Link board has two MDR26 connectors.

In base mode, each device requires one connector on the EDT board, and each connector provides one DMA channel. Thus, in base mode, an EDT simulator with two connectors has two DMA channels.

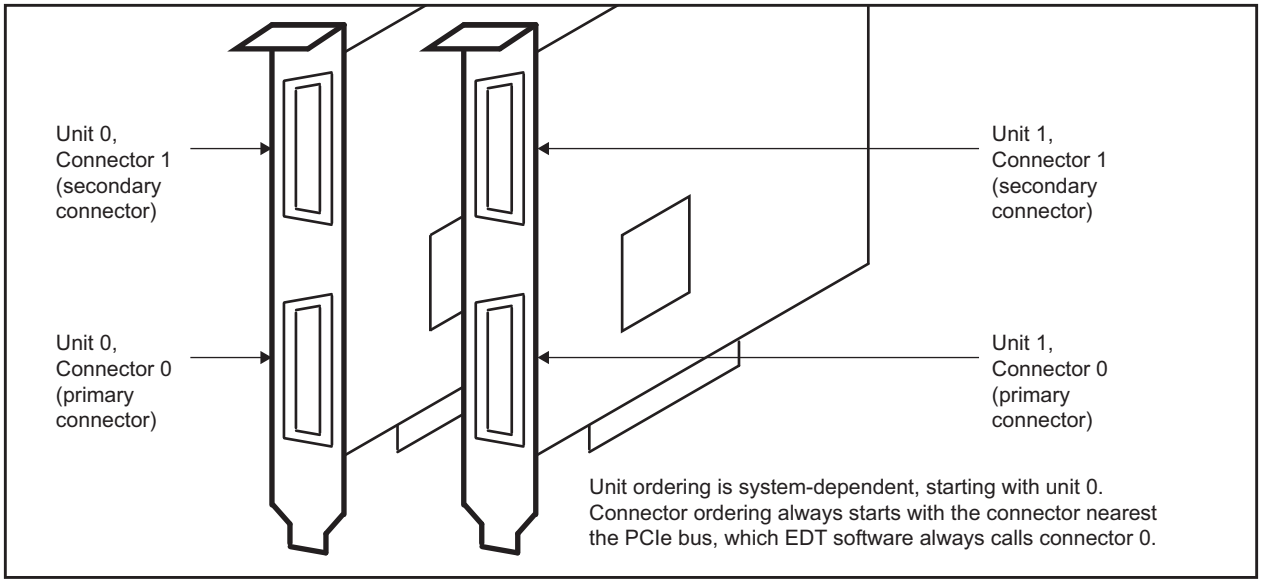
In medium or full mode, each device requires two connectors on the EDT board. In this case, the two connectors work together to support one DMA channel.

NOTE In EDT hardware, the connectors are labeled 0 and 1 on some boards, but 1 and 2 on others. Regardless of the labels on the hardware, the software always counts the connectors in order as 0 (primary), 1 (secondary), and so on, with 0 being the connector nearest the PCIe bus.

If you install one board in your host system, the system will assign the default unit number (0) to that board. If you install multiple boards, the system will assign a unique unit number to each board, starting with 0 (the sequence is system-dependent). Typically, the unit number is an argument when invoking an application (such as `clsiminit` or `simple_clsend`) or a parameter passed into one of the EDT subroutines.

The figure below shows an example of how the software enumerates the units (boards) and the connectors on each board. The default for the first unit number, connector number, and DMA channel number is always 0, with additional units, connectors, and DMA channels numbered as 1, 2, and so on.

Figure 3. Example of unit numbering and connector numbering



If you are using multiple EDT boards (units), or multiple connectors on a single EDT board, the software provides a unique handle to represent each unit and connector. Unless you specify something else, any application (either your own or EDT's) seeking access to the devices will default to unit 0, connector 0.

The way that you address the appropriate unit and DMA channel will depend on what you are doing.

- For EDT example and utility applications, use the arguments...

  `-u unit`

  `-c channel`

- For EDT API, use `pdv_open_channel(..., unit, channel)`. For each device, the open routine will return a pointer to the structure that represents the opened device (unit and DMA channel); this pointer appears in EDT examples and documentation as `pdv_p`. Each unit / channel combination can be opened and manipulated independently by passing the appropriate pointer to the library subroutines. For details, consult the EDT API (see Related Resources on page 7).

If you have multiple boards using the Pdv driver installed in your host computer, you must supply a unit number to specify which one you wish to initialize:

```
appname -u unit_number -c channel_number other_arguments
```

By default, EDT applications will address unit 0, connector 0 (the primary connector). If you are using multiple EDT boards (simulators, frame grabbers, or both), you can specify a particular board or connector by using the argument `-u` or `-c`, respectively.

For example, to initialize unit 1 (the second board) and connector 1 (the secondary connector), enter...

```
clsiminit -u 1 -c 1 camera_config/file
```

NOTE In your EDT installation package, running `pciload` with no arguments will display a list of all boards (by unit number) installed in the host, or in some cases to update or reprogram your board's firmware. For details, consult the user's guide for your frame grabber (see Related Resources on page 7).

Initializing the Board

As much as possible considering its different purpose, your CLS is designed to work like an EDT Camera Link frame grabber, and the software described here is designed to work in a manner similar to the `initcam` frame grabber initialization utility. EDT camera configuration directives and digital imaging library functions that can be used with a camera simulator are available for your own applications.

Using clsiminit

The `clsiminit` program can be used to initialize and configure the CLS.

For example...

```
C:\EDT\pdv> clsiminit -f camera_config/generic8cl.cfg
```

...sets up the CLS according to the directives specified in the camera configuration file `generic8cl.cfg`.

Camera configuration files are editable text files. To use `generic8cl.cfg`, first edit the file to specify at least the image height, width, and depth, according to the camera you wish to simulate. Alternatively, if you wish to simulate a camera for which a file already exists, you can use that file instead; the `clsiminit` application reads the same files as `initcam`. The files provide values for such parameters as image dimensions and number of taps. For details on `initcam` and camera configuration files, consult the Camera Configuration Guide and the EDT frame grabber user's guide (see Related Resources on page 7).

In addition to directives used to configure the software for specific cameras, the simulator accepts other directives that set its pixel clock speed, blanking interval, and other parameters (listed in Simulator-specific Configuration File Directives on page 17). You can set simulator-specific parameters via arguments to `clsiminit` or you can add these directives to any camera configuration file. Since `initcam` ignores any simulator-specific directives, you can use the same configuration file with `initcam` and `clsiminit`.

After you run `clsiminit,` the CLS is ready for output DMA at the specified size.

Just as `initcam` does for frame grabbers, `clsiminit` supports simulators by filling in the `PdvDependent` structure, which then is associated with the Pdv driver handle and persists across process calls. The values it sets are then set on the CLS board by calling the function `pdv_cls_set_dep` (defined in the library file `clsim_lib.c`). This function in turn calls a number of other functions to set individual registers on the board.

These same functions are available for you to modify for your own application; function prototypes are in `clsim_lib.h`.

For camera configuration, EDT provides two types of files: 1) camera-specific files for numerous camera models; and 2) generic `(generic*cl.cfg)` templates that you can use to create your own files if desired. Camera configuration files are described in the EDT camera configuration guide (see Related Resources on page 7). For simulator-specific details, see Simulator-specific Configuration File Directives on page 17.

To initialize the CLS with a specific camera configuration file, run `clsiminit` as follows...

```
clsiminit -f camera_config/filename.cfg
```

For example, to simulate the Basler Aviator 1000 camera, freerun at 8 bits, enter...

```
clsiminit -f camera_config/basler_ava1000-120km_8f.cfg
```

The CLS then will simulate the specified camera, according to the directives in the file.

Options for clsiminit

For a complete list of command line options, enter:

```
clsiminit --help
```

For example, you can specify such options as unit number, image list file, or number of buffers, as below.

`-u` *`unit`*	The unit number of the CLS (by default, 0).
`-B`	Do not load the FPGA configuration file.
`-q`	Disables program output (quiet mode).
`-s`	Sets the CLS so that data comes from the internal counter instead of DMA. Data comes from a simple counter that generates 16-bit pixels that start black and grow progressively lighter until they reach white.
`-C`	Sets the first 16-bit word of every frame to the value of a hardware counter which increments by 1 every frame.
`-l`	Lowercase “l” – enables UART loopback; serial received by the simulator will be looped back out (echoed) to the device.
`-F` *`freq`*	Sets the pixel clock in MHz; takes a floating point argument (by default, 20.0).
`-t` *`taps`*	Sets the number of taps.
`-f` *`config_file`*	Use the specified camera configuration file to set image parameters.
`-w` *`width`*	Image width (by default, 1024). Overridden by values specified in a camera configuration file specified with `-f`.
`-h` *`height`*	Image height (by default, 1024). Overridden by values specified in a camera configuration file specified with `-f`.
`-d` *`depth`*	Image depth (by default, 8). Overridden by values specified in a camera configuration file specified with `-f`.
`-v` *`vblank`*	Sets the interval between frames in lines (by default, 400).
`-V` *`VcntMax`*	Sets the total number of lines in the frame (ignored if *`vblank`* is specified instead with `-v`).
`-g` *`hblank`*	Sets the number of pixel clock cycles of horizontal blanking (by default, 300).
`-H` *`HcntMax`*	Sets the total number of pixel clock cycles per line, including blanking (ignored if *`hblank`* is specified instead with `-g`).
`-r`	Resets the board. Zeroes all settings.
`--help`	Prints complete and current list of all directives and arguments.

Simulator-specific Configuration File Directives

To specify simulator-specific directives, copy any camera configuration file and add the desired directives to your copy. Be sure also to edit the camera information directives (those starting with `camera_`) to provide a unique identifier, in case you later have to choose your configuration file from a list of several in an initialization application. (You can still use the same configuration file to configure an EDT frame grabber for a specific camera, as simulator-specific directives will be ignored.)

The first directive takes a floating point number:

`cls_pixel_clock`	pixel clock frequency, in MHz (20.0 – 85.0)

The next set are all single-bit directives that default to 0:

`cls_linescan`	1 enables linescan
`cls_lvcont`	1 enables continuous line valid
`cls_rven`	1 enables using the read-valid values
`cls_uartloop`	1 enables serial loopback
`cls_smallok`	1 enables small image sizes
`cls_intlven`	1 enables interleave (requires specifying the line_interleave directive, below)
`cls_firstfc`	1 puts the frame count in the first word of each frame
`cls_datacnt`	1 uses internal counters rather than DMA
`cls_dvskip`	number of clock cycles to skip between data-valid high
`cls_dvmode`	1 enables data-valid mode
`cls_led`	1 lights LED
`cls_trigsrc`	1 selects CC2 as the trigger source (0 selects CC1)
`cls_trigpol`	1 selects falling edge for trigger polarity
`cls_trigframe`	1 enables frame-valid triggering
`cls_trigline`	1 enables line-valid triggering
`cls_filla`	byte value to use for left margin
`cls_fillb`	byte value to use for right margin

One of the next two should be set – but not both, as they are redundant. If `hgap` is set, its value is preserved even if the total clock cycles per line changes; thus, horizontal active time must change instead.

`cls_hgap`	extra clock cycles per line to add to defined image width
`cls_hcntmax`	total clock cycles per line

One of the next two should be set – but not both, as they are redundant. If `vgap` is set, its value is preserved even if the total lines per frame changes; thus, vertical active time must change instead.

`cls_vgap`	lines between active video
`cls_vcntmax`	total lines per frame

The next values are all in pixel clock cycles. If they are not set, they default to a start value of 0 and end value equal to `width`:

`cls_hfvstart`	where frame valid starts on first line
`cls_hfvend`	where frame valid ends on last line

`cls_hlvstart`	where line valid starts relative to frame
`cls_hlvend`	where line valid ends
`cls_hrvstart`	where DMA data starts
`cls_hrvend`	where DMA data ends

To set the simulator interleave, set the `line_interleave` directive: a string in which the first value is the number of taps (as of this release, required to be four), followed by a start, delta pair for each tap. For example, the following line specifies four taps, the first one starting at pixel 0, the next at pixel 1024, the next at pixel 2048, and the final one at pixel 3072, with each tap incrementing by one:

```
line_interleave:"4 0 1 1024 1 2048 1 3072 1"
```

For details on how to simulate multi-tap cameras and how to use negative values to simulate cameras that implement taps moving from right to left, see Interleaving Registers on page 37.

NOTE With an EDT frame grabber, this same interleave value is then used for deinterleaving within `pdvshow`.

Sending Simulated Data

On a CLS board, simulated data can come from an internal counter, or from the host via DMA. In all cases, the data is output at the configured pixel clock speed.

Image Data From Internal Counter

Counter data comes from an internal counter that generates 32-bit pixel data: the 16 least significant bits start from 0x0000 and count up to 0xFFFF; the 16 most significant bits are an inverted version of these values. The count is cleared to zero at the start of each frame.

When functioning in base mode, the CLS transmits only the 24 last significant bits of these 32-bit data words to the frame grabber. Thus, the first word of each frame received by a base-mode frame grabber is 0xFF0000 and the second is 0xFE0001; for medium- or full-mode simulation, the first word of each frame is 0xFFFF0000 and the second is 0xFFFE0001. .

Channel 0 passes the 24 least significant bits; if functioning in medium- or full-mode, the eight most significant bits are transmitted on Camera Link port D out of channel 1.

This simulated data is useful for testing the cable, hardware, or DMA. Because data does not come from the host, no DMA transfer is involved.

Image Data From Host Via DMA – with simple_clsend

As an alternative to using an internal counter, you can send image data from the host via DMA.

The `simple_clsend` demonstration application shows how to use the EDT library to send images (up to 4096 pixels wide by 15,000 lines) through the CLS. The program reads a list of images and sends each one through the simulator. Although `simple_clsend` supports only TIF images, the EDT source code includes stubs for other image formats, some of which will be populated in future versions. You can use these stubs to modify the application to support other file formats as desired.

Before running `simple_clsend`, you must first initialize the CLS using `clsiminit` or `clsim`. For details, see Initializing the Board on page 15.

The simplest method of running `simple_clsend` is to run it with no arguments:

```
C:\EDT\pdv> simple_clsend
```

The `simple_clsend` application looks in the file `imagelist.txt` (by default, in the current directory) and reads in a list of image filenames, one per line, to send through the CLS. The application recognizes only the filename (which is the first word on each line, unless it is a comment line) and ignores everything else, including comments (which begin with `#`) and the tags designed for use by `send_tiffs`, described in send_tiffs image list format on page 20. The application then opens the default unit 0 board (which it expects to be the CLS), and then loops through the list of images one at a time, sending each image to that board.

NOTE In your EDT installation package, the application `pciload` can be used to display a list of all boards (with unit numbers) installed in the host, or in some cases to update or reprogram your board's firmware. For details, consult the EDT user's guide for your frame grabber (see Related Resources on page 7).

You can specify options such as unit number, image list file, or number of buffers. For a complete list of command line options, enter...

```
simple_clsend --help
```

`-u` *`unit`*	The unit number of the CLS (by default 0).
`-N` *`numbufs`*	The number of ring buffers to use. We recommend using four.
`-i` *`file`*	The input file with list of images and directives — by default, `imagelist.txt` in the current directory, but this option allows you to specify another file.
`-l` *`loops`*	The number of times to loop through the image list (0 to loop indefinitely).

For example, to instruct the program to use ten DMA buffers, enter:

```
simple_clsend -N 10
```

To instruct the program to look in the file named `list1.txt` for the list of images:

```
simple_clsend -i list1.txt
```

For example, to loop through the `imagelist.txt` twenty times, enter:

```
simple_clsend -l 20
```

Image Data From Host Via DMA – with send_tiffs

The `send_tiffs` demonstration application shows how to use the EDT library to send images (up to 4096 pixels wide by 15,000 lines) through the CLS. The program reads a list of images and sends each one through the simulator.

NOTE Unlike `simple_clsend`, `send_tiffs` has no stubs to support image formats besides TIF, and it includes functionality that is rarely used. Therefore, EDT recommends using `simple_clsend` instead whenever possible, and using `send_tiffs` only if you have a specific reason to do so.

Before running `send_tiffs`, you must first initialize the CLS using `clsiminit` or `clsim`. For details, see Initializing the Board on page 15.

The simplest method of running `send_tiffs` is to run it with no arguments:

```
C:\EDT\pdv> send_tiffs
```

The `send_tiffs` application looks in the file `imagelist.txt` (by default, in the current directory) and reads in a list of TIF images to send through the CLS. The format of the list is given below in detail (see send_tiffs image list format on page 20), but it can be a simple list of filenames, one per line.

The application then opens the default unit 0 board (expected to be the CLS), and then loops through the list of images one at a time, sending each image to that board.

NOTE In your EDT installation package, the application `pciload` can be used to display a list of all boards (with unit numbers) installed in the host, or in some cases to update or reprogram your board's firmware. For details, consult the EDT user's guide for your frame grabber (see Related Resources on page 7).

For a complete list of command line options, enter...

```
send_tiffs --help
```

Specific options such as unit number, image list file, or number of buffers are shown below.

`-u` *`unit`*	The unit number of the CLS (by default, 0).
`-N` *`numbufs`*	The number of ring buffers to use. We recommend using four.
`-i` *`file`*	The input file with list of images and directives — by default, `imagelist.txt` in the current directory, but this option allows you to specify another file.
`-t` *`images`*	The number of images to send – used to send only some of the images in the image list (cannot be greater than the number of images in the image list).
`-A` *`FillA`*	Set the left fill value to FillA.
`-B` *`FillB`*	Set the right fill value to FillB.

For example, to instruct the program to use ten DMA buffers, enter:

```
send_tiffs -N 10
```

To instruct the program to look in the file named `list1.txt` for the list of images:

```
send_tiffs -i list1.txt
```

And, for example, if `imagelist.txt` lists 2000 images, of which you wish to send only the first twenty, enter:

```
send_tiffs -t 20
```

send_tiffs image list format

Each line of the image list file contains either a comment, or the name of an image file (followed, optionally, by certain information about that file). A sample file at the end of this section illustrates the format.

Comments begin with `#`, so whenever `send_tiffs` sees that symbol, it ignores the rest of the line. Otherwise, the first string of characters up to a space character specifies the name of a TIF image. The filename is the only required information, but other values also can be specified in the following manner:

- Values are always numeric, specified in either decimal or hexidecimal. To use hexidecimal, precede the number with the string `0x`.
- The value must follow immediately after the directive, with no space in between.
- Directive names are not case-sensitive; `FillA` is treated the same as `filla`.
- For any image file listed, values not specified are taken from the last line on which they were specified or (if they were specified nowhere) from the program defaults as described for each directive.

`FillA:`*`value`*

Sets the fill value for the left margin of the image. A value of –1 instructs the program to use the pixel value found in the first pixel of the first line in the image as the margin value.

`FillB:value`

Sets the fill value for the right margin of the image. A value of –1 instructs the program to use the pixel value found in the last pixel of the first line in the image as the margin value.

`hStart:value`

Specifies where in the frame to place the pixels from the image file. For example, the line `image.tif hStart:600` places the first pixel of `image.tif` 600 pixels into the camera's image frame.

If `hStart` is not specified, the default behavior centers the image file within the camera's image. That behavior can be specified manually by setting `hStart` to –1. (To change this default, search for `DEFAULT_HSTART` in the `send_tiffs.c` source code and edit the value as required.)

`vgap:value`

Specifies how much vertical gap to leave between the current image and the next image. If left unspecified, the default value is 400 clock cycles. (To change this default, search for `DEFAULT_VGAP` in the `send_tiffs.c` source code and edit the value as required.)

For example, a file such as the following could be used to test different values for `vgap`...

```
image1.tiff filla:0xff fillB:0xff vgap:400
image2.tiff filla:0xff fillB:0xff vgap:300
image3.tiff fillA:0x1f fillb:255 vgap:100 hStart:600
image4.tiff
```

Given such a file, the CLS will send `image4.tiff` with the `fillA, fillB, vgap` and `hstart` values used for `image3.tiff` in the previous line.

send_tiffs code overview

The application `send_tiffs` uses functions defined in `clsim_lib.c` and `clsim_lib.h`; for details, see those two files, as well as the EDT API (see Related Resources on page 7).

First, `send_tiffs` parses the command line arguments, gets the list of images, and performs some initial setup, such as creating the EDT ring buffers and getting the simulated camera's size (as set by `clsiminit`).

Next, `send_tiffs` preloads all but the last ring buffer with the first images from the image list.

The main loop of the program has only four things to do:

1. Start DMA to send the image data to the simulator.
2. Set up the simulator for the next image to send.
3. Get another image from the list and load it into the buffer that just finished being sent to the simulator.
4. Wait for the image that started sending at step 1.

The settings for an image, such as width and height, can be set on the simulator any time before the simulator begins sending that image — that is, any time before the simulator sets frame-valid high.

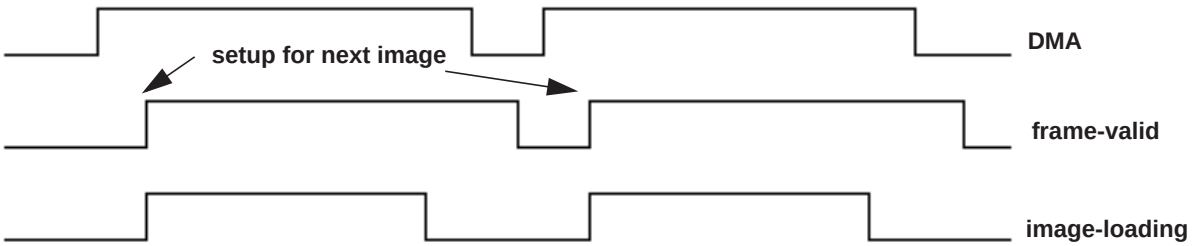
The setup required in step 2 uses the EDT library Event system to ensure that an image is never missed:

```
edt_set_event_func(pdv_p,
        EDT_PDV_EVENT_FVAL,
        (EdtEventFunc)setup_clsim_event,
        &cbinfo, 1);
```

This code registers the function `setup_clsim_event,` which is then called when the driver receives an interrupt notifying it that frame-valid went high. Therefore, as soon as frame valid goes high, it is possible to set up the simulator for the next image, and that is what `setup_clsim_event` does.

Figure 4 shows the relative timing of DMA, frame-valid, and image-loading. As indicated by the arrows, setup for the next image occurs on the rising edge of the frame-valid signal.

Figure 4. Timing of DMA, image setup, and image loading



DMA starts with the program's call to `edt_start_buffers(pdv_p, 1).` After the CLS has 16 KB of data, it starts sending data to the frame grabber. At the same time it sends the FVAL interrupt to the driver, which in turn calls `setup_clsim_event.` Immediately after the program calls `edt_start_buffers,` it loads the next image into the buffer most recently sent (so the buffer being loaded is just behind the buffer being sent).

NOTE To ensure maximum speed, image loading must take less time than the DMA transfer.

Firmware

The CLS boards have FPGAs containing two types of logic: PCI / PCIe interface logic, and image data logic.

On the PCI version, the two types of logic reside in two FPGAs: a PCI FPGA for PCI interface logic, and a user interface (UI) FPGA for image data logic. The first loads automatically at power-on; the second must be loaded after power-on, using the automatic initialization application `clsiminit`.

On the PCIe version, both types of logic reside in one FPGA, which loads automatically at power-on.

NOTE Logically, the single FPGA on the PCIe version still operates as two FPGAs (a PCIe FPGA for the PCIe interface logic and a UI FPGA for the image data logic), just like the two FPGAs on the PCI version.

At times, you may need to reprogram the PCI / PCIe interface flash memory using `pciload` – for instance:

- if you want to convert an EDT simulator into a frame grabber, or vice versa;
- if you want to switch from one Camera Link mode to another;
- if you need to use an FPGA configuration file that has special functionality;
- if you upgrade to a new installation package that includes a required update for your board; or
- if the firmware becomes corrupted.

To do so, follow the instructions in the sections below.

NOTE For details on using the CLS as a frame grabber, consult the EDT user's guide for current Camera Link PCIe frame grabbers (see Related Resources on page 7).

Checking and Loading the Firmware

The following procedure applies to standard firmware only. If you are running a custom firmware file and need to update it, first get a custom firmware configuation file from EDT.

NOTE Do not upgrade the firmware simply because you see a newer firmware file with a new driver; instead, consult the `CHANGELOG_PDV.txt` file in the package, which will tell you if there is a necessary upgrade.

- For Windows, `pciload` is an application in `\EDT\Pdv`. Double-click the *Pdv Utilities* icon to bring up a command shell in the installation directory `\EDT\Pdv`.
- For Linux or Mac OS X, `pciload` is an application in `/opt/EDTpdv`.

To see currently installed and recognized EDT boards and firmware, enter...

```
pciload
```

The program outputs the date and revision number of the firmware in the flash memory.

To compare the PCI / PCIe firmware in the package to the firmware loaded in flash on the board, enter...

```
pciload verify
```

If the two match, the firmware on the board is the same as the firmware in the installation package. If they differ, you'll see error messages, but these do not necessarily indicate problems; if your application is working correctly, you probably do not need to upgrade the firmware.

If you do wish to update the standard firmware, enter...

```
pciload update
```

To upgrade or switch to a specific firmware file, enter...

```
pciload firmware
```

...replacing *`firmware`* with the filename of the desired firmware, up to but not including the `.bit` extension.

Example: To load simulator firmware for base or medium mode on a PCIe8 DVa CLS board, enter...

```
pciload pe8dvaclsim
```

The board reloads the firmware from the flash memory only during power-on – so after running `pciload`, the old firmware is in the PCIe FPGA until the system has power-cycled.

NOTE After loading the firmware, you must do a power cycle (not just a reboot) to complete the update.

For a list of all `pciload` options, see the tables below or enter:

```
pciload --help
```

Table 3 and Table 4 show the FPGA configuration files for the CLS boards. For details on verifying and re-flashing your board, consult the Firmware section of the EDT user's guide for current Camera Link PCIe frame grabbers (see Related Resources on page 7).

NOTE Firmware files are not interchangeable. For current boards (with "DVa" in their names), use only Table 3; for legacy boards (with "DV" but no "a" in their names), use only Table 4.

Also, EDT recommends using firmware not ending in `_fm` (for full mode) whenever possible; but if you are switching just one board between full and other modes, you can use the `_fm` firmware with one base- or medium-mode camera of 40 MHz or more.

Table 3. Current ("DVa") board: Arguments to pciload

To use this board...	...for these modes...	...use this command...
PCIe8 DVa CLS as simulator*	Base, dual base	`pciload pe8dvaclsim`
	Base, medium, full	`pciload pe8dvaclsim_fm`
PCIe8 DVa CLS as frame grabber	Base, dual base, medium	`pciload pe8dvacamlk`
	Base, medium, full	`pciload pe8dvacamlk_fm`
* When flashed as a frame grabber, the PCIe8 **DVa** CLS simulator becomes a fully functional PCIe8 **DVa** C-Link frame grabber. For details, consult the appropriate EDT user's guide (see Related Resources on page 7).		

Table 4. Legacy (“DV”) boards: Arguments to pciload

To use this board...	...for these modes...	...use this command...
PCIe8 DV CLS as simulator*	Base, dual base	`pciload pe8dvclsim`
	Base, medium, full	`pciload pe8dvclsim_fm`
PCIe8 DV CLS as frame grabber	Base, dual base, medium	`pciload pe8dvcamlk2`
	Base, medium, full	`pciload pe8dvcamlk2_fm`
PCI DV CLS as simulator only	Base (either 24-bit RGB; or 8-bit, 1- or 2- tap) or medium (8-bit, 4-tap)	`pciload dvcl4`
* When flashed as a frame grabber, the PCIe8 **DV** CLS simulator becomes a fully functional PCIe8 **DV** C-Link frame grabber. For details, consult the appropriate EDT user’s guide (see Related Resources on page 7).		

Corrupted Firmware

In rare cases, the board firmware may become corrupted. Typically, the symptom is that the board is not recognized by the operating system, or the computer itself will not boot with the board in it. In such cases, booting from the protected (backup) sector will allow the board to be seen so that you can reprogram the programmable sector.

Each EDT frame grabber has a protected (backup) and a programmable flash memory boot sector. The sector from which the board will boot is determined by a jumper, which is preset to the programmable sector. If that sector becomes corrupted, you can move the jumper so the board will boot from the protected sector.

Most often, firmware corruption is the result of an interrupted load process or an unanticipated interaction with the host computer; if so, following the procedure below should solve the problem.

If the firmware file itself has become corrupted, first contact EDT for the current firmware you’ll need to replace it, and then follow this procedure.

To reboot from the protected sector:

1. If needed, move the new firmware file to the directory in which you installed the EDT software.
2. Power off the host and board.
3. To avoid later confusion, remove any other EDT boards from the host.
4. On the EDT board with the corrupted firmware, move the jumper from its programmable to its protected setting (to locate this setting on your board, see Firmware on page 22).
5. Power up the host and board.
6. Navigate to the directory in which you installed the EDT software.
7. At the command prompt, enter...

   `pciload`

   The program outputs the date and revision number of the firmware in the flash memory — in this case, the date and revision number that shipped as of your purchase date. If no errors are reported, you have successfully booted from the protected sector.
8. With the system still powered on, move the jumper back to its original position.
9. Enter...

   `pciload` *`firmware`*

   ...replacing *`firmware`* with the correct filename, as indicated in Table 3 and Table 4 (above). If the feedback shows no errors, the new firmware has been installed, although it is not yet running.

10. Power off the host and board again.
11. Power on the system.
12. Run `pciload` with no arguments to verify the board is recognized and is running with the new firmware.

Appendix A: Pin Assignments

This section shows the pin assignments for the Camera Link connectors and the debug connector.

Camera Link (MDR26) Connectors

The Camera Link (MDR26) connector pin assignments for various operating modes are shown below.

Camera or simulator end	Frame grabber end	Camera Link signal (base mode, primary connector)	Camera Link signal (medium mode, secondary connector)	Camera Link signal (full mode, secondary connector)
1*	1*	inner shield / ground*	inner shield / ground*	inner shield / ground*
14*	14*	inner shield / ground*	inner shield / ground*	inner shield / ground*
2	25	X0–	Y0–	Y0–
15	12	X0+	Y0+	Y0+
3	24	X1–	Y1–	Y1–
16	11	X1+	Y1+	Y1+
4	23	X2–	Y2–	Y2–
17	10	X2+	Y2+	Y2+
5	22	Xclk–	Yclk–	Yclk–
18	9	Xclk+	Yclk+	Yclk+
6	21	X3–	Y3–	Y3–
19	8	X3+	Y3+	Y3+
7	20	SerTC+	unused	100 ohms
20	7	SerTC–	unused	terminated
8	19	SerTFG–	unused	Z0–
21	6	SerTFG+	unused	Z0+
9	18	CC1–	unused	Z1–
22	5	CC1+	unused	Z1+
10	17	CC2+	unused	Z2–
23	4	CC2–	unused	Z2+
11	16	CC3–	unused	Zclk–
24	3	CC3+	unused	Zclk+
12	15	CC4+	unused	Z3–
25	2	CC4–	unused	Z3+
13*	13*	inner shield / ground*	inner shield / ground*	inner shield / ground*
26*	26*	inner shield / ground*	inner shield / ground*	inner shield / ground*

* For PoCL, pins 1 and 26 change to +12V DC power, while pins 13 and 14 change to +12V DC power return.
For details on using PoCL, consult the EDT user's guide for Camera Link PCIe frame grabbers (see Related Resources on page 7).

Debug Connector

On the PCI version of the CLS board, near the upper edge, there is a 2- by 10-pin connector labeled **DEBUG**, with test points soldered into it.

The pin assignments and test points are shown below.

PINS:

2	4	6	8	10	12	14	16	18	20
O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O
1	3	5	7	9	11	13	15	17	19

TEST POINTS:

Pin	Signal	Notes
1	ground	1–6 are driven as 2.5-volt CMOS signals directly from the UI FPGA.
2	ground	
3	line valid	
4	frame valid	
5	data valid	
6	pixel clock	
7–20	[undefined]	

The PCIe version of the board has the same DEBUG connector, but it is fully configurable. In addition, near the center of the PCIe board there are four assigned debugging pins labeled PCLK (pixel clock), FVAL (frame valid), DVAL (data valid), and LVAL (line valid), as shown in Figure 2 on page 6.

Appendix B: Registers

Your EDT installation package provides files for accessing the registers described below. The EDT driver on the host computer uses the registers implemented in the user interface (UI) FPGA to configure the simulated data.

The FPGA configuration file `clsim.bit` in the PCI version of the CLS (or the embedded equivalent in the PCIe version) defines these types of registers:

- Camera Link registers;
- simulator registers;
- timing registers; and
- interleaving registers.

In the descriptions below, a bit is *set* when its value is one, and *clear* when its value is zero. Unused bits are undefined when read; if your application writes to them, write them as zero.

Camera Link Registers

0x00 Command

Access / Notes: 8-bit, write-only / PDV_CMD

Always reads 0x02.

Bit	Name	Description
7–5	[no name]	Not used.
4	CLRFVINT	Clear FVINTSTAT in 0x0B Serial Data Status. Acts on one write and need not be cleared.
3–1	[no name]	Not used.
0	RESET_INTFC	Reset frame-valid interrupt. Acts on one write and need not be cleared.

0x01 Status

Access / Notes: 8-bit, read-only / PDV_STAT

Bit	Name	Description
7–5	[no name]	Not used.
4	FIFO_NOTEMPTY	Set when FIFO contains data from the DMA stream.
3–2	[no name]	Not used.
1	FRAME_VALID	Set when the simulator asserts frame-valid to the frame grabber.
0	UNDERRUN	Set if the FIFO has run out of DMA data.

0x02 Configuration

Access / Notes: 8-bit, read-write / PDV_CFG

Bit	Name	Description
7–4	[no name]	Not used.
3	FIFO_RESET	Set to clear the simulator (FIFO, counters, all state bits and interrupts). Remains set until explicitly cleared by the host.
2–0	[no name]	Not used.

0x05 [Reserved]

Access / Notes: This register address is reserved, and always returns zero.

0x07 Mode Control

Access / Notes: 8-bit, read-only / PDV_MODE_CNTRL

Bit	Name	Description
7–4	[no name]	Not used.
3–0	CC[4–1]	State of the four camera control lines from the frame grabber.

0x0A Serial Data

Access / Notes: 8-bit, read-write / PDV_SERIAL_DATA

Bit	Name	Description
7–0	[no name]	When written, the byte is serialized and sent out to the frame grabber using the SERTFG UART signal. Data from the frame grabber is deserialized and available for reading here, using the SERTCAM UART signal.

0x0B Serial Data Status

Access / Notes: 8-bit, read-only / PDV_SERIAL_DATA_STAT

Handles UART signals and supports an interrupt on the rising edge of the frame-valid signal.

Bit	Name	Description
7	INTFC_INT	Set when EN_GLOB_IN (in 0x0C Serial Data Control) is set and: • FVINT is set, or • TRANSMIT_RDY and EN_TX_INT (in 0x0C Serial Data Control) are both set, or • RECEIVE_RDY and EN_RX_INT (in 0x0C Serial Data Control) are both set.
6	FVINTSTAT	Set when the rising edge of a frame-valid is detected. Cleared by CLRFVINT or RESET_INTFC in the Camera Link Registers registers (0x00 – 0x28).

5	[no name]	Not used.
4	FVINT	Set when FVINTSTAT and ENFVINT (in 0x10 Utility 2) are both set.
3–2	[no name]	Not used.
1	TRANSMIT_RDY	Set when UART transmitter is holding the register ready for the next byte to be written to it.
0	RECEIVE_RDY	Set when UART receiver has a character available for reading.

0x0C Serial Data Control

Access / Notes: 8-bit, read-write / PDV_SERIAL_DATA_CNTL

Handles UART signals and global interrupt enable.

Bit	Name	Description
7–6	BAUD[1–0]	If the value in 0x24 Baud Rate is zero, sets the baud rate: 00 9600 01 19,200 10 38,400 11 115,200
5	CL_RECEIVE_RDY	Set to clear RECEIVE_RDY in 0x0B Serial Data Status.
4	EN_GLOB_INT	Global interrupt enable. Set to enable any interrupt; when clear, interrupt bits have no effect.
3	EN_TX_INT	Set to enable interrupt on UART transmit buffer empty.
2	EN_RX_INT	Set to enable interrupt on UART receive buffer full.
1	EN_TX	Set to enable the UART transmitter.
0	EN_RX	Set to enable the UART receiver.

0x0F Utility

Access / Notes: 8-bit, read-write / PDV_UTILITY

Accommodates a big-endian host. The PCI / PCIe bus and EDT boards are little-endian.

Bit	Name	Description
7–4	[no name]	Not used.
3	SSWAP	Swaps the position of the two 16-bit short words in one 32-bit data word, so that *short 2* is transferred before *short 1*. Does not change the order of the bits within each short. See Figure 5 for details of the data word structure.
2–1	[no name]	Not used.
0	BSWAP	Swaps the position of bytes 0 and 1, and also bytes 3 and 4, in a 32-bit data word, so that the bytes are positioned 1, 0, 3, 2. Does not change the position of the bits within each byte. Figure 5 shows the structure of a 32-bit data word, with no swapping. With SSWAP set, short 0 appears before short 1. With BSWAP set, byte 2 appears before byte 3, and byte 0 before byte 1. With both set, byte 0 appears first, followed by byte 1, byte 2, and finally byte 3.

Figure 5. Data Word Structure Without Swapping

short 0																short 1															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
byte 0								byte 1								byte 2								byte 3							

0x10 Utility 2

Access / Notes: 8-bit, read-write / PDV_UTIL2

Bit	Name	Description
7–4	[no name]	Not used.
3	ENFVINT	Set to enable the frame-valid interrupt.
2–0	[no name]	Not used.

0x20 PLL Programming

Access / Notes: 8-bit, read-write / PDV_PLL_CTL

Used to load the MPC9230 PLL clock generator to create a 3.5x pixel clock signal that can match all pixel clocks evenly divisible by 0.0625 between 20 and 32 MHz, all frequencies divisible by 0.125 between 32 and 64 MHz, and all frequencies divisible by 0.250 between 64 and 85 MHz.

Bit	Name	Description
7	PLL_CLOCK	Connected to the PLL serial clock input.
6	PLL_DATA	Connected to the PLL serial data input.
5–1	[no name]	Not used.
0	PLL_STROBE	Connected to the strobe input of the PLL.

0x24 Baud Rate

Access / Notes: 8-bit, read-write / PDV_BRATE

Bit	Name	Description
7–0	BAUD_RATE	Sets the baud rate. If this register is clear, then use BAUD[1–0] from 0x0C Serial Data Control. The baud rate is computed from this value as follows: `(20MHz/(baud_rate*16))-2` For example, 0x80 = 9600 baud.

0x28 Camera Link Data Path

Access / Notes: 8-bit, read-write / PDV_CL_DATA_PATH

A value of 0x37 configures the CLS for a 4-tap 8-bit camera.

Bit	Name	Description
7	[no name]	Not used.
6–4	TAPS	Number of taps, minus 1 (0x0, 0x1, or 0x3 for one, two, or four taps, respectively).
3–0	BITS	Number of bits per tap, minus 1 (0x07 for eight bits per tap).

Simulator Registers

0x40 Camera Link Configuration A

Access / Notes: 8-bit, read-write / PDV_CLSIM_CFGA

Bit	Name	Description
7	LINESCAN	When set, once the start-of-frame conditions are met, the simulator runs forever, emulating a linescan camera (as if 0x4A–4B Vertical Active and 0x4C–4E Vertical Count Maximum were set to infinity). When clear, frame-valid determines which rasters have active data.
6	LVCONT	When set, line-valid is asserted continuously with its normal timing, even during the vertical blanking interval between frames. When clear, line-valid remains low during vertical blanking.
5	RVEN	When set, the start and end margins of each raster are filled with the values from 0x44 FillA and 0x46 FillB respectively, and the positions of the margins are determined by 0x58–59 Horizontal Read Valid Start and 0x5A–5B Horizontal Read Valid End. When clear, the entire raster is filled with valid data.
4	UARTLOOP	When set, serial data emitted by the frame grabber is echoed back unchanged, enabling a test of the frame grabber's serial port. When clear, your camera simulator application could respond to serial commands over the UART.
3	SMALLOK	When set, simulator starts DMA when 1 KB of data is in FIFO, allowing the simulator to handle images smaller than 16 KB. When clear, simulator waits until 16 KB of data is in FIFO before starting DMA.
2	INTLVEN	When set, enables four-tap interleaving — the four-tap reordering of 8-bit pixel values — using the registers 0x60–61 Tap 0 Start through 0x6E– 6F Tap 3 Delta. Image data destined for the frame grabber is first passed through an interleaving mechanism to duplicate the data ordering that some cameras exhibit. When interleaving is enabled, rasters are restricted to a maximum of 4096 eight-bit pixels of active image data (DMA plus fill). When clear, interleaving is off.
1	FIRSTFC	When set, the first word of the frame is the frame count: a 16-bit flag of 0x3333 in the most significant bits and a 16-bit framecount in the least significant bits. It replaces the first 32-bit word of DMA data for each frame, after any interleaving. When clear, the first word is DMA or counter data. The FIFO_RESET bit in 0x02 Configuration initializes the framecount to 0x0001.
0	DATACNT	When set, image data comes from the counters instead of the DMA stream. The simulated 32-bit data generated has a 16-bit count in the least significant bits; the 16 most significant bits are an inverted version of the least significant bits. The count is cleared to zero at the start of each frame. Thus, the first 32-bit word of each frame is 0xFFFF0000, and the second is 0xFFFE0001. The CLS treats this data as little-endian, so the fourth 8-bit pixel of the frame has a value of 0x01. When set, also setting SMALLOK stops the simulator at the start of the next frame, to enable getting a single frame of counter data. When clear, simulator uses DMA data.

0x41 Camera Link Configuration B

Access / Notes: 8-bit, read-write / PDV_CLSIM_CFGB

By default, all zero, resulting in a data-valid signal that is low during blanking and high during active video.

Bit	Name	Description
7–4	DVSKIP	Number of pixel clocks skipped per data-valid strobe. When DVSKIP=7, data-valid is asserted every eighth clock cycle. If 0x54–55 Horizontal Line Valid Start set to zero, the first data-valid strobe lines up with start of line-valid. When DVSKIP is zero, data valid is always high, except during blanking. DVSKIP does not effect the timing of line-valid, frame-valid, or image data sent to the frame grabber. Therefore, a DVSKIP value of 7 causes only one-eighth of the DMA data to be captured by the frame grabber as valid image data; the rest is ignored.
3–0	DV MODE	Camera manufacturers exhibit no consensus on how to treat the data-valid signal. DVMODE allows most implementations to be simulated: 00 = data-valid low during blanking. 01 = data-valid high during blanking. 10 = data-valid is treated the same during blanking as during active video (so DVSKIP also affects blanking). 11 = data-valid always low (may change in future revisions).

0x42 Camera Link Configuration C

Access / Notes: 8-bit, read-write / PDV_CLSIM_CFGC

Set TRIGLINE to emulate a Dalsa camera's EXSYNC trigger.

Bit	Name	Description
7	LED	When set, turns on LED (visible only when host computer case is open).
6–4	[no name]	Not used.
3	TRIGSRC	When set, selects camera control line 2 as trigger source. When clear, selects camera control line 1.
2	TRIGPOL	When set, triggers on falling edge of signal. When clear, triggers on rising edge.
1	TRIGFRAME	Set to enable frame-valid triggering — simulator waits at the start of each frame until a trigger is detected.
0	TRIGLINE	Set to enable line-valid triggering. Simulator waits at the start of each raster until a trigger is detected. A Dalsa linescan camera starts the next raster when it detects a rising edge on the CC1 line.

0x43 External Sync Delay

Access / Notes: 8-bit, read-write / PDV_CLSIM_EXSYNCDLY

Bit	Name	Description
7–0	[no name]	Sets the amount of delay, in pixel clock cycles, from the trigger source specified in the 0x42 Camera Link Configuration C, when used for line-valid triggering. To any delay, add an additional six pixel clock cycles of pipeline delay.

0x44 FillA

Access / Notes: 8-bit, read-write / PDV_CLSIM_FILLA

Bit	Name	Description
7–0	[no name]	Fills any margin at the start of each raster, as determined by 0x58–59 Horizontal Read Valid Start.

0x46 FillB

Access / Notes: 8-bit, read-write / PDV_CLSIM_FILLB

Bit	Name	Description
7–0	[no name]	Fills any margin at the end of each raster, as determined by 0x5A–5B Horizontal Read Valid End.

Timing Registers

A 16-bit binary counter — `hcnt` — establishes horizontal timing. It increments with each pixel clock cycle until it reaches the maximum value specified by 0x48–49 Horizontal Count Maximum, after which it clears to zero. Therefore, a value of 0x04FF in 0x48–49 Horizontal Count Maximum causes `hcnt` to count repeatedly from 0x0000 to 0x04ff, for a raster period of 1280 pixel clock cycles.

The other timing signals are turned on and off by comparing specific register values with `hcnt`. For example, when `hcnt` is equal to the 16-bit value in 0x54–55 Horizontal Line Valid Start, the line-valid strobe is turned on. It is turned off again when `hcnt` is equal to the 16-bit value in 0x56–57 Horizontal Line Valid End. Therefore, if 0x54–55 Horizontal Line Valid Start is set to 0x0000 and 0x56–57 Horizontal Line Valid End to 0x0400, then the line-valid signal is on for 1024 pixel clock cycles.

The registers 0x58–59 Horizontal Read Valid Start and 0x5A–5B Horizontal Read Valid End turn on an internal read-from-DMA signal. When read-from-DMA is true, image data to the frame grabber is taken from the DMA data stream. When it is false, we use the values from 0x44 FillA and 0x46 FillB instead, for the left and right margins respectively. If RVEN is false, read-from-dma follows line-valid.

NOTE Margins are constrained to be on a pixel clock boundaries, so the number of bytes per raster for images sent through the DMA channel must be evenly divisible by the number of 8-bit taps being simulated. We recommend that images sent to the simulator have rasters that are an even multiple of four pixels long.

Frame-valid starts on the first raster of the frame, then remains high until the final raster of the frame, as determined by 0x4A–4B Vertical Active. However, frame-valid can start at the beginning of the first raster, or some number of pixel clock cycles into the first raster; likewise, frame-valid can remain asserted until the very end of the last raster, or go low some number of pixel clock cycles before the very end of the last raster. The registers 0x50–51 Horizontal Frame Valid Start and 0x52–53 Horizontal Frame Valid End determine the position within the first and last raster that the frame-valid signal starts and ends, respectively.

When the simulator is configured and ready for data, and sufficient data (by default, 16 KB, or 1 KB if SMALLOK is set) is in the FIFO, the frame-valid signal is turned on within that raster at the position determined by 0x50–51 Horizontal Frame Valid Start. During that same raster, a 24-bit counter called `vcnt` is zeroed, and then increments at the end of each raster. When the 16 least significant bits of `vcnt` are equal to the value in 0x4A–4B Vertical Active, then frame-valid is turned off during that raster at the position determined by 0x52–53 Horizontal Frame Valid End. When `vcnt` is equal to the 24-bit value in 0x4C–4E Vertical Count Maximum, the simulator is ready to start the next frame on the following raster (assuming that sufficient DMA data is available). Thus, the value in 0x4C–4E Vertical Count Maximum plus one is the number of rasters spent on a given image, including active rasters and rasters spent in vertical blanking.

The value of 0x4C–4E Vertical Count Maximum must exceed or equal that of 0x4A–4B Vertical Active. If the values are equal, then consecutive frames are butted together with zero rasters spent in vertical blanking. In this case, the frame-valid signal is low for vertical blanking for only part of a raster.

By default, 0x54–55 Horizontal Line Valid Start is set to 0x0000. Thus, the line-valid signal is asserted at the earliest possible time. To simulate most cameras, 0x50–51 Horizontal Frame Valid Start is also set to 0x0000, so frame-valid rises coincident with line-valid. If frame-valid needs to rise prior to line-valid:

1. Add one to the value of the 0x4A–4B Vertical Active.
2. Set the value of 0x50–51 Horizontal Frame Valid Start to a value between that of 0x56–57 Horizontal Line Valid End and 0x48–49 Horizontal Count Maximum, inclusive.

0x48–49 Horizontal Count Maximum

Access / Notes: 16-bit, read-write / PDV_CLSIM_HCNTMAX

Bit	Name	Description
15–0	[no name]	The raster period, in pixel clock cycles (including blanking minus one). If, for example, you set this register to 0x04FF, the line period is 1280 pixel clock cycles. 0x48 contains the least significant bits; 0x49 contains the most significant bits.

0x4A–4B Vertical Active

Access / Notes: 16-bit, read-write / PDV_CLSIM_VACTV

Bit	Name	Description
15–0	[no name]	The number of active rasters displayed, minus one. For example, with a value of of 0x1FF, 512 lines are displayed. 0x4A contains the least significant bits; 0x4B contains the most significant bits.

0x4C–4E Vertical Count Maximum

Access / Notes: 24-bit, read-write / PDV_CLSIM_VCNTMAX

Bit	Name	Description
23–0	[no name]	The total number of rasters per frame period, minus one. For example, with a value of 0x2FF, there are 768 rasters from the start of one frame to the start of the next, assuming DMA data is available and frame triggering is not used. 0x4C contains the least significant bits; 0x4E contains the most significant bits.

0x50–51 Horizontal Frame Valid Start

Access / Notes: 16-bit, read-write / PDV_CLSIM_HFVSTART

Bit	Name	Description
15–0	[no name]	The horizontal position within the raster where the frame-valid signal starts. Frame-valid signal usually stays high for many rasters between its start and end. 0x50 contains the least significant bits; 0x51 contains the most significant bits.

0x52–53 Horizontal Frame Valid End

Access / Notes: 16-bit, read-write / PDV_CLSIM_HFVEND

Bit	Name	Description
15–0	[no name]	The horizontal position within the raster where the frame-valid signal ends. Frame-valid signal usually stays high for many rasters between its start and end. 0x52 contains the least significant bits; 0x53 contains the most significant bits.

0x54–55 Horizontal Line Valid Start

Access / Notes: 16-bit, read-write / PDV_CLSIM_HLVSTART

Bit	Name	Description
15–0	[no name]	The position within the raster where the line-valid signal starts. 0x54 contains the least significant bits; 0x55 contains the most significant bits.

0x56–57 Horizontal Line Valid End

Access / Notes: 16-bit, read-write / PDV_CLSIM_HLVEND

Bit	Name	Description
15–0	[no name]	The position within the raster where the line-valid signal ends. 0x56 contains the least significant bits; 0x57 contains the most significant bits.

0x58–59 Horizontal Read Valid Start

Access / Notes: 16-bit, read-write / PDV_CLSIM_HRVSTART

Bit	Name	Description
15–0	[no name]	The position within each raster where the internal read-from-DMA signal starts. 0x58 contains the least significant bits; 0x59 contains the most significant bits.

0x5A–5B Horizontal Read Valid End

Access / Notes: 16-bit, read-write / PDV_CLSIM_HRVEND

Bit	Name	Description
15–0	[no name]	The position within each raster where the internal read-from-DMA signal end. 0x5A contains the least significant bits; 0x5B contains the most significant bits. When read-from-DMA is false, then data is taken from the FillA or FillB registers instead of from the DMA stream (or instead of from the simulated data counter, if DATACNT in the 0x40 Camera Link Configuration A is true). If RVEN=0 in 0x40 Camera Link Configuration A, then both 0x58–59 Horizontal Read Valid Start and 0x5A–5B Horizontal Read Valid End are ignored and read-from-DMA is true whenever line-valid is true.

Interleaving Registers

When interleaving is enabled (that is, when bit 2 of 0x40 Camera Link Configuration A is set):

- Image data destined for the frame grabber is first passed through an interleaving mechanism to duplicate the data ordering of cameras with multiple taps.
- Rasters are restricted to a maximum of 4096 eight-bit pixels of active image data (DMA, FillA, and FillB). Though the interleaving registers are 16-bit registers, the top four bits of each interleaving register are unused, as only twelve bits are required to address 4096 pixels.

Interleaving is always treated using a four-tap model. Each tap has two registers: one specifies the starting pixel address within the raster, and the other specifies the delta to add to the pixel address with each pixel clock. This delta is a two's-complement signed integer, permitting negative values that fill rasters from the rightmost edge and work leftward. The four-tap model can be configured to give the same transform as a two-tap model, as in the examples below.

Table 5. Interleaving: Two- and four-tap simulation

Two taps (to fill from starting points)

To simulate a two-tap camera that fills a 1024-pixel raster line with tap 1 starting at pixel 0 and tap 2 at pixel 512, each filling pixels consecutively from its starting point, use these starting points and deltas:

	Start	Delta
Tap 0	0	1
Tap 1	512	1

Four taps

Similarly, to emulate a four-tap model, use these starting points and deltas:

	Start	Delta
Tap 0	0	2
Tap 1	512	2
Tap 2	1	2
Tap 3	513	2

Two taps (to fill from both ends)

To simulate a two-tap camera that fills in rasters from either end, with one tap working rightward from pixel 0 and the other working leftward from pixel 1023, use these starting points and deltas:

	Start	Delta
Tap 0	0	+2
Tap 1	1023	–2
Tap 2	1	+2
Tap 3	1022	–2

The interleaving registers are shown below.

0x60–61 Tap 0 Start

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP0START

Bit	Name	Description
15–0	[no name]	The starting 8-bit pixel address within the 4096-pixel raster. 0x60 contains the least significant bits; 0x61 contains the most significant bits.

0x62–63 Tap 0 Delta

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP0DELTA

Bit	Name	Description
15–0	[no name]	The delta added to the starting pixel address with each pixel clock cycle. 0x62 contains the least significant bits; 0x63 contains the most significant bits.

0x64–65 Tap 1 Start

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP1START

Bit	Name	Description
15–0	[no name]	The starting 8-bit pixel address within the 4096-pixel raster. 0x64 contains the least significant bits; 0x65 contains the most significant bits.

0x66–67 Tap 1 Delta

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP1DELTA

Bit	Name	Description
15–0	[no name]	The delta added to the starting pixel address with each pixel clock cycle. 0x66 contains the least significant bits; 0x67 contains the most significant bits.

0x68–69 Tap 2 Start

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP2START

Bit	Name	Description
15–0	[no name]	The starting 8-bit pixel address within the 4096-pixel raster. 0x68 contains the least significant bits; 0x69 contains the most significant bits.

0x6A–6B Tap 2 Delta

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP2DELTA

Bit	Name	Description
15–0	[no name]	The delta added to the starting pixel address with each pixel clock cycle. 0x6A contains the least significant bits; 0x6B contains the most significant bits.

0x6C–6D Tap 3 Start

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP3START

Bit	Name	Description
15–0	[no name]	The starting 8-bit pixel address within the 4096-pixel raster. 0x6C contains the least significant bits; 0x6D contains the most significant bits.

0x6E– 6F Tap 3 Delta

Access / Notes: 16-bit, read-write / PDV_CLSIM_TAP3DELTA

Bit	Name	Description
15–0	[no name]	The delta added to the starting pixel address with each pixel clock cycle. 0x6E contains the least significant bits; 0x6F contains the most significant bits.

Appendix C: About the Camera Link Standard

Below is a brief overview of the signals in the Camera Link interface. For a complete description of these signals, refer to your camera manual or the Camera Link specification (see Related Resources on page 7).

The pixel clock is a continuously-running clock between 20 and 85 MHz that determines the rate of data transfer. Data transfer is managed via three timing signals from the camera:

Line-valid	The line-valid signal is true during those pixel clock cycles for which valid image data is to be transferred. Line-valid then goes false for the horizontal blanking interval before starting over for the next raster line.
Frame-valid	The frame-valid signal goes true at the start of the first raster line of the image, and remains true until the end of the final raster line of the image.
Data-valid	The data-valid signal was added to handle cameras having a pixel clock slower than 20 MHz. Typically, data-valid toggles with each clock edge to allow a 10 MHz camera to be used with a 20 MHz Camera Link interface.

In addition to the above signals, a base-mode camera has 24 image data signals to the frame grabber, four generic camera control lines to the camera, and two optional UART signals (one in each direction) which can be used to perform diagnostics or to set up the camera for various modes of operation.

One of the four camera control lines is often used to trigger the camera. If the exposure time is not set through the UART, it can be determined by the length of this expose pulse instead. The other three camera control lines are often left unused.

The Camera Link interface serializes the image data plus the line-valid, frame-valid, and data-valid signals at seven times the pixel clock rate, so they can be transferred using a cable with fewer wires. Therefore, an interface using an 85 MHz pixel clock can transfer data over the cable at 595 MHz.

Medium-mode cameras use a second 26-wire cable identical to the base-mode cable to provide up to 24 additional image data signals, for a total of 48 bits per pixel clock cycle from the camera. In medium- and full-mode operation, the UART and camera control wires of the second cable are not used.

In medium mode, the CLS supports a maximum of 32 bits per pixel clock cycle.

In full mode (supported only by the PCIe version of the CLS), the UART and camera control wires in the second cable can provide an additional 24 lines of image data beyond the 48 supported in medium mode, for a total of 72 data bits – although the Camera Link specification describes cameras no larger than 64 bits.

Revision Log

Below is a history of modifications to this guide.

Date	Rev	By	Pp	Detail
20141110	0004	PH	Title pg	• Deleted unused document # (008-02781-[rev#]).
20141110	0004	PH,RH	All	• Conformed “framegrabber” and “fiberoptic” to “frame grabber” and “fiber-optic.”
20141110	0004	PH,RH	10	• Under Building or Rebuilding an Application: Added Note.
20140630	0003	PH,CH	8	• Fig. 2: Deleted “Triggering” labels on Berg pins; CLS doesn’t support triggering.
20130311	02d	PH,CH, RH	5,24	• For PCI DV CLS only – in Tables 1 and 4, clarified which types of base and medium mode are supported by this legacy product.
20130311	02d	PH,CH, RH	12	• Under main head “Operational Details” and subhead “Image Data Source”: deleted no-longer-relevant paragraph 6 (“Though targeting primarily...”).
20121203	02c	PH,CH	37	• In Tap 0 Delta register, corrected hex address to 0x62-63.
20120803	02b	PH,CH	24	• In Table 4 - Legacy (“DV”) boards: Arguments to pciload, under “PCIe8 DV CLS as frame grabber,” added “2” to both filenames: – Changed “[...]`camlk`” to “[...]`camlk2`“. – Changed “[...]`camlk_fm`” to “[...]`camlk2_fm`“.
20120621	02a	PH,CH	1	• In table, for PCIe8 DV CLS (legacy), corrected data rate to 300 MB/s.
20110515	02	PH,RH	All	• Repaginated to use continuous arabic numerals from title page to end.
20110515	02	PH,RH	All	• Added new PCIe8 “DVa” CLS and other new or updated information (e.g., included files, directives, firmware, etc.). • Implemented new terminology: Changed “digital video” to “vision” or “digital imaging.”
20110104	01	PH,RH	All	• Added new PCIe8 “DV” CLS.
20070000	00	LW	All	• Created new guide.